Agile Design Project Methodology for Small Teams Developing Mechatronic Systems

by

Stephen Dwyer

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering
University of Alberta

Abstract

A project methodology is proposed that is suitable for small teams working on the design of mechatronic systems. It is an alternative to a waterfall development method, which presents challenges when project uncertainty and risk are high. The method focuses on agility, iteration, feedback and collaboration, drawing ideas from modern agile software development methodologies while considering the difficulties of physical prototyping. Following a review of existing approaches to the design process and project management, a framework and methodology are presented. The impact of the product lifecycle and business model on the project is explored. The role of iteration is discussed in terms of team management and the design process, with consideration for the design's capacity for change. Effective methods of cross-discipline team organization are described. Project characteristics are identified that indicate whether an agile approach is appropriate. Seven principles are established to improve agility of the design. Feedback and knowledge transfer is accomplished through retrospective activities, documentation, and frequent prototyping. Rapid prototyping techniques are described for a range of mechatronic disciplines which can be used to increase the frequency of prototypes, feedback and iteration. A minimum set of live artifacts are suggested to capture the state of the project. The proposed methodology model has three phases. Project initiation helps a team evaluate feasibility and create a foundation for success in the next phase, with a focus on client engagement, planning and research, and creating baseline artifacts. Project iteration follows a Plan, Execute, Review, and Go/No-Go workflow. Task management is supported by a kanban board and feature backlog, while high level planning and time-boxed syn-

chronization with stakeholders leverage a roadmap artifact. Project conclusion occurs at completion or termination, when feedback at the project level helps the team evaluate process and technical success. Two case studies of previously completed projects are presented to understand the methods and challenges in the context of the proposed methodology. Future work includes implementing the methodology in a mechatronics design project from start to finish so that the effectiveness of the methodology can be verified.

Acknowledgements

I would like to thank Dr. Mike Lipsett for supervising this work, and for his support and guidance throughout my undergraduate and graduate studies.

I wish to thank my friends and colleagues Nicolas Olmedo and Jamie Yuen, with whom I have enjoyed learning, working, and laughing immensely during team efforts spent on a broad range of interesting and sometimes frustrating projects. It is these projects that inspired this thesis.

I am also grateful to all those who have helped me develop my engineering skills over the years at the University of Alberta and beyond, whether in the classroom, the machine shop, the lab, or as part of extracurricular activities.

Most importantly, I wish to express my heartfelt gratitude to my partner Jaclyn, whose patience, love, and support continue to make all of my endeavors possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 What is Mechatronics?

Mechatronics can be described as the integrated design and development of products and systems incorporating mechanics, electronics, controls and computer engineering [25]. The term "mechatronics" originally emerged from the Yaskawa Electric Corporation in Japan, in the late 1960's [105, 19]. It combined the two words "mechanics" and "electronics", though has been broadened to now encompass software and computation [105, 19]. As the fields of computer and electronics engineering have evolved, so too has their importance in the function of an ever expanding field of products and systems. Take for example the automobile. While historically dominated by mechanical design, vehicles of today contain numerous microcontrollers and circuits tasked with both critical functionality (like an ECU controlling fuel injection and spark timing or brakes) and mundane features (like recalling an individual drivers' powered seat adjustments) [21]. To define the behaviour of and carry out computation within these electronics modules, tens of millions of lines of software code may exist in a modern consumer car [21]. Vehicles are far from the only application area that benefit greatly from mechatronics. Manufacturing and industrial processes are continuing to benefit from automation and software-augmented monitoring. Robotics are applied to ever more

complex tasks in industrial and commercial settings, while also rapidly expanding into the consumer space for work (household cleaning [47]) and entertainment (drones [106]). An ever growing variety of consumer products combine physical hardware, onboard firmware, internet connectivity, and a software package for enhanced functionality (security cameras, bluetooth locks). Though some of these products may be missing mechanics driving motion, the same engineering principles apply to both. Such products with a minimal focus on mechanics are typically referred to as embedded systems. In this work, the term mechatronics will be inclusive of embedded systems. In contrast to some traditional fields of engineering, synergistic understanding and cooperation between the variety of disciplines involved in the design and development of mechatronics systems is important in evaluating and applying the tradeoffs necessary to achieve an economical, efficient and innovative result. This multidisciplinary development needs the support of effective design and project management methodologies and processes.

1.2 Design Process and Project Management

The profession of engineering encompasses a wide range of activities. This includes both engineering design as well as project management. Development of a new product or system or making improvements to an existing one is often encapsulated as a project. One of the primary technical aspects of such a project is design. Project management incorporates a business viewpoint, and is used to obtain value out of the technical and scientific facets of engineering.

The ABET (Accreditation Board for Engineering and Technology, Inc.) definition of engineering design: "Engineering design is the process of devising a system, component, or process to meet desired needs. It is a decision making process (often iterative), in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective." [35]

As another example, Dym and Little state that engineering design: "is a systematic, intelligent process in which designers generate, evaluate and specify designs for devices, systems or processes

whose form(s) and function(s) achieve clients' objectives and users' needs while satisfying a specified set of constraints." [29]

Each of these emphasize the technical side of the process, but hint at the need for additional considerations by noting that design should "convert resources optimally" and "satisfy...constraints". Achieving value with design activities is supported by project management to guide and manage these activities.

A project is "a one-time activity with a well-defined set of desired ends", according to Dym and Little [29]. The obvious ends in a design project include the finished design deliverables that meet requirements. There may be other less concrete goals reflecting other values of the project or organization, such as building good customer relationships. According to the Project Management Institute [46], a project is temporary and unique. It has a defined beginning and end, as well as defined scope and resources. It looks to accomplish a single goal with a specific set of operations that together are not routine.

Managing is composed of four activities: planning, organizing, leading and controlling [29]. From a project management standpoint, planning involves understanding the time and resources allocated to a project and defining the scope. Organizing looks to apply resources (especially human resources) to tasks and activities forming the scope, while leading tries to motivate the team to make progress towards goals through understanding the task at hand and cooperative and fair division of work. Controlling a project involves comparing progress against planned goals and taking corrective action to meet these goals.

From this, project management may be described as "the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements." [46]. Alternatively, it is the "activity of organizing available resources to accomplish the tasks required to meet an objective." [19]

A wide variety of methods of project management have been developed and studied throughout the evolution of business and engineering in pursuit of the most effective. New or adapted methods

are regularly suggested. The definition of success may vary across methods and models. Practices, activities, tasks, tools, artifacts and work product all support a project lifecycle or design process methodology or model. The wide range of methods have varying contexts, with some focused on business aspects of design project management and others on the technical design aspects.

Project management methodologies typically build upon an underlying philosophy or set of principles. The two ends of the spectrum rest on either a predictive approach or an adaptive approach to project management. Predictive approaches are characterized by significant emphasis on planning and estimation early in the project. These approaches are considered more traditional, and are often found in construction and mechanical design. Rigorous software development methodologies are also focused heavily on up-front planning. Adaptive project management is best exemplified in Agile software development methodologies [44]. Less consideration is placed on creating a plan early in a project, as it is expected to lack precision. Agile processes build on the premise that requirements are uncertain at the beginning of the project, which makes predicting schedule difficult, and that change should be expected and embraced. The focus is placed on obtaining feedback on the process early and often, to counteract this lack of precision and uncertainty.

While engineering design and project management may be studied separately, it is more appropriate to consider them together, as they are tightly coupled by a number of elements. Attributes of, constraints on, or decisions made for one will almost certainly impact the other. Traditional engineering education experienced by the author has focused on the Waterfall design process and a project management approach that prioritizes planning and control based on scope, spending and schedule.

1.2.1 Traditional Waterfall Design Process

One of the simplest design process models is the Waterfall model. The basic steps in the model are Specification, Preliminary Design, Design Review, Detailed Design, Design Review, Implementation, and Review, as illustrated in Figure 1.1 [69]. A more detailed waterfall process model was

historically applied to software development, as shown in Figure 1.2 [13]. The linear, sequential progression through design phases does not readily support the iterative nature of design. Problems that arise late in the design may prove difficult and expensive to correct, depending on the nature of the product. This is modeled by the hockey stick curve (Figure 1.3), relating the time spent on a design project and the cost of fixing issues [69]. The waterfall model is based on the assumption that each step can be completed before moving to the next. This would require all relevant information to be known, which is simply not feasible. Also, while the hockey stick curve reflects the cost of fixes when the design or system is difficult to change, it is less representative when improvements are easy. Projects will range in their capacity for change. Designs in physical domains will be more resistant, especially examples of large structures or with significant production infrastructure. Solutions built in software are examples where change is readily achievable.

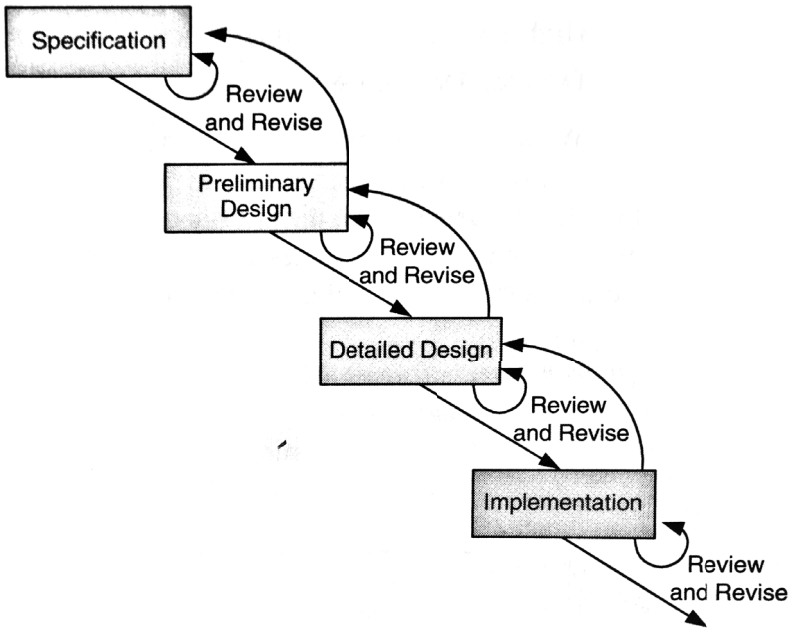


Figure 1.1: Traditional waterfall design process model. Adapted from Figure 9.8 of [69]. Copyright 2008 © John Wiley & Sons, Inc. All rights reserved. Reproduced by permission of the publisher.

1.2.2 Traditional Project Management Triangle

Dym and Little describes tradeoffs in engineering projects as a balance between Scope, Spending and Schedule (3S's), while Alite identifies Quality as an alternative to Scope [29, 5]. This balance is interpreted as a tetrahedron in Figure 1.4. In a typical project with limited resources, a primary purpose of project management is to manage the tradeoffs between these priorities.

Schedule maps the activities in a project to a timeline. Understanding the estimated and actual progression of activities and the overall project are necessary in coordinating with the project within its environment. A project will likely need to fit into higher level business goals and activities, meet externally imposed deadlines, and interact with outside constraints such as procurement, manufacturing or shipping times. Commonly, projects need to be completed within a specific timeframe in order for it to be successful and provide value to stakeholders. Failing to do so may have economic or other impacts, which is known as the 'cost of delay' [6].

Spending is relatively straightforward, and is also referred to as cost or budget. Any project (or any tasks or actions supporting business activities) may be associated with a monetary expenditure. For example, the parts or components purchased to build a prototype in a product development project clearly add to the cost of the project. Costs in a project include manufacturing costs, labour costs (direct project team member salaries, etc.), and overhead costs (cost of doing business, support staff). It is important to note the difference between the costs associated with the product or system (manufacturing, bill of materials, i.e Cost of Goods Sold (COGS)) compared to the NRE or non-recurring engineering costs, the costs associated to carrying out the project and doing the design work. Different types of cost will lead to different tradeoff decisions, depending upon the stage and type of project.

As previously noted, the third corner of the triangle is sometimes represented as Scope, while others use the term Quality. Both are used to represent the value of the outcome from a different standpoint and are tightly coupled. Scope refers to the breadth and depth of activities or features like a list or inventory [65]. Quality evaluates the value or worth of the final outcome intrinsically

(e.g. regardless of the project size or quantity of work, the quality may be high or low) relative to similar outcomes [63]. Quality and Scope may also be considered as competing constraints, whereby given fixed resources, increasing the scope of a project will likely lead to lower quality as the resources are spread across a larger volume of work. Conversely, applying the same resource set to a smaller scope will allow more effort to be placed on each feature or component, improving quality. This does not consider other more complex factors impacting quality. Beyond simple constraint management, it should be possible to achieve greater quality and value without trading off resources or scope with an effective team, suitable practices, and a good culture.

Projects are resource limited. At an abstract level, resources may be reduced to time and money. These are concretely mapped to schedule and spending. The project output is dependent on the availability of these resources. The management of a project will involve making tradeoffs between these to achieve a valuable result. A variety of priorities will impact tradeoff decisions. Some examples of simple tradeoffs include:

- If a project prioritizes schedule and must be completed by a specific date, some approaches to achieve success include:
  - Controlling or reducing the scope to reduce the number of work hours (design effort) required
  - Increasing spending to allow for a larger team that may accomplish the same work in reduced time without affecting scope
  - Increasing spending to expedite manufacturing or reduce shipping times
- If a project prioritizes cost savings, it may be appropriate to:
  - Increase time spent optimizing the design for manufacturing (in a COGS limited project)
  - Relax specifications and requirements to potentially reduce either design effort or COGS, or both
  - Outsource design effort by incorporating turnkey third party modules or subsystems (in an NRE limited project)
- If a project prioritizes high quality or greater scope, consider:

- Increasing team size to achieve more under a similar schedule
- Extend timelines to give sufficient time for careful or more detailed design and review
- Provide resources for a team to obtain training to improve quality of work if experience with the scope is not sufficient

Effective project management will make tradeoffs that reflect the priorities and goals of the project, while efficiently managing resources to provide the greatest return on resource investment. In other words, minimize the resource use while achieving appropriate scope at high quality.

1.2.3 Uncertainty and Risk

Uncertainty indicates a state of being unknown or not definite, that one may not be confident in or sure of something [66]. It is present in any design project—in fact, any business or technical activity or decision is likely to have a level of uncertainty. Uncertainty manifests due to incomplete or imprecise information available.

Risk refers to the possibility that an unwelcome situation may occur [64]. It stems from an uncertainty in the expected outcome of an event. From a technical standpoint, risk considers the combination of the probability of an event and its consequences, most often in a negative sense [88]. In a design project, the risk present may fall into a number of categories. Outcome- or stakeholder-oriented risk categories include occupational health and safety risks, environment and public health risks, legal and financial risks, and reputation or societal risks [88]. Source-oriented or internal risk categories include technical risks, organizational risks, market risks, and external risks.

One underlying goal of project management is to understand and manage risk within the project and organization. Many of the problems and decisions encountered can be framed in a risk management context, whereby risk is evaluated and controlled. In the context of a mechatronics design project, these risks may relate to meeting schedule or budget constraints. It may also relate to meeting specifications or achieving functionality that is difficult from a technical standpoint. The product or system created during a project should be designed to minimize risk to stakeholders. As a design

evolves, the team responsible should consider risks and incorporate mitigation through prioritization or design features.

Even though the design process and project management may be treated separately, it makes sense to consider them together in the context of engineering design projects. The management of the project as well as the technical decisions made during the design process must both be considered when making tradeoffs in the project over a range of categories. Further coupling is uncovered when managing risk in the design project. In this work, the term "design project" will be used to refer to both the design process as well as the project management, given the dependency between them. The previous sections provide only a brief insight into the concepts of design project management aimed at providing some context for further discussion. The topics mentioned have significant breadth, far beyond the scope of this work.

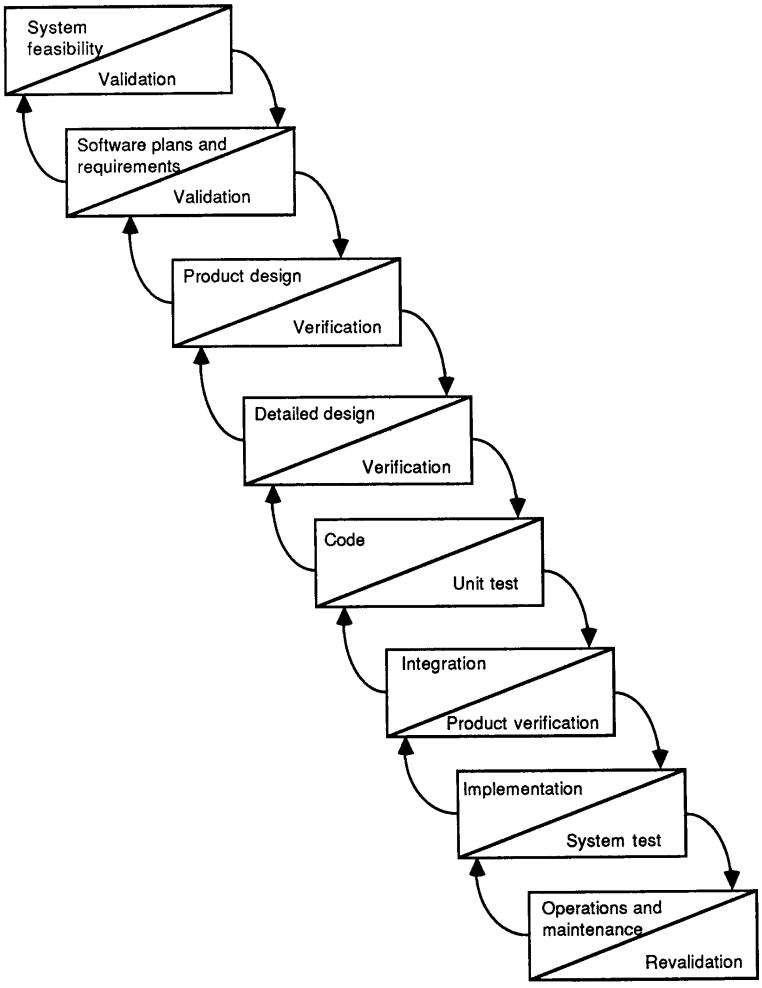


Figure 1.2: A more complex software development waterfall process model. Adapted from Figure 1 of [13]. © 1988 IEEE. Reproduced by permission of the publisher.

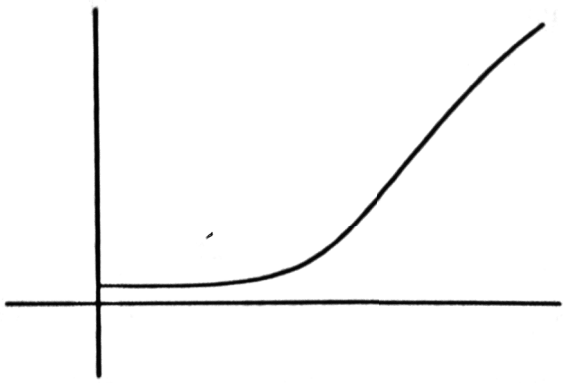


Figure 1.3: Basic hockey stick curve relating project progress (*horizontal axis*) to cost of fixes (*vertical axis*). Adapted from Figure 9.7 of [69].

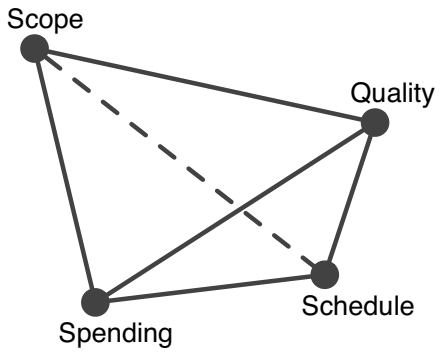


Figure 1.4: Project mangagement triangle [tetrahedron] of tradeoffs.

1.3 Systems Engineering

Historically, cross-disciplinary systems design and development was primarily the domain of large and complex military, aerospace, telecommunications or software projects [81]. Systems engineering was developed to address the challenges of coordinating and managing design and development teams working on these projects [20]. It is typically viewed as a separate branch of engineering, whereby a system architecture and overall design, as well as requirements, a project structure and a management plan will be developed at the project outset by dedicated systems engineers [1]. Upon completion of the architecture, subsystems are passed to specialized, discipline-specific teams and managed in a more traditional sense, which is reflective of the emphasis placed on specialized training and education. In another view, systems engineering is used across a full project lifecycle, addressing both technical and management activities with the goal of balancing all project objectives [2]. While mechatronic systems would appear to benefit from this approach, the original target domain for systems engineering is characterized by very large projects with huge teams, extended timelines and massive budgets. Thus, the practices and methods may be less applicable to small projects and small teams. Compare the complexity and project size between two transportation systems, the space shuttle project and an electric-assist bicycle. Systems engineering is not necessarily a solution to all cross-disciplinary design projects, and alternatives warrant investigation.

1.4 Motivation

Copperstone Technologies (CST) is a small design and product development engineering firm in Alberta, with a focus on field robotics and remote sensing [23].[1] CST's small cross-functional development team focuses on low volume or prototype systems, as well as small volume industrial and commercial products. Most projects are based on mechatronics content, or a subset thereof, including embedded systems, software and web application development, mechanical and electronics

[1] As a co-founder of Copperstone Technologies, the author has a deep understanding of the company's vision, practices, successes and challenges.

design, and control and communications. System level design and understanding is a key component to the successful integration of cross disciplinary projects, while rapid prototyping and a tendency towards lean methods push the team to deliver value quickly. Furthermore, the team and organization is sufficiently small that most activities (design, management, test, etc.) are carried out collaboratively and concurrently.

Currently, CST's approaches are informal and ad hoc, primarily drawing on methods discussed during undergraduate education, though there is a desire to explore lean and agile alternatives in greater detail. Preliminary exposure to these methods in a primarily software context has shown a clear synergy with the team's culture. Frustration and poor performance with methods traditional to a mechanical engineering background further motivate an evolution in methodology. As a member of the Copperstone Technologies team, the author sees value adopting an agile, customizable design project methodology to improve the team's performance.

1.5 Objectives and Scope

It is possible to develop a design project methodology suitable for small teams working on primarily early stage mechatronic systems development that incorporates agile principles while considering the challenges of physical prototyping. The methodology should provide a substitute to the traditional waterfall approach of development that has limitations and shown poor performance in the experience of the author. Alternative elements are needed to replace predictive planning tools like Gantt charts, fixed budgets and rigid schedules. The proposed methodology should leverage iteration to support a team's goals of agility, innovation, and minimalism. It should be applicable to the varied disciplines that together compose mechatronics in an integrated fashion.

To meet this objective, it is necessary to understand what comprises a design project methodology suitable for the domain of interest. A review of a selection of existing approaches will be made, considering aspects of both project management and the design process.

Based on this review, a framework will be developed that provides guidance for the elements,

practices and philosophy of a methodology. Topics of concern and importance when applying agile principles to a methodology will be studied. This includes understanding the project lifecycle, iteration and project characteristics. The organization of a design team will be considered, along with the role of risk management. Design approaches that improve agility will be identified, and incorporating deliverables like prototypes and documentation will be examined.

The framework will be used to outline an agile methodology model. Ideally, it will be flexible enough to apply to a range of projects with different characteristics. The phases of the methodology process will be described with a focus on iterative progress. Practices will be described that help the team work together to plan, review and make decisions. Finally, key artifacts will be identified that can appropriately capture design project information.

Two previously completed projects will be reviewed within the context of the framework and methodology. These case studies will compare how the teams executed the projects compared to the proposed approach. Areas where the teams were successful will be identified, specifically those that demonstrated use of techniques suggested in the framework and methodology. Challenges will be considered that may have benefitted from implementing the suggested techniques as well.

1.6 Organization

This introduction is followed by a brief review in Chapter 2 of engineering design process and design project management methods. In Chapter 3 the framework is established that lays the foundation for the proposed approach with the description of relevant key principles and topics. Next, the agile methodology model is described in Chapter 4 that is suitable for small teams working on mechatronics design projects. Chapter 5 presents two case studies of previous mechatronics projects would have benefitted from an appropriate methodology. Finally, the work is summarized in Chapter 6, along with areas for future study.

Chapter 2

Literature Review

Engineering design and project management has been studied countless times and continues to be an area of active research. Academically, research includes creating and modeling design processes for analysis and optimization, with the goal of improving success rates. A focus is also placed on design project education, suitable approaches and ways to improve students' understanding of design and project management. In industry, companies continually try to improve processes and increase productivity. Internal processes are created, adapted and eventually mature. These processes may be informal, or very complex and heavily documented. Often, this is kept within an organization as part of their competitive advantage. When insight into an organization's methods are made available, it is often in blog posts, white papers, and other "grey literature". Overall, design project methodologies continue to evolve. Companies and researchers explore design projects with both depth and breadth from many angles. The common goal is always trying to achieve a better method, each attempting to present their view on the most effective approach.

First, a review of what might be considered "traditional" engineering design project methods will be presented. Most often presented as textbooks, the authors look to provide a discipline independent view of the design process. Next, these ideas will be explored further in a domain specific context. Texts covering design of mechatronic and embedded systems will be covered in particular. This will

be followed by a look at agile philosophy and methodologies. Subsequently, previous approaches to modeling, categorizing and quantifying the design process will be considered. With these other ideas in mind, ideas for and examples of introducing agile into disciplines outside of pure software will be explained. It is noted that the vastness of the subject area led to only a small sampling of design texts being reviewed.

2.1 Domain-Agnostic Design Project Methods

Texts often cover the design process model with steps and phases, include some discussion of managing the project and the design team, and may also outline a variety of additional techniques or methods that help an engineer complete each part of the model. Traditional interpretations of the design process are typically presented in a way that appears to be domain-agnostic. This is often not the case, and most of the following appear to target the design of physical products that will undergo manufacturing. There is still considerable insight into the design process presented that is applicable across a wide range of project domains.

2.1.1 The Engineering Design Process

Ertas and Jones [35] use the diagram shown in Figure 2.1 to describe the steps of the engineering design process. They note that the process may need to be modified to suit an individual project, which may include eliminating some steps. Their process is targeted at projects of significant magnitude, and the need to modify the process "is especially true for design efforts accomplished within small organizations, in which the design process is less formal."

The authors describe the design process in an incremental, stepwise manner, similar to the basic predictive waterfall approach. To complement this, Ertas and Jones highlight a number of traditional design and management tools throughout. It is also clear that the method is considered primarily for projects involving the design of products or systems that require manufacture and

assembly. This is supported by the discussion of production, design for manufacture and assembly, product testing, distribution, maintenance, and design life.

Even though the foundation of the process is a waterfall approach, the authors discuss a variety of approaches and considerations that, if framed in a slightly different context, very much reflect agile philosophies and practices. This is demonstrated clearly in a number of observations, particularly in the steps prior to the detailed design phase. First, the diagram shows the importance of iteration in the design process, during the conceptualization and feasibility phases, and then again during the preliminary design and detailed design and testing phases. It is suggested the detailed requirements for the system should be established before or during the Preliminary Design phase, rather than as a first step. This is to highlight the difficulty in knowing all requirements prior to the start of the project, and need to explore the design space early during conceptualization and feasibility to understand the project and possible solutions. The authors suggest that while requirements are typically considered fixed, they should undergo continuous review and validation until the design is finalized. This further emphasizes the fact that design is a highly iterative process where requirements can not be known with certainty at project outset. During requirements review, it is especially important to consider cost and the difficulty of change or achievement as the design matures. Ertas and Jones encourage that all stakeholders be involved in requirements, and all team participants both understand and agree to the requirements. This suggests requirements are primarily a tool to communicate what will make the design successful amongst the team.

As the design progresses, the authors maintain the importance of communication and iterative design refinement. It is suggested preliminary design be carried out by a small, tightly-integrated group with good communication. Ertas revisits a previous work and outlines several abstract design concepts, quoted here:

- "Impossible to specify all requirements of a system correctly at the outset of its development. Iterative refinement is needed."
- "Feedback is needed, and rapid and successive feedback promotes the refinement process."

- "Computer models and other automated tools must be an integral part of the refinement process. Computer models of the end product—including subsystem and component interfaces—that identify data, function, and control parameters can be an invaluable tool in establishing subsystem and component requirements and in providing a mechanism for managing these requirements during the detailed-design phase of the process."

Finally, the authors briefly introduce concurrent engineering as a way to further integrate product design and product manufacturing.

From a management standpoint, the authors' ideas continue to be consistent with agile and lean philosophies, highlighting quality management methods. An emphasis on quality in addition to schedule and cost, the importance of employee work ownership, and the need for feedback and adjustment in management are all examples of this.

Throughout the discussion of design process and project management, Ertas and Jones suggest a variety of tools to support a project. Examples include: brainstorming, hierarchical design and architecture drawings, formal requirements, work breakdown structures, gantt charts, PERT networks, critical path analysis, engineering change orders, CAD/CAM, House of Quality, objective trees, decision matrices, decision trees, functional cost analysis, and Failure Modes, Effects and Criticality Analysis. Most are traditional predictive project management tools, but again, the authors point out underlying goals of using these tools, again in line with agile thinking. For example, they mention use of a formal Work Breakdown Structure may not be warranted for small projects. Their support of minimizing process ceremony and complexity is clear: "If the effort required to make the tool useful approaches the benefit that the tool provides to the program, then utilization of the tool in that application is questionable."

2.1.2 Engineering Design

The design process model proposed by Dym and Little [29] is built on a traditional linear, waterfall-style model. They use both a descriptive and prescriptive design process model, attempting to both

describe elements (or stages) of the design process, while also prescribing tasks to be done during the design process. The model is illustrated in Figure 2.2a. The linear foundation is clear in that while the authors explain the necessity of iterating upon and refining a design with feedback, it is not emphasized in the model. Basic feedback is shown in Figure 2.2b. An interesting feature of the model is that the authors identify pre- and post-processing stages as the problem definition and design communication phases, respectively.

Dym and Little suggest and describe numerous tools and methods useful in various stages of the design process. They note the target audience for the tools described is small design teams. The model essentially encourages the tasks at each step to be carried out in order.

The authors suggest a few strategies to keep in mind throughout the design process. Least Commitment recommends designers do not commit to a particular concept or configuration until forced to by the exhaustion of additional information or alternate choices or time. The Decomposition approach breaks a larger problem into sub-problems (or ideas or entities). They also suggest numerous formal methods to support the process at different stages, especially during problem definition and concept design. Objective trees can help clarify and understand the client's project statements, while pairwise comparison charts (PCCs) help prioritize objectives early. Establishing formal metrics help assess a design compared to the objectives and functional analysis is used to more precisely identify what a design must do. The performance specification method helps create evaluation criteria by identifying attributes and hard performance specifications. A morphological chart helps explore design alternatives that will provide the means for a specific function. Different styles of evaluation matrices aid in selecting the preferred design solution for continued development.

Acquiring information is crucial in understanding a problem space and constraints or objectives, and the authors suggest a number of methods including: literature reviews, user surveys and questionnaires for market research, focus groups, informal interviews, structured interviews, brainstorming, synectics or analogies, competitive product benchmarking, and reverse engineering.

Later in the design process, Dym and Little suggest a number of methods for analyzing information and testing outcomes. To evaluate whether a design's objectives are achieved, formal metrics can

be developed and measured against. The functional performance should be described in engineering terms with a statement of requirements and specifications. Field or laboratory experiments provide useful information, while defining outcomes that are sufficient or appropriate help to determine if a concept should be accepted or rejected during proof of concept testing. Rapid prototype development can provide useful insight, even if it has limited functionality to reduce cost and development time. If prototype development is difficult due to cost, size or hazards, simulation with an analytical, computer or physical model may be possible. The authors cover in detail some methods of creating mathematical models as well as materials selection and techniques for physical prototype fabrication.

Obtaining feedback on the design is important to ensure it meets the requirements and to find out ways it may be improved. Some methods of obtaining feedback include: regular meetings with clients and users, formal design reviews with clients, users and stakeholders, focus groups, and beta testing. In some projects, public hearings may be required.

A linear model similar to the design process model is suggested for the management of a design project, summarized as a list:

1. Project Definition (or Scoping)
    (a) Client's feasibility study
    (b) Orientation meeting (Project kick-off)
    (c) Scope defined; budget and schedule limits set
    (d) Team charter written
2. Project Framing
    (a) Project team set
    (b) Project tasks developed: Work Breakdown Structure established
3. Project Scheduling
    (a) Assignment of tasks: Linear Responsibility Chart established
    (b) Resources and tasks scheduled
    (c) Budget defined

4. Project Tracking, Evaluation, and Control
    (a) Work, time, and cost monitored
    (b) Actual and planned work compared
    (c) Trends analyzed
    (d) Plans revised as needed

From an abstract view, Dym and Little state that project management is balancing the scope, spending and schedule of a project. They point out that design projects are more difficult than other project types like construction due to the many uncertainties and ambiguities in these three categories. The authors discuss that teams and leadership are central to engineering design projects, and introduce the group formation model suggesting there are five stages in group development: forming, storming, norming, performing, and adjourning. Suggested tools for project management include team charters, work breakdown structures (WBS), linear responsibility charts (LRC), team calendars, Gantt charts, activity networks, budgets and percent-complete matrices (PCM).

The design process and project management models presented by the authors are very linear in nature, with limited emphasis on iteration and feedback. The process and method is very clearly laid out, almost algorithmic in nature, and favours doing things once and in order. However, they argue a number of key ideas underpin successful design projects. First, good communication is critical, especially between designer, client, and user. This might be extended to include all stakeholders. Further, an appropriate "language" must be found for different types of communication, e.g. words, images, numbers, rules, properties, drawings, etc., especially when clarifying client objectives. Second, asking increasingly detailed questions throughout the process helps to drive each step, and continually refine the problem and solution. This is indicative of the importance of iteration. Another point made is that their model of project management may actually be at odds with the open endedness of design. Finally, the authors only briefly mention that concurrent engineering and design for manufacture may be important for success. They introduce concurrent design as trying to actively work with a team of stakeholders to collectively design an artifact such that fabrication and other lifecycle stages are considered.

2.1.3 Engineering Design

Dieter and Schmidt [27] reiterate most of the ideas and tools suggested by other authors [35, 29, 43]. The content continues to target development of physical products. There is a trend toward more complex and process or algorithm based methods, such as quality function deployment (QFD), the theory of inventive problem solving (TRIZ/TIPS) and axiometric design. The design process is grouped into three main phases: conceptual design, embodiment design and detail design. Steps and some suggested artifacts or tasks are summarized as follows:

- Conceptual Design
  - Define Problem
    - Problem Statement
    - Benchmarking
    - QFD
    - PDS
    - Project Planning
  - Gather Information
    - Internet
    - Patents
    - Trade
    - Literature
  - Concept Generation
    - Brainstorming
    - Functional Decomposition
    - Morphological Chart
  - Concept Evaluation
    - Pugh Concept Selection
    - Decision Matrices
- Embodiment Design

        - Product Architecture
            - Arrangement of Physical Element to Carry Out Function
        - Configuration Design
            - Preliminary Selection of Materials and Manufacturing
            - Modeling and Sizing of Parts
        - Parametric Design
            - Robust Design
            - Tolerances
            - Final Dimensions
            - Design for Manufacture
- Detail Design
    - Detailed Drawings and Specifications

This is suggestive of a very linear and waterfall-like process similar to the models presented by others. On the other hand, Dieter and Schmidt describe a basic module of the design process, first proposed by Asimow [8]. This module illustrates a feedback loop: *Specific and General Information* → *Design Operation* → *Outcome* → *Evaluation*, where successful evaluation indicates a transition to the next step and otherwise the feedback loop flows back to the beginning. This indicates a clear notion that iteration provides a basis for most design activities, and this should be applied to the phases and steps suggested in the model. Beyond just the design, additional project phases are suggested that may include manufacture, distribution, use and retirement.

As a basis to the design process model, Dieter and Schmidt present a basic problem-solving model: definition of the problem, gathering of information, generation of alternative solutions, evaluation of alternatives and decision making, and finally, communication of the results. In addition, they compare the design method to the scientific method, after Hill [45], as shown in Figure 2.3.

It is suggested design can be broken into different categories: original/innovative/disruptive design, adaptive design, redesign or variant design, selection design and industrial design. Original design is rare and often includes invention. It has the potential to significantly disrupt markets.

Adaptive design is more common, and leverages synthesis of an existing solution by adapting it to a novel application. Redesign delivers a variant or improvement of an existing design, which is a common activity. Selection design refers to the process of identifying standard components that meet project specifications. Industrial design is typically more artistic, with the goal of improving human interaction and appeal. Products in each design category will have different activities, risks and goals.

The authors explain that a quality, cost-effective product begins at the design phase; poor decisions during design lead to increased cost and issues in manufacturing and life-cycle. They note that "[t]he design process should be conducted so as to develop quality, cost-competitive products in the shortest time possible." It is also suggested that many design projects may not reach completion due to a lack of economic or technical feasibility. In these cases, new information is created during the system design process, and this information represents experience. It has future value if it can be stored in a retrievable form. A successful and quality design should be evaluated against achievement of performance requirements, total life-cycle issues, and regulatory and social issues (like the environment). This holistic view considers more fully all stakeholders and external constraints in a project.

The authors argue that design reviews are important to ensure success in a design project. A design review "is a retrospective study of the design up to that point in time. It provides a systematic method for identifying problems with the design, determining future courses of action, and initiating action to correct any problem areas." Design reviews should be attended by a variety of stakeholders, and cover both technical and business aspects of the design. It provides an opportunity for monitoring and feedback from both a design acceptability and project management standpoint. It is suggested formal design reviews be carried out between phases. The design is easy to change after conceptual design, interfaces are important after embodiment design, and the product should be ready for manufacture after detail design.

2.1.4 Engineering Design Process

Haik and Shahin [43] identify five basic principles or steps common to any design process: requirements, product concept, solution concept, embodiment design, and detailed design. These are further broken down into a number of steps, as illustrated in Figure 2.4. The authors present several prior process models, including one from Pahl and Beitz [67], which they suggest is the basis for all modern systematic design processes. This is shown in Figure 2.5. They discuss that design processes are evolving to focus on the importance of good requirements and specifications. There is also a clear argument that iteration is very important during engineering design activities; each step is iterative with decision points needed between them—some oscillation may occur. However, the underlying process flow still presents in a very linear fashion. Haik and Shahin observe that a design process should guide a designer to achieve goals, but not hinder creativity. There is limited discussion of project management beyond the suggestion of using several standard tools, namely Gantt charts and critical path analysis in the form of program evaluation and review technique (PERT) networks or critical path method (CPM). The overall approach to design is generally targeted at physical product development.

The first part of the process model is to identify needs at a general level and understand the market driving the need. Needs are then expanded into customer requirements, after which they should be prioritized and organized. Suggested tools for accomplishing this includes the use of a customer needs statement, a clear rating system for assigning importance, and a hierarchical objective tree to organize requirements. Needs and requirements should be clear and concise, but also solution independent. The authors imply that project failure is likely if the market is not well understood in an objective manner. Methods for conducting market research are presented. One additional tool recommended is the Kano Model for customer needs assessment, which looks to categorize requirements into five types: must-be, one dimensional, attractive, indifferent, or reverse (illustrated in Figure 2.6). This builds on the basic idea that some needs are critical, while others are just desirable.

Continuing the theme of understanding the problem in a solution neutral manner, requirements

are used to identify functions and create a functional structure through decomposition. Functions are described as verb-noun pairs and help map customer requirements into engineering actions or terminology, which can be used to aid in the development of specifications. Decomposition identifies boundaries around a function, understanding the function's environment, inputs, and outputs, and breaks down a function at one level into sub-functions at a lower level. The final function structure exposes a solution independent internal functionality by considering the flow of material, energy, and information between sub-functions in a tree form, an example of which is shown in Figure 2.7. The goal of creating such an architecture is to eliminate duplicate or unneeded functionality, provide a greater understanding of the system complexity, and improve system modularity. Any number of alternative architectures are possible, and iteration is likely required. A potential challenge with this approach is balancing the depth of the function tree and maintaining a solution-neutral result.

The authors use the function structure and customer objectives to create specifications—limits on the functions and objectives. The goal is understand ways in which the function and objectives may be measured and identify metric/value pairs suitable for such a measurement. Two methods of mapping requirements and functions into specifications are suggested: the Performance Specification Method and Quality Function Deployment (QFD). The Performance Specification Method focuses on defining the required performance of a design solution, but avoids describing solution-specific means of achieving that performance. The level of generality in these specifications must balance specific requirements with the size of the design space. Performance requirements the design should apply to relevant attributes. These attributes may be obtained through analysis of a function structure described above, or customer or market requirements. Pahl and Beitz present a list of possible attributes from which appropriate ones can be selected for the project at hand, found in Figure 2.8. QFD is composed of four stages that link a set of requirements to an implementation. For example, the first stage links customer requirements to technical requirements to help understand the relationships between them, along with the possibility of evaluating competitor solutions. QFD is typically applied to large, long term projects due to the significant effort involved in the process. There are many resources available on explaining the QFD method, including the "Practical Manual

of Quality Function Deployment", by Maritan [55].

Haik and Shahin suggest a fairly standard set of methods and tools for developing and evaluating concepts. Concept development may use brainstorming and morphological charts, and approach the task by following the function structure developed previously. Evaluating concepts can be done with a decision matrix or Pugh's evaluation matrix. It is noted that when working with a team, individual evaluation followed by group discussion and consensus may be more effective than a direct team evaluation.

The authors address safety considerations during embodiment design, as well as design for "X", where manufacturing, assembly, the environment, or other life-cycle elements are considered. Safety, reliability and risk analysis can be supported by fault trees and failure modes and effects analyses (FMEAs) [36, 94]. They recommend that careful cost analysis is performed as part of detail design to support cost reduction, and outline elements of and approaches to cost estimation. Of particular importance is decisions around building versus buying components.

During embodiment and detail design, it may be useful to construct and test hardware (experiment) to verify the design concept or analysis. Haik and Shahin identify mock-ups, models and prototypes as different approaches that may be used together, or independently. Mock-ups are non-functional scale representations. Models try to predict behaviour by representing the design with mathematical similitude, and may be further categorized as true models, adequate models, distorted models and dissimilar models. A true model looks to reproduce the design geometry exactly, while adequate models target the testing of specific characteristics. A distorted model violates some design conditions that are difficult to satisfy, and dissimilar models try to give insight into behavioural characteristics through appropriate analogies do not resemble the design embodiment. Prototypes of a design can be very costly, but give the greatest insight.

2.1.5 Theory of Technical Systems

Eder [30] provides a brief discussion of Hubka's Theory of Technical Systems that has been in development since the mid-1960's, which is compared to methodology of Pahl and Beitz [67]. While Pahl and Beitz's methods are targeted primarily at mechanical engineering, Hubka worked to develop a theory that is applicable to all engineering devices, or "technical systems" (though typically still physical products). The theory is based on the concept of a general "transformation system" that includes a transformation process (intended purpose) and interaction with five operators: environment, humans, technical systems, information systems and management systems, with the goal of transforming a specific operand (material, energy, information, living matter) in structure, form, location or time. Within the theory, a technical system moves through a lifecycle composed of multiple transformation systems, whereby operators or subsystems are also considered transformation systems in a hierarchical manner.

When compared to Pahl and Beitz, Hubka's theory and methods seem more complete. Hubka formalizes the lifecycle and also a problem solving cycle. Also incorporated are formal lists classifying requirements, technical system properties, and transformation process properties. To support the theory and methods, numerous case examples have been developed as well. However, Pahl and Beitz's methods have proven popular (especially in Germany) due to adoption by academic institutions and professional organizations, while Hubka's methods have generally not been presented to students or adopted by industry.

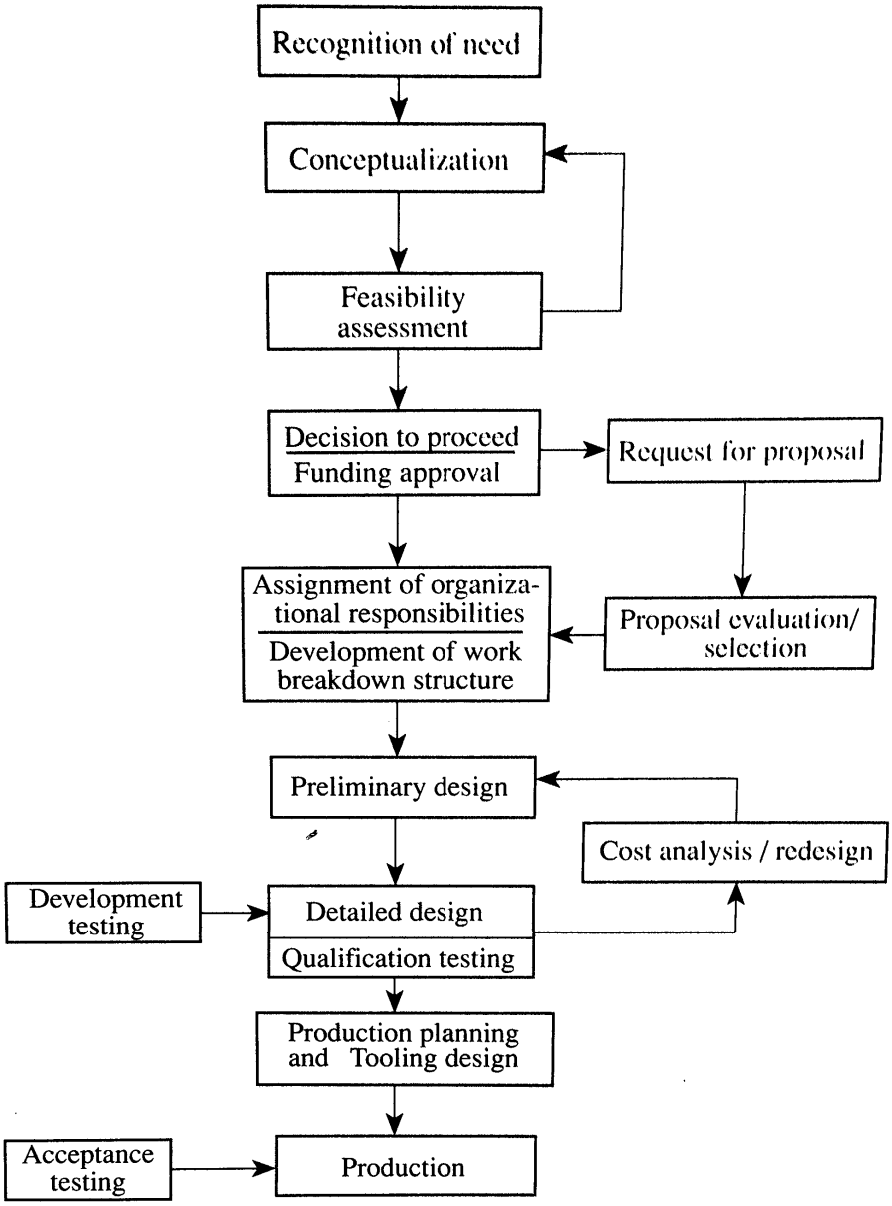
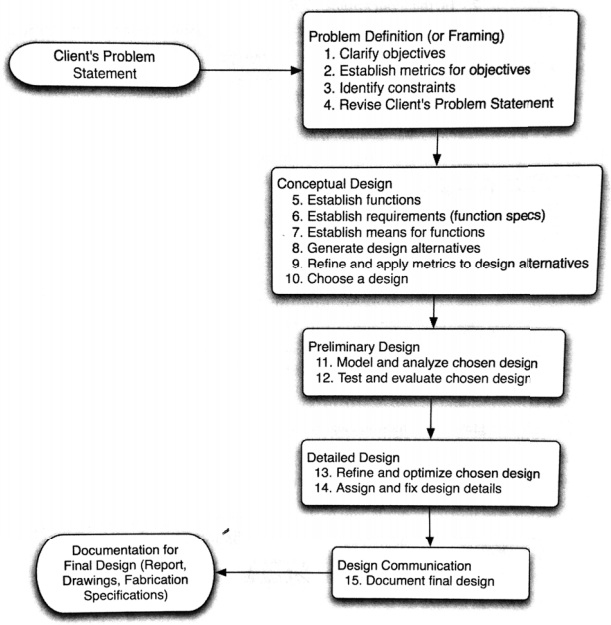
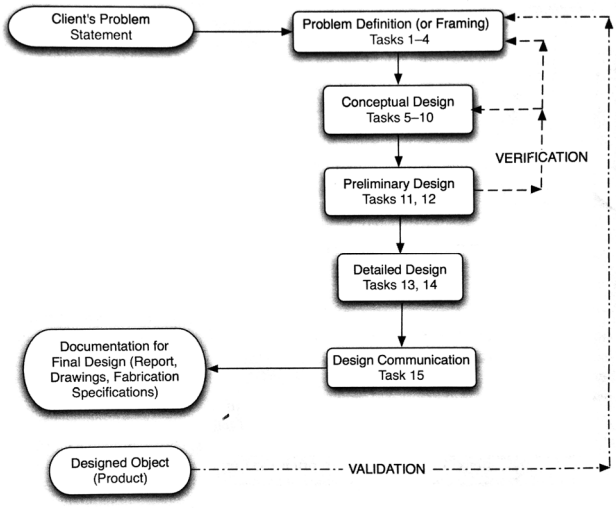


Figure 2.1: Engineering design process according to Ertas and Jones. Adapted from Figure 1.1 of [35]. Copyright 1996 © John Wiley & Sons, Inc. All rights reserved. Reproduced by permission of the publisher.



(a) Basic design process model. Adapted from Figure 2.2 of [29].



(b) Addition of some basic feedback to the model. Adapted from Figure 2.3 of [29].

Figure 2.2: Dym and Little's design process model. Adapted from Figures 2.2 and 2.3 of [29].

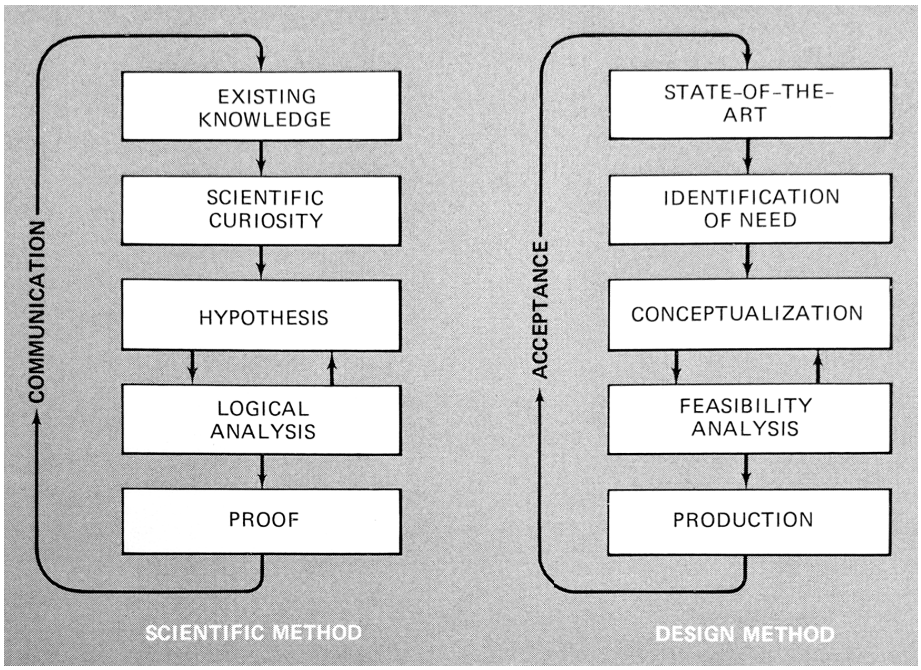


Figure 2.3: Comparison of the design method to the scientific method. Adapted from Figure 3.1 of [45]. © 1970 Percy H. Hill.

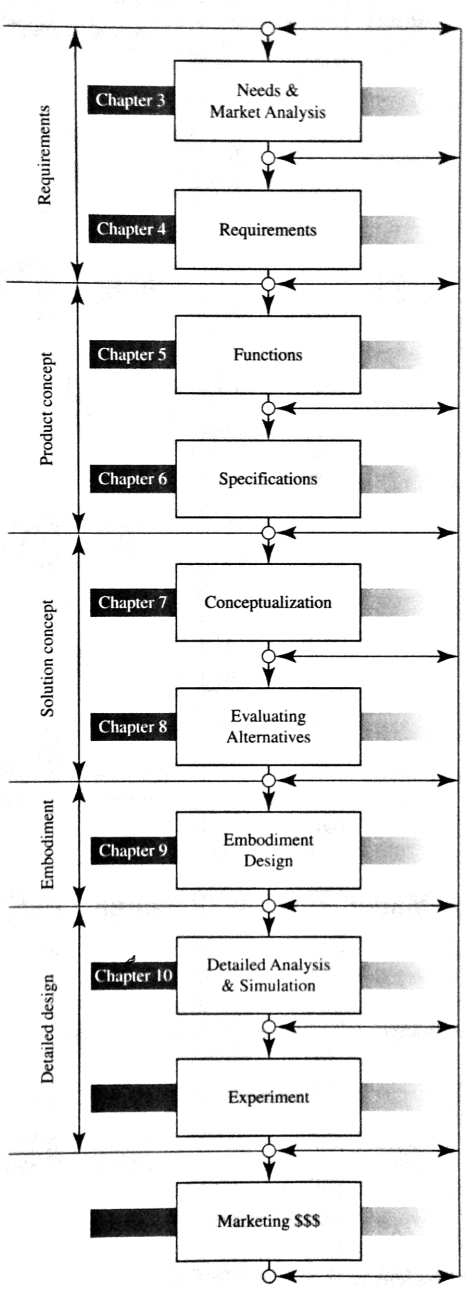


Figure 2.4: Design Process Steps identified by Haik and Shahin. Adapted from Figure 1.4 of [43]. From Haik/Shahin/Sivaloganathan. Engineering Design Process, 2E. © 2011 Cengage Learning, a part of Cengage Learning, Inc. Reproduced by permission. www.cengage.com/permissions

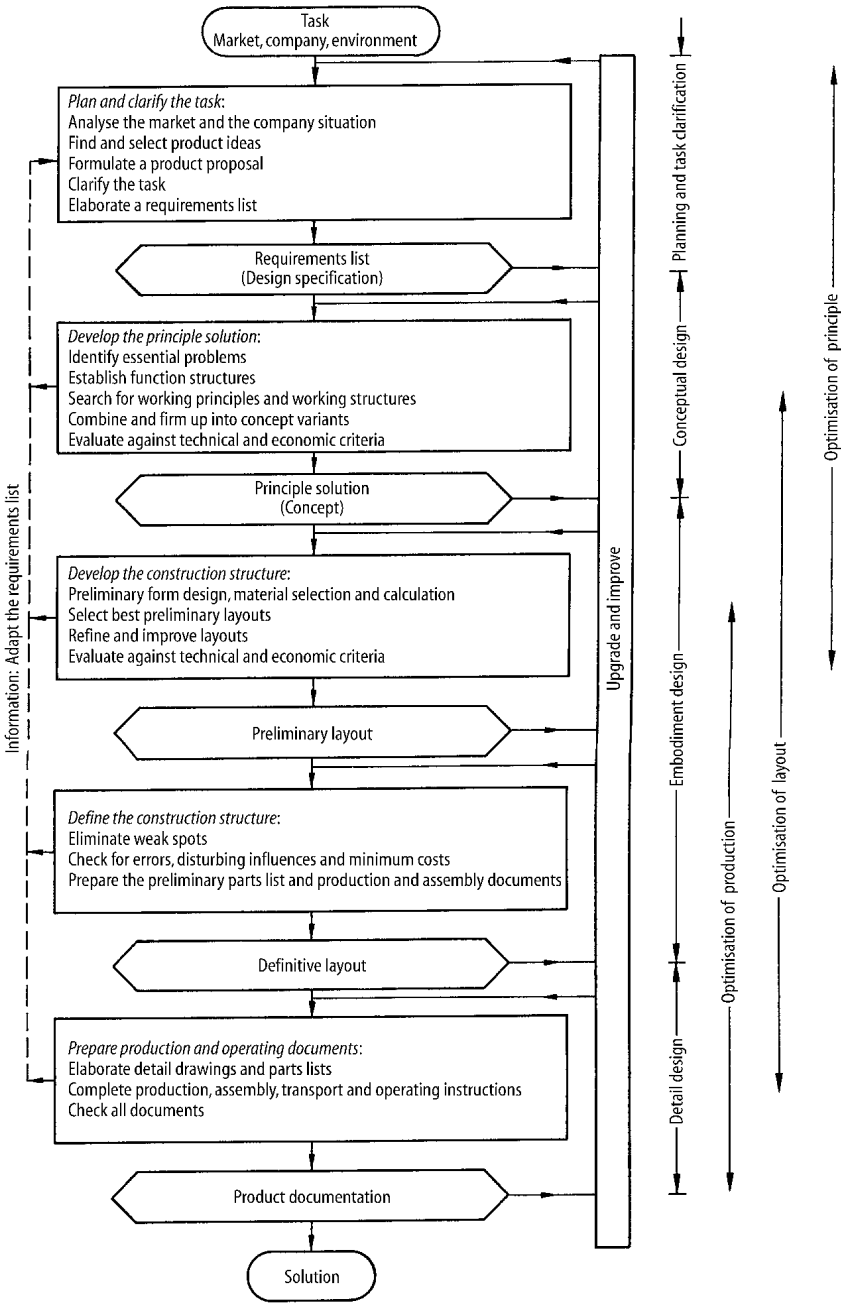


Figure 2.5: Steps in the planning and design process. Adapted from Figure 4.3 of Pahl and Beitz [67]. © Springer-Verlag London Limited 2007. With permission of Springer.

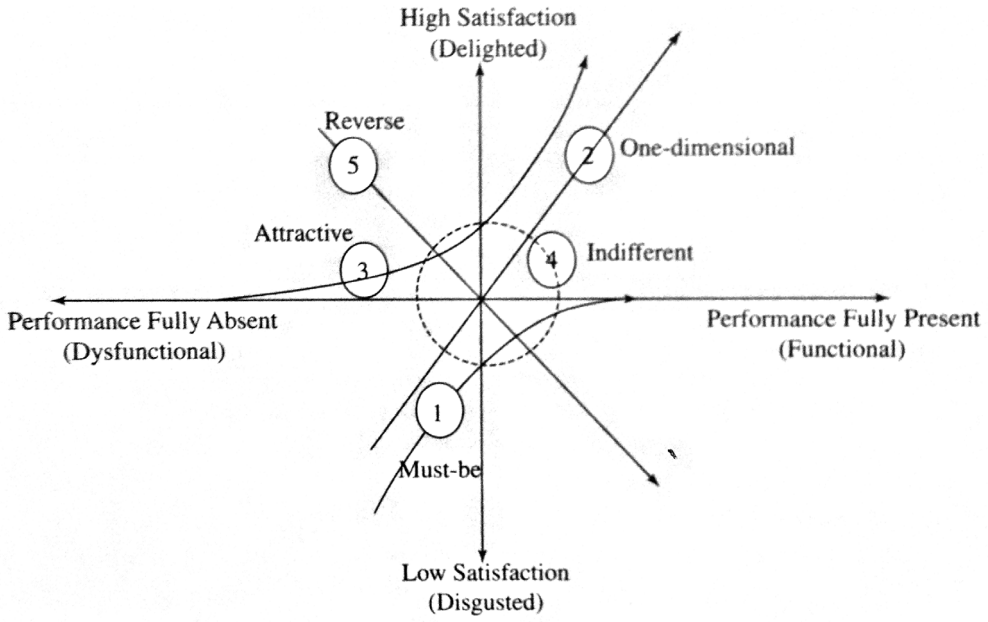


Figure 2.6: Kano Model for customer needs assessment. Adapted from Figure L7.1 of [43]. From Haik/Shahin/Sivaloganathan. Engineering Design Process, 2E. © 2011 Cengage Learning, a part of Cengage Learning, Inc. Reproduced by permission. www.cengage.com/permissions

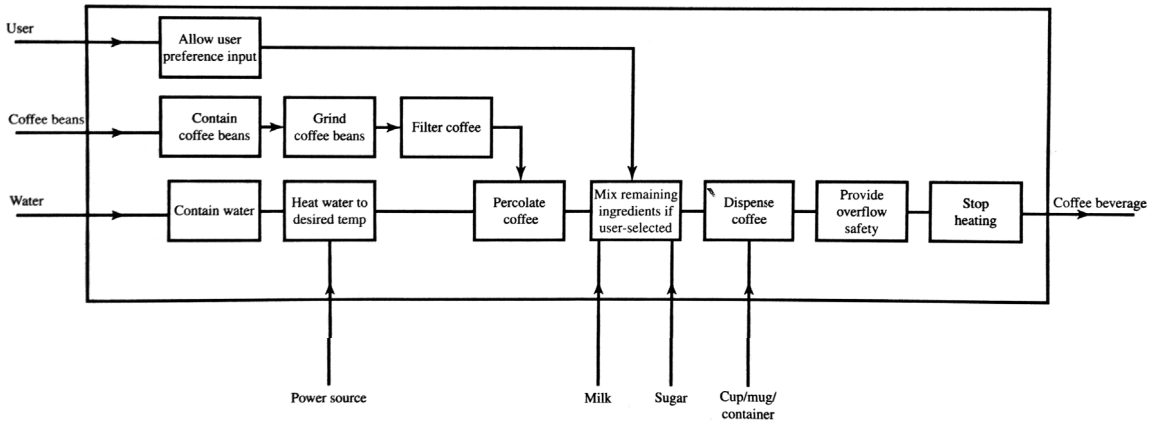


Figure 2.7: Function structure example for automatic coffee machine, combining a function or boundary diagram (external to the bounding box) and a function tree (internal to the bounding box). Adapted from Figure 5.4 of [43]. From Haik/Shahin/Sivaloganathan. Engineering Design Process, 2E. © 2011 Cengage Learning, a part of Cengage Learning, Inc. Reproduced by permission. www.cengage.com/permissions

Main headings	Examples
Geometry	Size, height, breadth, length, diameter, space requirement, number, arrangement, connection, extension
Kinematics	Type of motion, direction of motion, velocity, acceleration
Forces	Direction of force, magnitude of force, frequency, weight, load, deformation, stiffness, elasticity, inertia forces, resonance
Energy	Output, efficiency, loss, friction, ventilation, state, pressure, temperature, heating, cooling, supply, storage, capacity, conversion.
Material	Flow and transport of materials. Physical and chemical properties of the initial and final product, auxiliary materials, prescribed materials (food regulations etc)
Signals	Inputs and outputs, form, display, control equipment.
Safety	Direct safety systems, operational and environmental safety.
Ergonomics	Man-machine relationship, type of operation, operating height, clarity of layout, sitting comfort, lighting, shape compatibility.
Production	Factory limitations, maximum possible dimensions, preferred production methods, means of production, achievable quality and tolerances, wastage.
Quality control	Possibilities of testing and measuring, application of special regulations and standards.
Assembly	Special regulations, installation, siting, foundations.
Transport	Limitations due to lifting gear, clearance, means of transport (height and weight), nature and conditions of despatch.
Operation	Quietness, wear, special uses, marketing area, destination (for example, sulphurous atmosphere, tropical conditions).
Maintenance	Servicing intervals (if any), inspection, exchange and repair, painting, cleaning.
Recycling	Reuse, reprocessing, waste disposal, storage
Costs	Maximum permissible manufacturing costs, cost of tooling, investment and depreciation.
Schedules	End date of development, project planning and control, delivery date

Figure 2.8: Checklist for setting up a requirements list. Adapted from Figure 5.3 of Pahl and Beitz [67]. © Springer-Verlag London Limited 2007. With permission of Springer.

2.2 Domain-Focused Design Project Methods

2.2.1 Embedded Systems: A Contemporary Design Tool

Peckol [69] approaches the design process and project management from the viewpoint of embedded systems, with a focus on software (or firmware) and electronics hardware. Similar to other authors, Peckol outlines 5 general steps in a system design process: requirements definition, system specification, functional design, architectural design and prototyping. These steps clearly target the needs of embedded systems, and depart somewhat from general product development models. The design process is illustrated in Figure 2.9, which shows integrated system level development at the outset and near the end, and the separate but parallel development of both software and hardware during the middle of the process. The author indicates that there are two approaches to this multi-discipline process model. Traditionally, design is done separately by discipline and integrated later in the process. A more contemporary method looks to simultaneously and iteratively specify and design both hardware and software elements to meet system requirements. This makes it easier to manage and optimize trade-offs between disciplines, but requires careful architectural design and partitioning with clear specifications. It is noted that systems of greater complexity are more likely to benefit from the application of tools and formality in the design process. Peckol's model reflects a top-down approach whereby the design is iterated in ever increasing detail, though it remains generally linear even in light of parallel work between disciplines.

The author describes several other process models besides their alternative. The V-cycle model extends the waterfall approach with an increased emphasis on test and verification, as pictured in Figure 2.10. Though the implemented system is verified against the proposed design, there remains limited mechanism for the feedback enabling design changes efficiently, and does not provide insight into life cycle phases beyond initial product release (like post-delivery support). The spiral model, first proposed by Boehm [13], provides one of the first major departures from a linear process model. Boehm's original version is shown in Figure 2.11, which highlights the cyclical, iterative nature of the model. It allows for multiple builds or prototypes, and increased customer involvement,

while also explicitly including risk assessment in the process. However, the full spiral model is complex and elaborate, making management and efficient resource use challenging. Finally, Peckol explains the rapid prototyping method, which provides incremental prototypes in a similar manner to the V-cycle method, with additional functionality at each stage. This approach allows for rapid feedback from customers and improves problem detection and architecture validation. The author cautions against letting the prototype become the final product, but notes prototypes may either be disposable between stages, or evolutionary, by building on the existing system. Care must be taken to ensure the cost of continued prototyping is reasonable.

Peckol's design process is architecture-centric. The first steps involve requirements gathering and the development of specifications; these two activities are separated. Requirements are what the design should do, and are customer focused: they should use language appropriate for the customer or user, and consider the system from the outside only (inputs, outputs, behaviour, and environment). Specifications, link the requirements to the designers, and should use more formal engineering language with specific, quantifiable values. The focus is shifted further to how the system will accomplish the requirements, with a look at the system itself and its externally visible characteristics—the public interface. Peckol implies the specifications should be as formal as possible, going as far as suggesting they should be executable. By formalizing the capture of specifications for use with a modeling tool suite, the specifications may be used directly in the generation of code and potentially also be linked to the test and validation processes [41]. The author moves to functional design after the specifications and requirements are finalized. Using system partitioning and decomposition to create an architecture with loosely coupled, cohesive modules will improve reuse and parallel development, help stabilize interfaces early in the design, improve safety and robustness and reduce failure propagation, and break the system into smaller, more manageable problems. Functional design looks to specify and break down functionality in an iterative fashion to a sufficiently low level. Subfunctions should have behaviour specified, with messages and signals between functions and the external environment specified. Suggested tools for achieving this include a hierarchical decomposition, a functional partition, and boundary diagrams (interface between system and environment). Next, the functional modules are mapped to hardware

and software in the system architecture. Ideally, full specifications for physical components and interfaces for hardware modules may be finalized, and software modules' implementation defined using tools like block diagrams or UML diagrams. Tradeoffs associated with assigning functions to software or hardware must be evaluated. Connections between functional components must be included (hardware interfaces, software drivers). Architecture may be documented with the help of data and control flow diagrams or hardware architecture diagrams. Finally, prototyping stages involve detailed design, debugging, validation and testing. Appropriate level of documentation will support design review or static analysis, as well as testing or dynamic analysis, both of which are required to verify the design against original requirements.

Throughout the design process, Peckol gives several examples of documentation approaches and tools to accomplish the stage, but does not explicitly specify many. Instead, the focus is on using a level of detail and formality in both the approach or tools used for design as well as the documentation that is suitable for the complexity of the project. When determining the level of ceremony, one should consider intellectual property (IP) management and reusability, as well as forward and reverse traceability - being able to map between requirements and design data - which is useful for review and handling changes effectively. When a project nears completion, its design should be archived for future use and appropriate distribution. It is important to include all elements and tools of the project to allow it to be completely recreated from scratch (consider the applications and environments required to edit source files and generate artifacts).

Peckol highlights the importance of considering safety, reliability and risk throughout the design process. No real system is free of risk, and it must be managed in a manner appropriate to the application or product. The author defines reliability as "the probability that a failure is detected by the user is less than a specified threshold". A number of guidelines are given for achieving safe, robust and reliable designs. Understanding and managing requirements and specifications is important as well as reviewing throughout the process. FMEAs are suggested as a tool to help manage risk during the design process. It is suggested the system be tested beyond normal operating conditions to ensure faults are handled appropriately. During development, test and after release,

any errors or bugs that are observed should be recorded and reviewed. Finally, design features such as runtime exception handling and built-in self tests may improve fault detection and possible recovery.

2.2.2 Making Embedded Systems

White [98] focuses on specific techniques and practices to aid in the development of software/-firmware in embedded systems. There is limited discussion on managing a design project or considering the design from multiple disciplines due to the viewpoint taken. The suggested design process for embedded firmware is similar to a subset of that recommended by Peckol. A top down design approach is recommended, starting with functional decomposition and maintaining a solution neutral design as long as practical. Several tools may help with capturing the design. An architecture block diagram is suggested to capture the high level hardware and system design. A hierarchy of control organization chart and software layering view both provide insight on the software design. The former is similar, and helps identify clear software modules, outline control and priority for threads or other constructs, and show any potential resource contention. The latter further refines the complexity and partitioning of different modules and assists in identifying larger architecture layers such as portable software libraries or hardware abstraction layers (HALs). Finally, White explains a variety of techniques to help make a design successful. One should use existing design patterns where applicable (for example, consistent APIs), and incorporate logging and debug features early to help with development, testing, and later, runtime fault diagnosis. Making software modules more portable may allow testing on different platforms or virtual environments to speed development. Versioning the code effectively and using version control systems will also aid the development, testing and support of the project. Overall, White's approach is fairly minimal with limited ceremony, but does encourage clear upfront design of the the system architecture.

2.2.3 Embedded Systems Architecture

Noergaard [60] focuses on the design of embedded systems, similar to Peckol, and specifically focuses on the tasks and methods associated with developing the system architecture. An architecture is an abstraction of the system that considers behavioural and relational information of individual and between multiple hardware or software elements in a system. The author suggests a number of rules for teams carrying out embedded systems design. A systems engineering approach should be used, whereby both technical and non-technical influences on the design and development process are understood. Teams must be disciplined in following development processes and best practices, and also review and improve these processes and practices regularly. These processes and practices are often based on existing schemes like waterfall, spiral, XP or Scrum (explained previously or later). Noergaard presents an example embedded systems design and development life cycle model, illustrated in Figure 2.12. This process model differs from those in other sources in that it clearly shows several key areas of iteration. First, product concept, requirements and architecture design may be iterated upon. After creation of the architecture design, iteration with review and feedback generates different versions of the architecture. After the final version of the architecture, the system is developed, reviewed and tested in an iterative fashion. In general, the method looks to lock in requirements, then use this to create a final version of the architecture, after which a tested system is delivered.

The author suggests teamwork is critical in a successful design project, and encourages smaller teams with open and effective communication. The processes and practices should be developed with the team and be appropriate for the team, the project, and the requirements. Noergaard recommends the following methods for success, which are directly quoted here:

- "Ship a very high quality product on-time."
- "Have a strong technical foundation."
- "Sacrifice fewer essential features in the first release."
- "Start with a skeleton, then hang code off the skeleton."

- "Do not overcomplicate the design!"
- "Systems integration, testing and verification from day 1."

The author explains that to develop a complete set of requirements, a wide range of stakeholders and influences must be considered. Incorporating all influences into the process early will help reduce changes and associated costs later, after the architecture has been developed. It is suggested requirements should be clearly documented, and this will help ensure understanding between stakeholders. Prototypes are helpful for modeling the design and behaviour, and also determining which hardware and software solutions may be feasible early in the process. Using requirements, an architecture may be developed starting from functional requirements. A reference model is useful in describing the topological layout of components. Decision matrices may aid in evaluating whether hardware or software elements should be purchased or built in-house using cost, time-to-market, and performance as metrics. The architecture should be documented. Documentation should cover all elements of the system and their inter-relationships. The methods used to document will vary and should match the team, though documentation should be usable across both non-technical and technical audiences. Finally, the architecture should be evaluated against requirements (or scenarios), for potential risks and for possible failures. The evaluation should be documented, and include any design change recommendations.

2.2.4 Mechatronics: A Foundation Course

De Silva [25] explains what composes a mechatronic system, and outlines concepts and technologies associated with them, as illustrated in Figure 2.13. They further compare a mechatronic system directly to a control system, containing a plant, controllers, actuators and sensors, and other components. The author focuses on technical content, and does not explain the design process or project management in detail, though several core strategies are highlighted and a general design process is outlined. De Silva encourages the use of modeling and simulation to drive analysis and design. Ideally, this process is iterative, continually refining both the design and the model. Models

and simulation can provide an alternative to physical prototypes, which can be very expensive. As mechatronics systems incorporate multiple domains, a successful system benefits from the concepts of integrated and concurrent design across all domains. The interactions between domains can use "coupled design", supported by modeling the whole system to provide insight into the dynamic behaviour within and between domains. Typically, the "unifying thread" for a multidomain model is the energy and power flowing between domains. The importance of integrated and concurrent design goals in mechatronics systems should be reflected in the team and project organization. A multidisciplinary team working together with effective communication among the team and with stakeholders is more in line with these goals than breaking the project down and assigning different elements to separate, domain-specific teams. Part of effective communication is record keeping - making a communications and decisions history available in the future - as well as documentation. The medium or format of different communication should be considered carefully to be effective. Schematics and block diagrams are used heavily to explore and document design. Like Haik and Shahin [43], De Silva introduces the Quality Function Deployment (QFD) method as a formal way to communicate and include customer needs and requirements into the engineering design process.

The author briefly describes the engineering design process as a list of steps that are quoted here:

- "Identify need for the required design"
- "Describe the objectives of the designed product or system"
- "Gather preliminary performance specifications (some may be qualitative)"
- "Outline several concept designs"
- "Identify the technologies and expertise that are needed"
- "Identify available (COTS) components or subsystems for each conceptual design"
- "Establish quantitative and final performance specifications"

- "Obtain preliminary detailed designs"
- "Evaluate the designs through modeling, analysis (including economic), and the use of experts and narrow down the design alternatives (to one or two)"
- "Carry out a detailed design"
- "Evaluate the detailed design through analysis, prototype development, and testing, etc."

These steps imply a very linear, waterfall-like process. The author does note that several iterations may be required for some or all of these steps. In addition, requirements and specifications are explicitly evaluated twice to reflect additional information obtained early in the project.

2.2.5 Introduction to Mechatronic Design

Carryer et al. [19] says that creating a good mechatronics design demands cross disciplinary knowledge, especially when making architecture partitioning tradeoffs. Further, the key is to effectively use devices by understanding their behaviour, rather than having a deep understanding of the underlying physics. The design process model presented tries to stress that iteration occurs throughout all steps in the process, even if the base process is linear in nature. This is highlighted by showing feedback connections between each and every step between Define Requirements and Validate, with some additional feedback from Maintain. This model can be summarized as the following steps:

- Project Initiation
- Define Requirements
- Generate Designs
- Evaluate Concepts
- Define Specifications
- Develop Project Plan
- Execute Plan
    - Design

    - Build
    - Verify
    - Validate
- Deliver
- Maintain
- Project Completion

The model includes some of the steps that may occur after a design is finalized and tested. Interestingly, it also puts emphasis on the importance of the requirements, concepts and specifications early in the process, the results of which can be used to develop a plan for the actual project. This reflects the difficulty in planning without first fully exploring and understanding the project (i.e. without complete requirements and specifications). Requirements should be solution independent, and thus are separated from specifications, which are prepared after the concept designs are developed and evaluated and more solution specific. Concept development may be aided by decomposing the system and generating and documenting as many possible solutions as possible, perhaps with the aid of techniques like brainstorming and morphological charts. The authors recommend using an iterative prototyping approach to evaluating the merit of concept designs, which is explained in more detail below. As compared to requirements, which describe the overall goals, specifications look to define quantifiable metrics of performance to describe exactly how the system must perform to meet requirements. Further, specifications are very important in defining interfaces between subsystems when a system is decomposed.

Carryer et al. briefly mentions that the most important tasks in design project management are to: define and relate tasks and subtasks, budget resources, create and maintain a schedule, and control the processes. Gantt charts and network diagrams are suggested as possible tools to aid in this. It is important to actively manage and update the project schedule and resource usage. This is in addition to reviewing and managing the goals of the project (scope and approach) regularly. All together, monitoring the process and incorporating feedback will lead to the project meeting base requirements on schedule and within budget. To complement this, the project team must have the

discipline to maintain a level of documentation appropriate for communicating the design and the process. The authors suggest at minimum a team keep an up to date logbook that is accessible and easily updated, containing information quoted as following:

- "Up to date schedule"
- "System requirements"
- "System and subsystem specifications"
- "Concepts generated and selected"
- "Sketches, diagrams, and descriptions"
- "Schematics (current and previous versions)"
- "Code listings"
- "Guides to the locations of files and programs"
- "Notes from meetings and decisions that are made"
- "Current status"

This logbook is targeted at small design teams completing senior design projects during undergraduate engineering education. Finally, systems engineering and concurrent design practices should be incorporated. Systems engineering looks to break a project down into subsystems in a top down manner to divide between appropriate personnel and resources (i.e. multiple teams). This is especially useful with complex projects where some subsystems require the work of specialists. Note that careful change management and communication must be implemented to ensure all teams stay in sync. Concurrent design demands that all parts of a lifecycle and all stakeholders be considered simultaneously during design, with the goal that this will reduce costly changes or side effects late in a project. Activities to consider include manufacturing, assembly, test, service, shipping and distribute, and future upgrades.

Creating a prototype early will help a designer evaluate the feasibility of a concept. Prototyping

should be done in an iterative fashion, starting with a simple (but sufficient) prototype and repeatedly making refinements such that the concept may be proven to a sufficient level of detail or to investigate problems that may arise. If a concept does not meet the metrics defined to consider an approach acceptable, it should be discarded as early as possible to avoid taking any more time or resources. Even if a prototype is discarded, it will provide insight into design weaknesses and help in estimating the effort for the final system design. The authors suggest two types of prototype: building a physical prototype, or simulating with a model. Ideally, both will be used. Simulation may require up front investment to learn the tools and develop a model, but can allow for rapid iteration and feedback with low cost. Physical prototypes may range from very cheap and fast, to expensive and time consuming, depending on the level of fidelity, techniques used, and the available resources and timeline. The suggested steps for both mechanical and electrical domains are to:

- Create a schematic or solid model.
- Determine what parts or phenomena in a system may be effectively modeled and simulated and do so.
- Build and test a prototype.
- Iterate by revising the models and simulation and updating the prototype.

When creating a prototype, a number of techniques are suggested. Prototyping and testing only some subsystems may sufficiently prove the design. If possible, testing subsystems independently prior to an integrated test will help make the full system successful. When designing and building a prototype (or model/simulation), consider including features that allow for observability, as well as easy adjustments, accessibility, and assembly/disassembly.

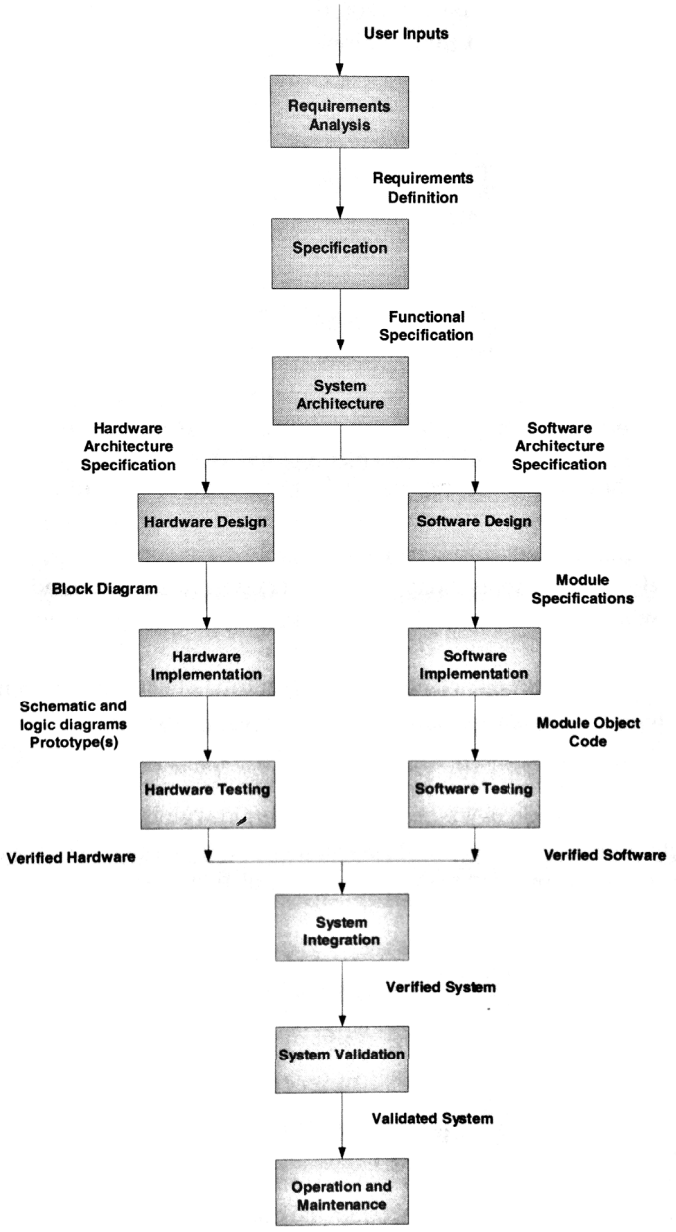


Figure 2.9: Peckol's embedded systems design process model. Adapted from Figure 0.5 of [69]. Copyright 2008 © John Wiley & Sons, Inc. All rights reserved. Reproduced by permission of the publisher.

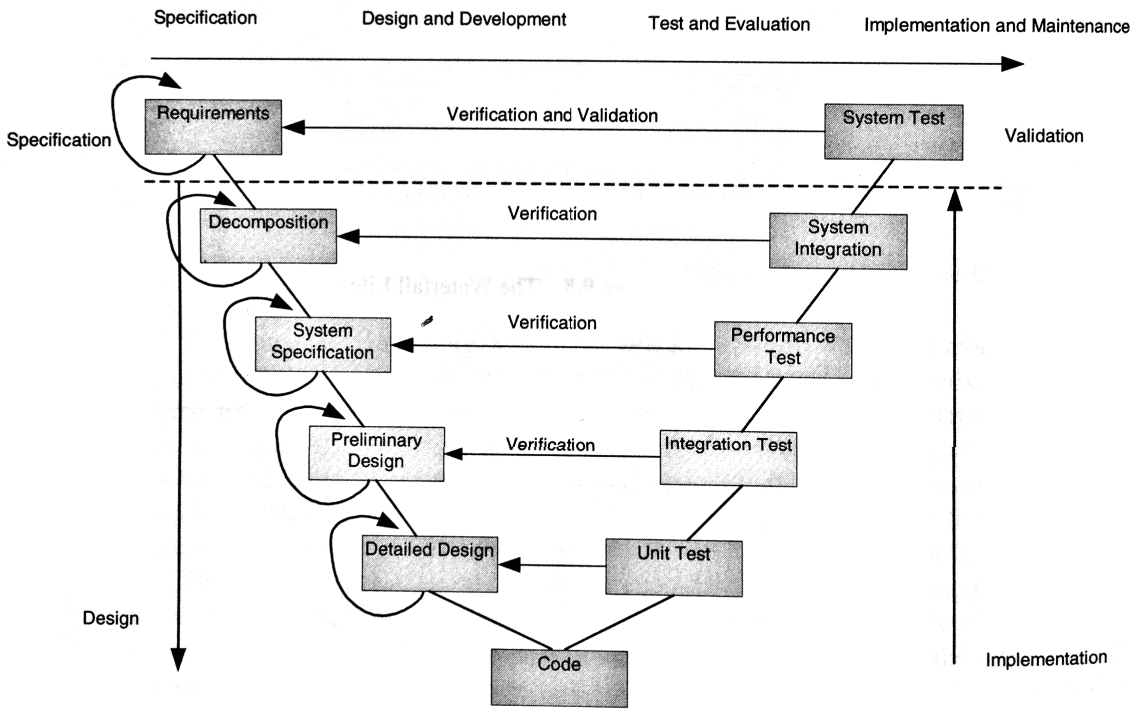


Figure 2.10: Traditional V-cycle design process model. Adapted from Figure 9.9 of [69]. Copyright 2008 © John Wiley & Sons, Inc. All rights reserved. Reproduced by permission of the publisher.

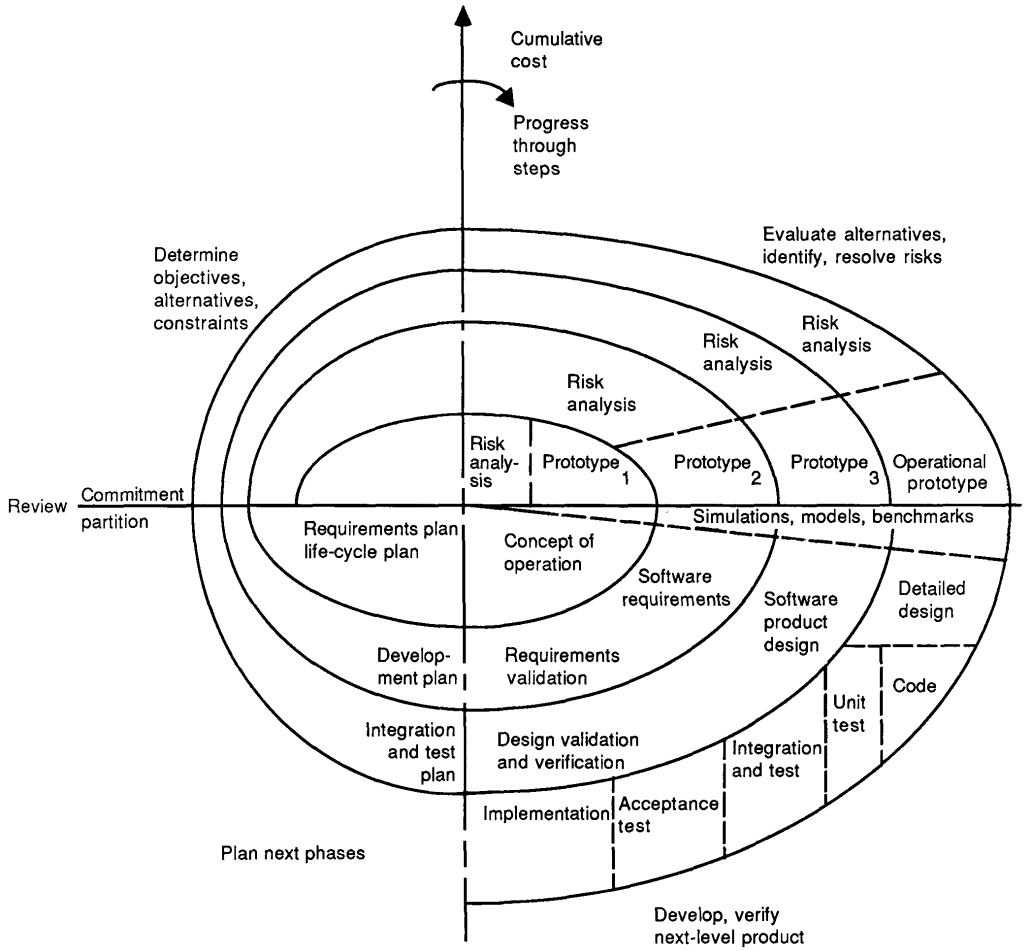


Figure 2.11: Spiral design process model as proposed by Boehm. Adapted from Figure 2 of [13]. © 1988 IEEE. Reproduced by permission of the publisher.

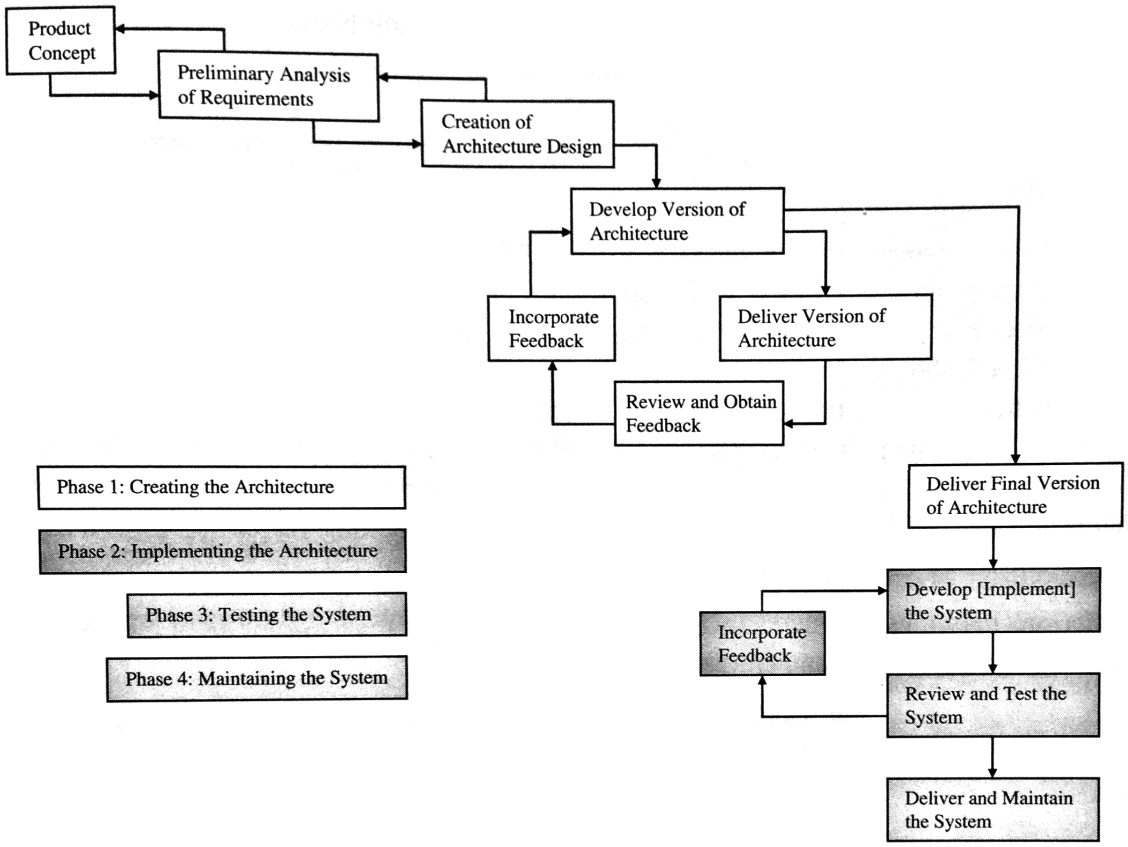


Figure 2.12: Noergaard's embedded systems design process model. Adapted from Figure 1.2 of [60]. Embedded Systems Architecture : A Comprehensive Guide for Engineers and Programmers by Noergaard, Tammy. Reproduced with permission of Elsevier Science in the format Thesis/Dissertation via Copyright Clearance Center.

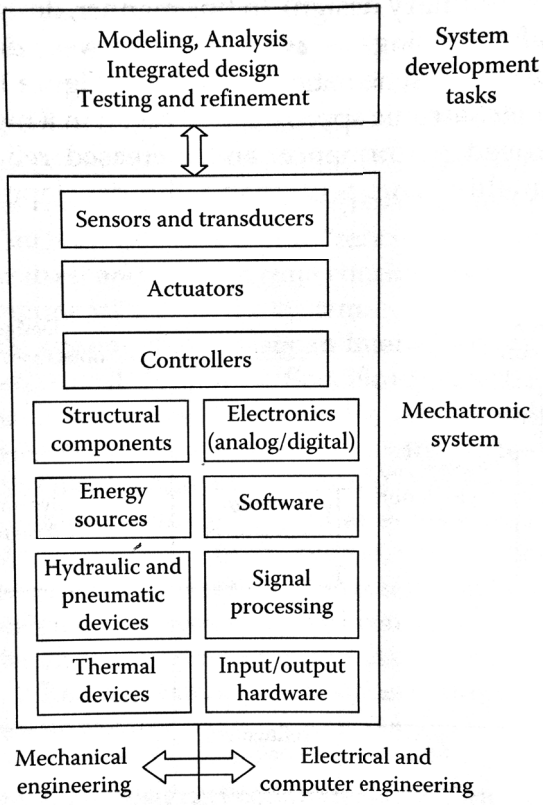


Figure 2.13: Components of a mechatronic system. Adapted from Figure 1.4 of [25]. Copyright 2010 From Mechatronics : a foundation course by Clarence W. de Silva. Reproduced by permission of Taylor and Francis Group, LLC, a division of Informa plc.

2.3 Agile Design Project Philosophy and Methods

Agile development methods have become prevalent in software projects within the past two decades [37]. There are a wide range of methodologies that leverage agile principles as an alternative to traditionally formal and plan-driven approaches. The vast number of different methodologies target different domains, and have unique focuses across developement techniques and project management support. A few examples of the many agile-like methods include Scrum, Kanban, Scrumban, XP, Feature Driven Development, Dynamic Systems Development Method, various Lean methods, Crystal Methods, Adaptive Software Development, various Unified Processes, Scaled Agile Framework, Large-scale Scrum, Disciplined Agile Delivery, and Extreme Project Management. Typically, agile methods have been targeted at projects where outcome is less critical and requirements are less certain. However, there are starting to be more examples of adapting agile approaches for use in critical projects demanding safety and quality [37, 58]. A discussion of agile principles and a brief description of some agile methodologies follows.

2.3.1 The Agile Manifesto

The agile software development movement was solidified in 2001 with the release of the Agile Manifesto [24], a set of four key values and twelve associated principles. It was created and supported by key agile software development proponents of the time, including Jim Highsmith, author of "Agile Software Development Ecosystems" [44], and Ken Schwaber, Jeff Sutherland, and Mike Beedle, the main developers of the Scrum agile methodology. The four values are quoted here:

- "Individuals and interactions over processes and tools"
- "Working software over comprehensive documentation"
- "Customer collaboration over contract negotiation"
- "Responding to change over following a plan"

The manifesto states that the items on the left are deemed more important than items on the right,

even if the latter still hold value in a project. These values are reflected in the twelve core principles, quoted here:

- "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."
- "Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."
- "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."
- "Business people and developers must work together daily throughout the project."
- "Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done."
- "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."
- "Working software is the primary measure of progress."
- "Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."
- "Continuous attention to technical excellence and good design enhances agility."
- "Simplicity—the art of maximizing the amount of work not done—is essential."
- "The best architectures, requirements, and designs emerge from self-organizing teams."
- "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

The values and principles influence the development of agile methodologies.

2.3.2 Agile Software Development Ecosystems

Highsmith [44] first outlines the concepts and philosophies of agility as applied to software development projects. This is followed by introducing the ideas and influences of a number of other Agile Manifesto signatories. A brief review of Agile software development methodologies that were popular at the time of writing leads to a description of how one may approach creating or adopting an agile software development ecosystem that is appropriate for a project and organization. The author uses the term ecosystem to refer to the collection of organizational culture, people, various methodologies and other elements that must work together rather than consider only a single methodology to guide all design projects. It is noted that while the focus of the work is an application to software, the underlying agile principles are readily applied to other aspects of business development and project management.

Characteristics of Agile Approaches

The author describes agility as "the ability to both create and respond to change in order to profit in a turbulent business environment." The agile software development movement started as alternative approach in software development in the 1990's, spurred by apparent failure of rigorous methods of software development in large IT software projects. Highsmith claims agile approaches are characterized by a different focus than traditional project management and software development in a number of different areas or dimensions.

First, there are two clear cultures or project types, those of optimization and those of exploration. Optimization is focused on prediction with plans and schedules and then controlling or reducing deviation from these, while exploration uses plans and schedules as a guide to manage uncertainty and make tradeoffs in cost, schedule and scope. Agilists accept that plans are "virtually always wrong", and look at plans as guides for decision making and collaboration instead of a control mechanism. The culture aspect indicates the tendencies of individuals and the organization as a whole, while projects may also be categorized in a similar manner, based on the level of uncertainty

or knowledge and understanding in all parts of the project. An agile approach is most often suited to exploratory projects or cultures, or projects with high levels of uncertainty.

Traditional and agile methods must both balance structure and flexibility. Traditional methods tends to heavily favour structure, while agile methods favour flexibility, relying on the ability to respond to changes more than the ability to plan accurately. Part of this is balancing the disciplined adherence to rules with the understanding of when to improvise or modify to better suit the situation.

The agile mindset embraces change and looks to adapt quickly to varying business conditions. This demands rapid decision making, even if the decision is not the best or most optimal, and trying to meet goals with the minimum effort. The feedback obtained from these shorter cycles will help guide future decisions and activities. Highsmith emphasizes that "agile projects are not controlled by conformance to plan but by conformance to business value." Traditional project management takes high level goals and constraints to create a plan, with a specific and detailed schedule, budget and scope. Management activities try to monitor and control the process to conform to the plan, and success is evaluated by looking at deviation from the plan, which may be unrealistic or outdated. Agile project management tries to view the project more holistically, and evaluate success of a project by the business value it delivers. The focus is on the high level goals and constraints. While this may mean meeting a specific deadline in some circumstances, the result is a more flexible framework to achieve success and deliver value to customers and stakeholders.

Traditional project management uses a predictive approach and emphasizes greater and more detailed upfront planning, while agile methods use an adaptive approach. A comparison may be drawn to control systems. In a well understood physical system, a detailed and accurate model may be created. This model may be used as part of an effective control system leveraging feedforward control and minimal monitoring and feedback to achieve stability and tracking. This predictive approach relies on a level of stability and minimal uncertainty and noise in the model, inputs, outputs and measurements. When controlling a system with high uncertainty and noise and a poor model, rapid feedback using a variety of measurements is important. Further, the control strategy

may need to be adjusted or adapted across a wide operating range using additional insight into the system behaviour that may be collected.

Highsmith explains that traditional approaches tend to emphasize prescriptive processes - describing in detail the method of carrying out tasks that contribute to project success. Agile looks to move one step back and focus on the team and individuals that make up the team, and how they work to make the project successful. This is similar to the view on value versus conforming to plan, where the goal of various processes and policies prescribed by traditional management is to help the team or organization be successful. Agile methodologies simply acknowledge this and try to put it at the forefront. Traditional methods view people as interchangeable, parts of the overall process in a standardized form. Agile methods try to leverage the variance in and unique capabilities of people on a team to be more effective. Team members have skills, which are readily transferable across projects and tasks. Compare this to the policies or rules of a project or organization. A set of policies for communicating and coordinating amongst a team or organization may be thought of as the methodology. When dealing with unique, exploratory projects, each is different, and requires an appropriate set of policies matched to the needs of the project. A single methodology will not be appropriate for all projects or all teams. Further, as a project evolves, the methodology should evolve with it to remain relevant and practical.

A good manager should trust that team members with appropriate skills will work towards the success of the project, with the key being good teamwork. The team should be supported by policy and process, but not governed by it. The priority is to use these tools to enhance and improve the performance of the team and the value of the project. If the team is inexperienced, no amount of process will replace a lack of skill among team members. Rather it is important to ensure the team has the skillset required or to work towards the learning and knowledge transfer necessary to build the appropriate skills.

Achieving high performance within a team requires communicating and coordinating between team members, and other stakeholders. Collaboration, or active participation, further enhances the performance of the team when making decisions, transferring knowledge and learning, and creating

deliverables. While this is supported with policies and practices appropriate for the the team and the project, it should not be impeded by them. It is important to consider the characteristics of the interaction (frequency, time scale, etc.) and the nature of the information. Explicit information is readily communicated and archived in static mediums, while tacit information relies more on interactive and collaborative approaches to communication. Practices should suit the type of information being communicated, with the goal of reducing the burden or cost of communication to make it more effective. At the lowest level, where frequent communication between team members supports collaboration, the author asserts that face-to-face interaction is the most effective.

An agile methodology relies heavily on regular collaboration with customers or other stakeholders in addition to effective interaction within the team. An evolving project needs constant feedback from the customer to make decisions on what activities will deliver the greatest value while also ensuring any new information is communicated that will reduce uncertainty and risk in the project. A project with a client unwilling to actively collaborate is not well suited to an agile management approach.

Part of collaborating effectively with the client is communicating the progress of the project and the value delivered. Agile approaches consider a working product to be a better method of demonstrating progress and eliciting feedback than intermediate design artifacts or documentation. The value is in understanding, which documentation may help convey, but not in the actual generation of the documentation. Documentation is an artifact which may be useful in some methodologies, and an avenue with which to provide value of communicating information and keeping it intact for future use, but not the only or necessarily best method. It is limited only to explicit information, and takes time and effort, and often does not keep up with the reality of the current state of the project. While failing to document a design and the decisions made in a project altogether is not logical, nor is creating reams of documentation. A balance must be struck with effective practices and policies.

Contracts with clients remain important and necessary regardless of design project management methodology, though they must be tailored to fit the situation. Traditional project management

following a predictive, waterfall-style approach often use a fixed pricing scheme with a target schedule and scope. This does not reflect the philosophy and values of an agile approach. A contract for an agile project should incorporate the need for constant customer interaction and continuous demand-driven refinement of goals in the face of uncertainty. Highsmith suggests the use of "delivered-feature contracts", which are fixed-schedule, variable-scope contracts built around the periodic (iterative) delivery of features. Each iteration should include the customer in a collaborative plan to set delivery commitments using up to date information, the acceptability of previously delivered value, and the high level plan. The end of the iteration should include a feature or progress demonstration by which the customer can evaluate the value delivered. Overall targets for scope, schedule and cost may be included in the high level plan, but should not be a contractual obligation. This model is similar to that used by venture capitalists, which looks to manage and share risk among parties and encourage all parties to take actions in the best interest of the project.

Description of Agile Methodologies

Highsmith describes a number of agile methodologies prevalent at the time of writing. These include Scrum, Dynamic Systems Development Model, Crystal Methods, Feature-Driven Development, Lean Development, Extreme Programming, and Adaptive Software Development. Each of these methodologies are based on key agile principles. Iteration is key to applying feedback in complex and uncertain projects. Customer requirements will change throughout the project, making detailed long term plans tenuous. Constant communication within a development team as well as with management and customers is important to facilitate understanding and feedback. The focus should be on delivering the minimum that meets the customer requirements, and avoid anticipating and including extra features that the customer may never use. However, the design and the process should be such that it is easy to change to meet evolving customer priorities. While some up-front high-level planning is necessary, the design should regularly be reviewed and refined based on new information. All of these principles look to continuously reduce uncertainty and risk as the project progresses, which tend to be high in many software development projects.

Scrum Scrum is heavily focused on project management and has few software-specific features. It uses time-boxed iteration to monitor the progress of the project, adjust based on feedback from different project stakeholders, and provide an environment conducive to success facilitated by team interaction. This interaction is composed of communication, collaboration, coordination and knowledge sharing. An iteration in Scrum is called the sprint and may last one to eight weeks, with shorter periods being more common.[1] A sprint has a planning stage at the beginning, and a demonstration or delivery and review stage at the end. A product backlog describes all of the tasks or features that together make up a successful project. In Scrum terminology these are "user stories" or just "stories" to reflect that they may be written in a non-technical manner understandable by both the customer and the development team. It requires review and revision regularly, typically at the beginning of sprint planning. A sprint or release backlog is a subset of the product backlog that describes the priorities for the next sprint and release. The sprint planning meeting at the beginning of the iteration allows the product owner (who represents the customer's interests) and the development team to cooperatively plan what will be completed during this sprint based on task complexity and available and resources. This set of tasks is then locked until the next sprint planning meeting. During the sprint, the development team draws tasks from the sprint backlog and works together to complete all tasks in the backlog by the end of the sprint. Part of the planning phase involves identifying the criteria for a task to be considered complete, which usually means it will be demonstrable at the end of the sprint. During the sprint, there will be a daily scrum meeting (also known as a daily stand-up). All development team members should attend this short meeting (short enough one may stand for the whole thing), along with the product owner, management, and the "scrum master". Team members should outline what they have accomplished since the last scrum, what they intend to accomplish before the next scrum, and if there is anything impeding their progress. The goal of a scrum master is to work with the team to follow the process, identify impediments (typically logistical or organizational), and remove them. While the product owner and management may obtain useful status information from the scrum meeting, they should be actively involved in questioning or guiding the development team. At the end of a sprint, the

[1] Highsmith suggests 4 week sprints, but as Scrum has evolved since this source was written, sprints are more commonly 1-2 weeks based on anecdotal evidence.

development team demonstrates progress to the product owner and other stakeholders. This information is used to revise the product backlog, plan the next sprint, and make go/no-go decisions on whether the project continues. Additionally, the development team should have a retrospective meeting to reflect on the progress made and how to adjust their practices to be more successful or address shortcomings. The Scrum methodology is illustrated in Figure 2.14. Currently, Scrum is one of the most well known and used agile methods in software development.

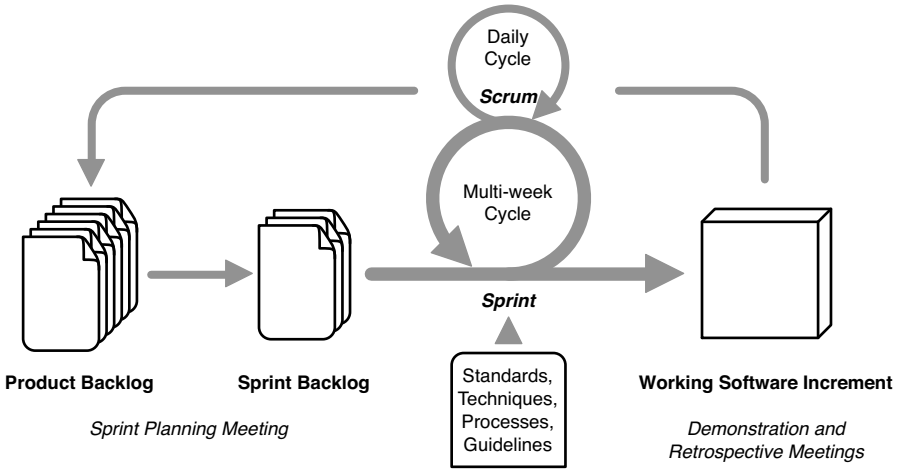


Figure 2.14: Scrum method of project management.

Dynamic Systems Development Method The Dynamic Systems Development Method (DSDM) was based on Rapid Application Development, and provides a framework for time-boxed, iterative development. DSDM has much more complexity and formality than most other agile methodologies, but may be more suited to transitioning from a more traditional culture. In addition, the formality lends itself well to quality certification like ISO 9001, and it is backed by an industry consortium [3]. DSDM has three main iterative phases, shown in Figure 2.15, and numerous principles, roles and work products. However, it is also designed to adapt to existing techniques of the adoptive organization. The first phase is the Functional Model phase, which looks to gather and prototype functional requirements, and specify non-functional requirements, in an iterative manner. The Design and Build phase looks to meet all functional and non-functional requirements by engineering

the software and refining the prototypes. The time-boxes of these phases may be based on features, whereby a small set of features goes through all of the phases during each time-box, and overlap between these first two phases is likely. Finally, the Implementation phase is about deploying the system to the user's environment. There is a significant focus on the use of prototypes as part of what is delivered in each stage. While the author cautions against letting a rough prototype become the final product, DSDM sees prototypes as a minimal building block that may either be used to demonstrate feasibility (business, usability, performance, capacity) or as a foundation for the system. Prototypes are used to capture information for collaboration as an alternative to heavy documentation. One underlying idea driving the focus on iteration and prototypes is the 80-20 rule, whereby "nothing is built perfectly the first time" and the last 20 percent of the work is very time consuming. Further, it is difficult for the team and the client to foresee all requirements at the beginning of the project, and this change is supported by iteration and the idea that "the current step need be completed only enough to move to the next step."

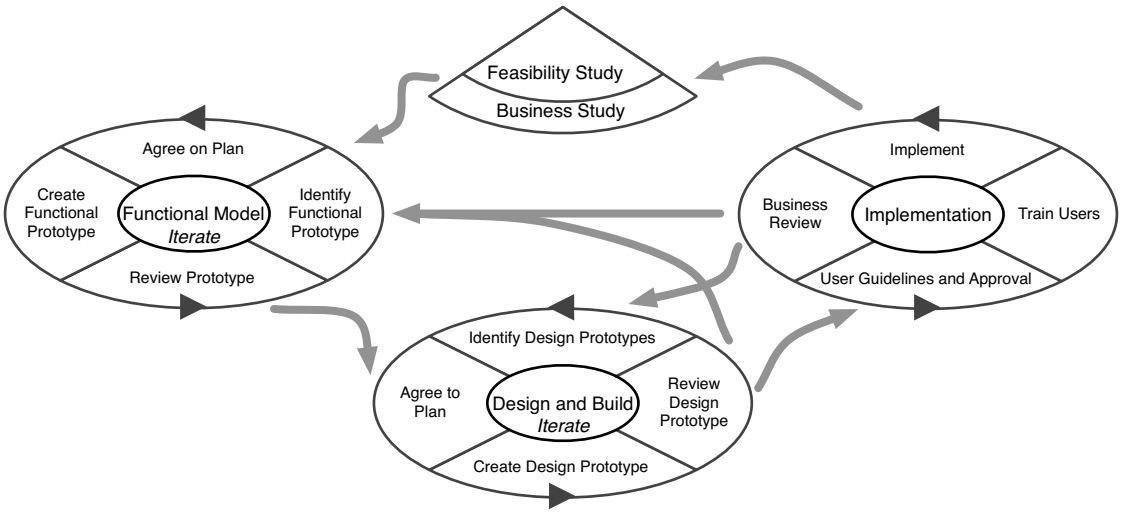


Figure 2.15: Dynamic Systems Development Method.

Crystal Methods The Crystal Methods, developed by Alistair Cockburn, create a framework of methodologies of varying weight designed to be adapted to a specific team, project and domain (Crystal Clear, Crystal Orange, etc.). The three main factors affecting methodology design are the communications load, the system criticality and the project priorities. Communications channels grow exponentially with larger teams, and this requires more practices and tools to supplement or replace face-to-face interaction. Projects where the severity of software failure is high (i.e. life threatening) will likely have external regulatory demands on methodology elements (such as more policies and less tolerance on practices). This may increase the weight of the methodology and add constraints, but should not significantly impact methodology design. For example, even if FDA approval is targeted, the detailed documentation and traceability requirements may supplement agile development later in the process. Note that methodology weight is defined as the product of size (i.e. number of items per element) and ceremony (e.g. formality and detail of documents). Finally, different project priorities impact methodology design (e.g. time to market, cost reduction, exploration, legal liability). Crystal methods provides a matrix to aid in selecting and tailoring an appropriate methodology with each of these three main factors on an axis (communications load, criticality and project priorities), represented in Figure 2.16. Crystal Methods highlight people, interaction, community, skills, talents, and communications as having the primary impact on project success—the team must work well together. Process is deemed secondary, and should only be barely sufficient. However, it should be adapted to and evolve with the specific team (and its unique talents), the project and the domain to be effective. Considering people, interactions, etc. reduces the need for heavy, expensive methodology elements. Crystal Methods outlines a number of methodology design principles. Improving communications channels between people and delivering a working product more frequently can reduce the number of intermediate work products required. This is supported by using incremental cycles, with a maximum period of four months. The team must adjust the methodology to match each project, especially as it evolves. This is supported by "reflection workshops". The elements in a methodology are broken into thirteen categories: roles, skills, teams, techniques, activities, process, milestones, work products, standards, tools, personality, quality, and team values. These must all be considered when creating and adapting

the methodology. In following the goals of lightness, agility and bare sufficiency, any methodology element that does not help the team work together effectively should be discarded. A methodology can easily become heavy as it is "embellished" with increasing numbers of ever more complex elements. It is noted that greater methodology weight may still be appropriate in projects with critical software. Cockburn describes his Crystal Clear methodology in the book "Crystal Clear: A Human-Powered Methodology for Small Teams" [22].

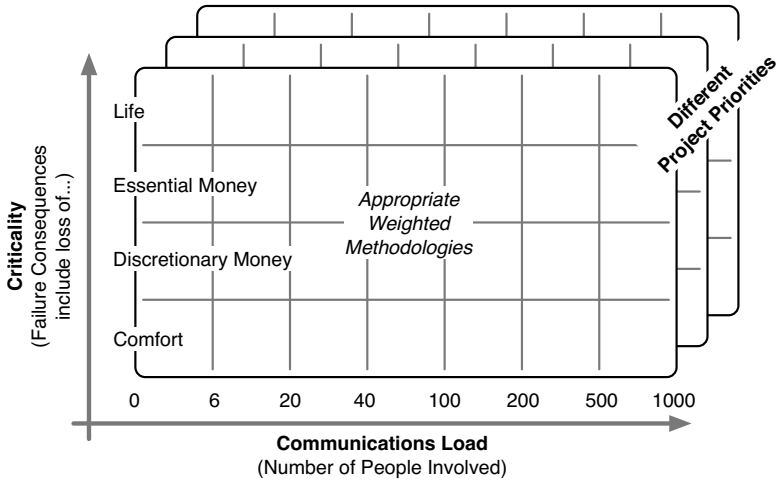


Figure 2.16: Representative selection matrix for Crystal Methods.

Feature-Driven Development Feature-Driven Development (FDD) is a set of simple process guidelines for agile development based on short feature-driven iterations. Five processes make up the FDD software development "system", as pictured in Figure 2.17. First, an overall model is developed by senior team members to describe the high level architecture and provide a basis for feature design and modeling. This model also communicates understanding about the system design and project domain to all team members. It is suggested this phase take about ten percent of total project time, with some additional rework throughout the project. Next, a list of small features is developed through an orthogonal functional decomposition of the overall model. The features should be a vertical slice of the system that can be implemented in two weeks or less and provide business value to the customer, who should be involved in the process. Features that remain too

large must be decomposed further. It is suggested this decomposition be restricted to four percent of the project time, with some ongoing rework. After the feature list is set, the team plans the project schedule by feature in multiple levels. High level business activity milestones and completion dates sit on a month or year scale, with packages of business activities and individual features being assigned to sub-teams and individual developers to be managed in two week increments. Planning must consider feature order and dependencies, risk, complexity, work-load balancing, client-required milestones, and checkpoints. While this planning is important, it should be revisited frequently, with up-front time (two percent suggested) matching ongoing revision effort. Finally, the design and build by feature phases iteratively address the planned feature list by sub-team to deliver features. Related sets of features have their design and models refined to create design packages. These design packages are then implemented, inspected, tested and integrated into the overall system. The design and build phases are expected to use over three quarters of the total project time. In comparison to other Agile methodologies, FDD shows greater structure through more traditional hierarchical teams led by senior members, more limited explicit customer involvement later in the process, and an emphasis on modeling and planning early in the process. Changes that arise are accepted through the allowance of rework of the planning phases, but management must ensure that changes and the schedule remain compatible. The skeletal modeling that addresses architecture and domain early is targeted at reducing design rework and also allows tasks to be distributed to more independent and potentially specialized sub-teams. This structure more readily scales to larger projects and teams with fewer adaptations. Another interesting aspect of the emphasis on upfront and by-feature modeling is that executable models are encouraged, bringing the models closer to prototypes or working code than design documents that will become outdated.

Lean Development Lean Development considers agility and change tolerance a competitive advantage in a turbulent marketplace. It is built on the themes of creating a top down culture of agility, and that change should not only be tolerated but leveraged in fast-moving, high-risk "exploration" scenarios to create profit. This is branded as moving from risk management, which looks to protect against loss, to "risk entrepreneurship". The ideas are based heavily on those

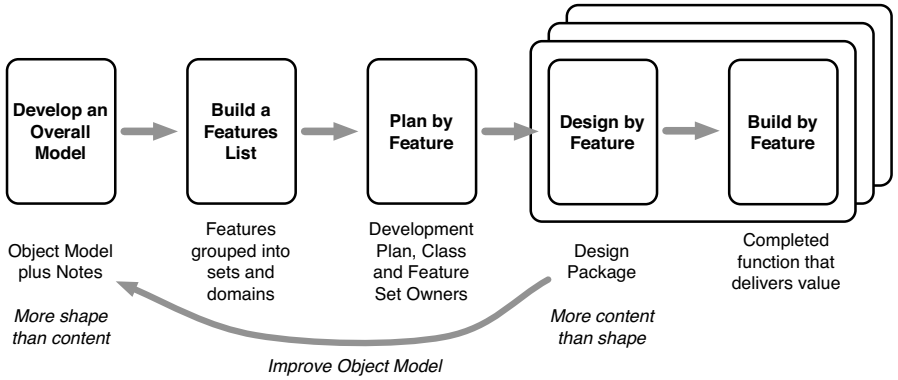


Figure 2.17: Feature-Driven Development model.

of lean (or craft) production, which focuses on using highly skilled generalists and simple and flexible tools to create what the customer wants instead of the mass production approach of using unskilled workers and single-purpose equipment to replicate a design created by specialists. There are four critical success factors driving Lean Development, which lead to twelve principles, which together may be summarized as follows. The primary goal of this methodology is to provide value to the customer which is achieved with teamwork. Satisfying the client demands close interaction and collaboration with the customer. Value can be further improved by creating change-tolerant software. This includes carefully considering what may change and reducing the cost of making likely changes in the future during design and development. Extending this, an organization should look to create domain solutions that are applicable or adaptable across multiple projects. The team should also create only what is necessary in an effort to deliver value to the customer more quickly. This may be aided by purchasing rather than building some components to improve speed. Customers usually prefer receiving eighty percent now instead of one hundred percent in the future, which may be difficult to achieve in rapidly changing markets. Minimalism is another key to agility, which is helped by reduced paperwork and reducing the effort to communicate. For example, documentation often adds little value, but uses significant resources and time, while small, co-located teams can communicate easily. As with other agile methodologies, Lean Development is well suited to specific problem domains, and will not be appropriate in all projects.

The Lean Development environment focuses on three parts - the technical foundation of the software and architecture, the policies and guidelines for managing the effort, and the processes, methods and tools used. From a process standpoint, Lean Development is typically broken into three phases. The startup phase looks to evaluate project feasibility (technical and business) and understand what the customer values in an effort to reduce risk. The steady-state phase uses time-boxed iterations to complete analysis, design, testing, integration and implementation. These time periods should be less than ninety days and preferably less than sixty days, with careful evaluation between iterations. The steady-state phase finishes with the delivery and acceptance of the product. Finally, the transition and renewal phase addresses the continued evolution of the product as well as knowledge transfer activities like documentation and training.

Extreme Programming Extreme Programming (XP) outlines a concrete set of practices targeted at small, colocated software development teams, backed by core values and principles. To be successful, the culture of the organization must be aligned with the five key values and principles: communication, simplicity, feedback, courage, and quality work. XP looks to create an environment suited to creativity and communication through the disciplined use of these complementary practices. The practices are designed to encourage interaction between developers and with the customer, prioritizing face-to-face communication over documentation to enhance understanding of the problem domain. The design should be kept simple but tolerant to change, and continuous and rapid feedback is desired. Courage alludes to the idea the team and it's members should strive to do what is right and remain disciplined even when under competing pressures (like deadlines). Quality work is the product of the team's fundamental attitude and culture, combined with an appropriate definition of quality, whether this is value based or quantifiable (like no defects). Extreme Programming describes twelve complementary practices that work together to make a project successful, summarized as follows:

- "The Planning Game: XP targets three week iterations, using "stories" as feature descriptions created through collaboration between the customer and the development team to outline a release plan that gives a high level indication of scope, cost and overall schedule. Customers

drive value by prioritizing stories, and the overall plan should be updated continuously."

- "Small Releases: Software should be released frequently in small increments containing the most valuable business requirements. When a full release is expensive due to installation, training, and migrations, deeming the project releasable may be a reasonable alternative."
- "Metaphor: A high level description or vision is used to keep the project and it's features on track as well as improving the overall understanding for both developers and customers."
- "Simple Design: The system design should be the best and simplest to meet only the functionality that has been defined, limiting anticipatory design as much as possible."
- "Refactoring: XP encourages continuous refactoring of existing code as features are added or modified to ensure the design remains simple and robust, and to reduce degradation of existing code (for example through quick fixes or dirty hacks)."
- "Testing: The extensive and frequent use of unit and functional tests helps reduce defects, especially with refactoring. Test-driven Development (TDD) fits well with the XP development lifecycle of listen (to requirements), test (written first), code, design (or refactor, as design grows and evolves). Tests should be as automated as possible."
- "Pair Programming: This is an intense programming practice in which two developers work together at the same terminal on the same code, with the goal of simultaneous coding, code inspection, and learning or knowledge transfer. It is similar to a code then inspection cycle, but maximizes the feedback frequency."
- "Collective Ownership: A complement to Pair Programming, the idea is that all code in the project belongs to the team rather than an individual, and that any developer can modify and improve any of the code, rather than assigning specific portions of the code to individuals. The goal is also to avoid blame, reduce ego and encourage learning through greater (but appropriate) risk-taking."
- "Continuous Integration: In conjunction with the testing practices, increasing the frequency of

building and testing all code in the project helps to get feedback on and find defects (especially integration related issues) as early as possible."

- "40-hour Week: Sustainable productivity and quality is possible if developers are enthusiastic and committed on their own, rather than being pushed to the limit and forced to work (more than a week of) overtime by unrealistic schedules (overtime being defined as time in the office the developer does not want to be there)."
- "On-site Customer: Continuous interaction and collaboration between the customer and the development team must be easy, which is achieved by having the customer on-site to allow face-to-face communication and rapid feedback."
- "Coding Standards: The intensely collaborative nature of pair programming and collective ownership necessitate the team uses coding standards for consistency."

Extreme programming targets collaboration, minimalism and simplicity, and care must be taken when removing from, adding to or substituting this complementary set of practices if the methodology is to remain effective.

Adaptive Software Development Adaptive Software Development (ASD) was developed by Highsmith and draws influence from complex adaptive systems theory. It is focused on change and continuous adaptation using foundation of agile practices and people. ASD bears many similarities to other agile methodologies, and considers using the product the best way to learn how it should evolve, with rapid iterative development cycles improving "good enough" requirements and design. The methodology suggests a dynamic, iterative lifecycle of Speculate, Collaborate, Learn as being more appropriate in projects of high uncertainty and change instead of the linear, static Plan, Design, Build cycle typically found in a waterfall approach. Speculate encourages exploration of complex problems with the understanding that uncertainty demands that plans be updated and revised. Collaborate alludes to the fact that the information exchange in complex projects requires a team effort. Learn highlights that knowledge is never complete and must be augmented with regular feedback from customers and the team itself. This lifecycle has six characteristics, being

mission-focused, feature-based, iterative, time-boxed, risk-driven and change-tolerant. A focus on mission over detailed requirements helps guide the project direction and tradeoff decisions even when requirements are fuzzy and likely to change. ASD prioritizes customer-oriented features as deliverables over the general completion of tasks. Fixing the delivery of projects and iterations by time-boxing the iterations is important for focusing and forcing trade-offs, and should not cause overtime or corners to be cut. The iterations should be adaptive and planned in part through the analysis and management of risk in the project. The ASD lifecycle is illustrated in Figure 2.18. Note that there are some explicit pre- and post-iteration aspects to the speculate and learn phases. Project initiation is used to set mission and objectives, understand constraints, establish the project organization, identify and outline requirements, make initial size and scope estimates, and identify key project risks. The use of Joint Application Design (JAD) sessions is encouraged to help gather requirements. It is important to provide an initial estimation of schedule (total project time-box), as well as the quantity and size of individual iterations. It is noted these tasks should be revisited regularly at each adaptive cycle planning phase and revised as appropriate. This second phase is dominated by developing an overall theme or objective for the next iteration, and assigning a set of features that can be delivered within the time-box. During feature assignment, the development team works to evaluate effort, risks and dependencies, while the client should decide priority. Note that a preliminary, rough plan of some or all iterations may be completed, not just the current iteration, to continuously improve the plan. Concurrent feature development allows the technical team to deliver the features in a collaborative and concurrent manner, with managers facilitating rapid communication and interaction. Tools beyond face-to-face discussion may be necessary as a team grows or becomes distributed. This interaction may include other stakeholders and cover not only technical challenges, but business requirements and decision making as well. The quality review encourages learning from several sources: the customer's view of the quality, the quality from a technical standpoint, the effectiveness of the team and the practices in use, and the status of the project relative to the plan (which is used to adjust the plan). Customer focus groups are suggested for obtaining structured feedback from clients. Reviewing the architecture regularly and continuous code reviews provide insight on technical quality. The author suggests waterfall

approaches make learning and feedback more difficult, as the "do it right the first time" philosophy discourages experimentation and backtracking. In addition, team members may benefit by sharing work in progress early, to help find mistakes or small problems that can be fixed before they become larger.

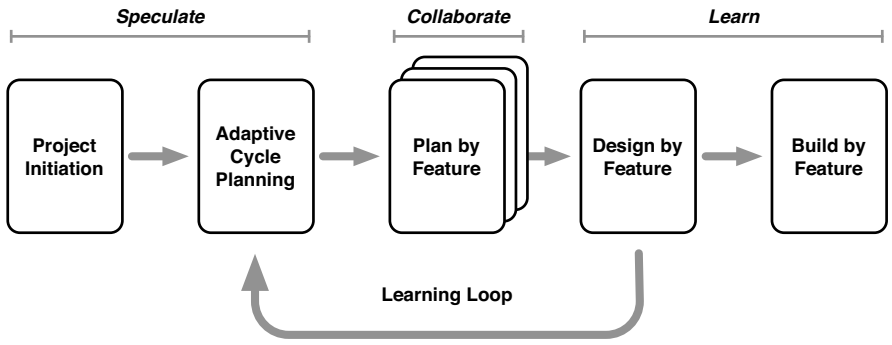


Figure 2.18: Adaptive Software Development lifecycle phases.

In support of Adaptive Software Development, Highsmith specifically describes the importance of a shift in management style. Traditional hierarchical command-and-control management focuses on optimizing to achieve a more efficient and predictable process through rigour and control. It is suggested that agile projects greatly benefit from a Leadership-Collaboration style of management, with a focus on adaptability over optimization. Such an approach looks to balance rigour and stability with flexibility and improvisation, described as "barely sufficient". It requires a cultural shift whereby managers are leaders that provide direction and guidance, facilitate connecting people and teams, and help to create environments for innovation, creativity, and effective decision making. It is suggested practices such as risk management and configuration control help support rapid and collaborative decision making.

Developing an Agile Software Development Ecosystem

Based on the previous efforts to understand agile values and principles and an overview of several established agile methodologies, Highsmith outlines the steps to create an agile software development

ecosystem and describes elements that should be considered when developing one.

Initial Steps The first step is to understand the opportunity and problem domain, and organization culture, and use these as a basis for selecting or designing a methodology. Either agile or rigorous approaches may be appropriate depending on the domain and culture, and the author organizes methodologies into three broad categories: adhoc or none, agile or rigourous. Highsmith suggests that the domain and culture be evaluated in a manner suggested by Geoffrey Moore [57]. Moore describes four different market types: Early Market, Bowling Alley, Tornado and Mainstreet. Mainstreet markets are traditional and stable, while the others typically work to bring a project to this stable zone. Projects in the different market types are characterized by differences in objectives, uncertainty, risk profiles, criticality, and constraints. Similarly, there are four different organizational cultures: Cultivation, Competence, Collaboration, and Control. The underlying core values associated with an organization's culture should be matched to the methodology and the project domain. Understanding the culture is important, as it has significant impact on how difficult business decisions are made. Culture may vary across an organization or stages of a project.

Early market is characterized by early research and development, which would be unlikely to benefit from any significant methodology (adhoc or light agile may be appropriate). Bowling Alley and Tornado markets may be volatile and demand a methodology suited to rapid change, as well as supporting close collaboration with customers or clients to obtain feedback and adapt quickly, which fit well with agile methodologies. The stability associated with a main street market lends it to a rigorous approach to support optimization.

The Cultivation culture focuses on individuals and tends to be self organizing, though it can be difficult to scale due to a resistance against structure and process, making it best suited to an adhoc or perhaps light agile methodology. A Competence culture focuses on skills and technical excellence driving a high performance team that may not have much tolerance for process, fitting well with agile methods. Collaboration cultures are often built on cross-functional teams that benefit from strong leadership and enhanced interaction with limited capacity to accept rigour, making agile

approaches a good option. A Control culture emphasizes predictability and planning, which better accommodates rigorous methodologies.

Highsmith suggests that different methodologies may be fitted to the opportunity and culture as illustrated in Table 2.19. The author notes that while a Control culture is really only suited for a Mainstreet project managed in a rigorous manner, this constitutes a reasonably large number of projects. In cases where culture and opportunity are not well matched (for example Cultivation and Mainstreet), the only solution may be to divide the organization such that a more appropriate culture may be developed independently (for example, a spin-off company or new division).

	Control	**Cultivation**	**Collaboration**	**Competence**
Early Market	*Clash*	Ad hoc	Ad hoc/Agile	Ad hoc/Agile
Bowling Alley	*Clash*	Ad hoc/Agile	Agile	Agile
Tornado	*Clash*	*Clash*	Agile	Agile
Main Street	Rigourous	*Clash*	*Clash*	*Clash*

Figure 2.19: Highsmith's evaluation of market opportunity and compatible cultures [44].

Finally, Highsmith describes the tasks that guide methodology selection. First, the predominate culture of the relevant group must be articulated in terms of values and principles, an exercise which may in itself be both difficult and introspective. It may also be of interest to evaluate the compatibility of this culture compared to other groups and the overall organization. Next, the problem domains, projects or business opportunities that will comprise the work of the group should be identified and characterized. If the culture and domain do not seem compatible, a new organizational structure or major cultural shift may be necessary. Based on the predominant culture, a base methodology should be selected, which will hopefully be applicable to planned projects. Finally, for each project the group tackles, the applicability of the culture and base methodology or ecosystem should be evaluated. The base methodology may then be adapted to the specific project.

It is important that the development ecosystem be suited to the culture first, and the project second, as this is likely to be more successful than trying to change the culture using a methodology for a

specific project. It is certain that project characteristics will vary, and it can be beneficial to have a portfolio of ecosystems that are suited to both different team cultures and different projects.

Methodology Design Highsmith provides a detailed basis for designing an agile methodology. First and foremost, the author reiterates that an effective methodology must adapt to external influences as well as the team using it. The methodology must balance the flexibility desired to help exploratory projects excel with the stability and predictability necessary for effective business management. While a single methodology will not be effective for all projects, careful methodology design will encourage efficient reuse and adaptation. In handling variation among projects, a well-designed methodology will provide consistency and improve predictability, even if the process is not repeatable and the project has considerable uncertainty (as most product development and exploratory projects do).

Understanding Parts of a Methodology A methodology is made up of elements. Elements are the "what" of the methodology, described as items or components or traits that the team and methodology have or own, and can be written down, described, isolated, etc. The concept of elements may be extended to describe the ecology of the team which also makes up the ecosystem (ecosystem encompasses everything, potentially including multiple methodologies and the culture or ecology. Based on Highsmith's model, elements present in a team's methodology and ecology are described in Figure 2.20. Methodologies are characterized by size (the number of elements), ceremony (detail and formality of elements), weight (the product of size and ceremony) and the tolerance (how closely a team must adhere to the methodology; the capacity for variation and flexibility). Practices are the "how"—they outline the steps in a process or behaviours of the team or activities carried out, and use or create elements. These practices fall into three categories. Collaboration practices describe interaction between individuals within a team as well as those outside. Project management practices outline how the project is managed and monitored, describing the vision and helping to bound the work. Software development practices guide how the software is

designed, built and tested.[2] Some practices may fall into more than one category and a set of practices in a methodology should complement one another as a system. Different agile methodologies may focus more on some categories than others, leaving greater flexibility or adaptability in some areas. For example, while both Scrum and XP incorporate collaboration practices, Scrum tends to focus more heavily on project management practices and XP prioritizes software development practices. The principles and underlying values of the organization should guide the development of practices. Practices may be easily adjusted to fit different cultures, and the implementation of practices should always consider the intent of the underlying principles and culture to be useful. The author promotes simplicity in a guiding methodology, following lean themes of minimal process and instead relying on the collaboration of the team supported by a creative environment to generate value. "When a methodology becomes something to circumvent in order to get the job done, it impedes progress rather than assisting it". It is suggested that there are six key areas where practices are needed to create an effective agile methodology:

- Mission Elaboration helps define a vision that supports business goals to direct the project and maintain focus.
- Project Initiation looks to create a brief baseline estimate for scope, cost and schedule to be used as a guide by both the team and the customer.
- Iterative Cycles encourage regular feedback and should be feature-driven to provide customer value over short time-boxes.
- Constant Feedback allows comparison against vision and evolving plans helps keep the project focused.
- Customer Intimacy encourages close collaboration between the development team and the client to understand business values and prioritize accordingly.
- Technical Excellence and a quality product provide a foundation for creating value in the

[2]Highsmith specifically focuses on software projects, but this category isolates discipline-specific practices and helps modularize a methodology by making collaboration and project management practices more general in cross-disciplinary projects.

present and the future.

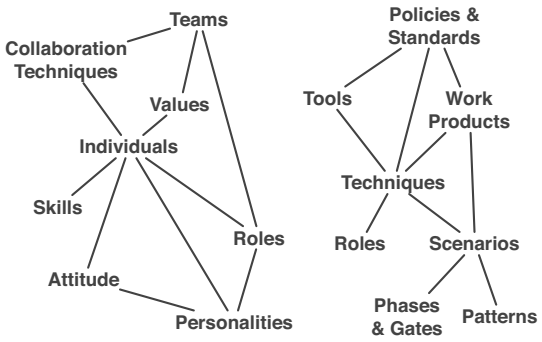


Figure 2.20: Highsmith's key elements in a team's ecology and methodology [44].

A final note on methodology tolerance is that it may be quantified by the number of policies present. A greater number of policies typically indicates a lower tolerance. Policies are parts of a methodology that are mandated by the organization.

Methodology Design Principles Highsmith provides nine principles to guide methodology design, quoted here:

- "Principle 1: Interactive, face-to-face communication is the cheapest and fastest channel for exchanging information"
- "Principle 2: Larger teams need heavier methodology"
- "Principle 3: Excess methodology weight is costly"
- "Principle 4: Greater ceremony is appropriate for projects with greater criticality"
- "Principle 5: Formality is not discipline, process is not skill and documentation is not understanding"
- "Principle 6: Increasing feedback and communication reduces the need for intermediate deliverables"

- "Principle 7: Efficiency is expendable in non-bottleneck activities"
- "Principle 8: Think flow, not batch"
- "Principle 9: Greater methodology ceremony may be required to meet regulatory, security, and legal considerations"

Larger teams or distributed teams may compromise Principle 1 and may also impact Principle 6, thus requiring a heavier methodology, especially regarding collaborative practices. Principle 3 drives the goals of simplicity and bare sufficiency. Principles 4 and 9 are very similar and address managing risk and meeting externally imposed requirements. Principle 5 suggests that the former emphasized in rigorous methods is actually supporting the latter, which is the focus of agile methods (e.g. formality supports discipline). Principle 7 encourages team members to focus efforts where it is needed most, while Principle 8 looks to streamline activities and process and reduce any queueing when possible.

Frameworks, Templates and Scenarios In an effort to leave behind a linear methodology and shift focus from prescriptive elements to guidelines, Highsmith suggests a phase and gate life cycle framework supported by problem domain templates and development scenarios. These should be customizable to match the organization, units within the organization, and project teams within units.

The use of a phase and gate life cycle framework helps to outline high level flow and important milestones. The author stresses that visual depiction of an iterative life cycle can be difficult and appear linear. Milestones provide important opportunities for high level monitoring and critical decision making, while also making it possible to synchronize projects for the integration of multiple subsystems. Examples of key milestones include project approval, project launch, iterative feature completions, and product launch. The author asserts that while decision criteria will certainly change, most projects fit into a basic set of phases: product management, project feasibility, project planning, iterative product development, stabilization, and deployment.

Problem domain templates are created by combining methodology elements targeted at a specific problem domain, addressing the variation of problem domains an organization may encounter. Project teams may select a template appropriate for the project at hand, and should customize the template to be most effective. This is encouraged with a light description in practice (i.e. the author suggests one to two pages). It is important to ensure the problem domain the template applies to is clearly identified. The Crystal methodology series mentioned previously may be likened to a set of problem domain templates.

Scenarios strive to describe how people work together as a team to produce results with narrative rather than prescribe a sequential list of tasks for individuals. They emphasize the interaction and collaboration both within the scenario as well as between scenarios and within the template or phase (especially in a non-linear manner). Scenarios may share phase input information or contribute together to generate phase outputs. Between scenarios, interaction may reflect dependencies, iteration or cooperation. Small projects may be described in a handful of scenarios while large projects will require more. Scenarios focus on interaction, but may describe activities and reference other elements of a methodology, like techniques, roles, patterns and work products. It may be possible to reuse individual scenarios across different templates and life cycle frameworks. The methodology applied to specific projects is readily adapted by substituting or modifying the scenarios to accommodate unique characteristics of the project.

Collaborative Methodology Design Steps As mentioned previously, the first steps in designing a methodology include evaluating the organizational and project team culture, values and principles, and then understanding the problem domain. Project objectives and characteristics must also be determined. Objectives and characteristics may be evaluated in a general sense with consideration of the typical projects taken on by the organization or team, and on a per project basis when customizing the methodology for use on a specific project. Only then can a life cycle framework, templates and scenarios be developed successfully.

When looking at project objectives, business value and trade-off priorities should be considered.

Understanding business value and how the project fits into long term strategy is important when articulating the priority and balance of project cost, schedule and scope, which will drive important and potentially difficult project management decisions throughout the project. Project characteristics that must be understood to properly apply a methodology include team size, criticality, risk and uncertainty, and activity scope. Activity scope describes which parts of a development life cycle will be addressed by the project. For example, an early research and development project is unlikely to include any product stabilization or deployment activities.

Finally, the methodology may be described. Explicitly defining the principles provides a basis for understanding the other parts of the methodology. It also leads to a brief description of the intent of the methodology, and a brief description of its scope. The phase and gate lifecycle should be described, after which specific domain templates can be identified. A key part of domain template design is the design of the scenarios that make up the template (along with other elements). Then, a description of each element in the methodology templates may be presented. The author suggests limiting a methodology description to maximum of 25 - 50 pages to maintain simplicity and brevity. When creating the methodology, it is important to avoid falling into a traditional methodology description: focus on how to work together and collaborate, which may reference skills and knowledge, but do not explain skills or convey knowledge through detailed prescriptions. Ensuring the project team has the skills and knowledge required is a separate business and management activity from methodology design and application.

Customize Templates to the Team Once a methodology has been established, with a base life cycle framework, templates targeted at specific domains built from a collection of scenarios and other elements, it may be used by project teams. As previously stated, the methodology should be customized to match the team and the project at hand. Involving the team in the customization process ensures everyone understands and may contribute to the method they will use, at least the more detailed aspects thereof. The author suggests numerous ways that the methodology may be adapted. One strategy is to start with a bare skeleton domain template and embellish with elements and practices appropriate for the team and project that address shortcomings. Additions

add weight, and each addition should be carefully considered against base principles to make sure it will contribute to team performance. Practices must remain simple to avoid degeneration into prescriptive procedures and task lists. Another strategy is to make adjustments to a more complete domain template based on variations from the target domain and differences in team composition. An example of domain variation is whether a team is co-located or distributed, which would certainly require adjustment of collaboration practices. Team collaboration will also need to be adjusted for differences in talent, skills, knowledge and personalities. Additional customization may be applied to work products. Ceremony or formality may be increased or relaxed to match project characteristics. Increasing traceability increases effort, but may be necessary to match project criticality. The importance of work products being up to date and reflecting the current state of the project should be considered. Some work products may be living and constantly updated, while others may need to be permanent and static. The author maintains that collaboration and interaction practices should be the focus of customization instead of the ceremony of work products or other elements, especially as a team grows.

After the methodology template has been customized at the outset of a project, its scenarios and practices should be regularly adapted to better reflect the actual behaviour and practices of the team. Team retrospectives are a good opportunity to do so, and should be conducted at a rate on the same order as development iterations.

Scaling A main theme in the applicability of agile methods to projects and organizations is that most methodologies are targeted towards small teams. Typically, as projects and teams become larger, additional elements are added to and ceremony is increased in the methodology, with a shift towards optimization and a heavier methodology. It is important to balance the increase in process with improved adaptive practices as well, supporting effective collaboration and interaction. When scaling collaboration practices, care should be taken to maintain a minimum of overhead to make sure they continue to add value and support the team instead of controlling or limiting the team. When scaling software architecture and development practices, it may be necessary to divide a large project into sub-teams focused on loosely coupled modules, coordinated in part through a

high level architecture team which is responsible for continuous architecture design and integration, with integration feedback used to improve the design. A balance between up front anticipatory design and rework-driven adaptive design must be made, taking care to consider the rate at which different design decisions or architectural components may change. Collaboration between the subteams and architecture team must be tight and efficient to achieve efficient knowledge transfer across the entire project.

ASDE Summary

Overall, Highsmith provides excellent insight into what it means to be agile and how to approach the adoption of agile development methodologies, especially as applied to software development. Agile is about the underlying values and principles held within the culture of teams and organizations, and the overall approach to problem solving taken by them, not about following a specific agile methodology. With themes of adaptability, collaboration and simplicity, Highsmith outlines the key parts that make up an agile methodology, and explains in general terms how create one that fits within the culture and problem domains of an organization. Though the author previously describes a number of popular agile methodologies that may be adopted, there is emphasis on customizing and adapting the methodology to fit specific projects and teams. Finally, the reader is reminded that agile approaches are not appropriate for all projects. They excel when projects are focused on exploration over optimization, characterized by high uncertainty and rapid change, where the ideal path and outcome is difficult to predict, making detailed plans tenuous at best.

2.3.3 Kanban

Anderson and Carmichael [6] provide a brief introduction to Kanban and describe its components, and also provide insight into applying Kanban to an organization. Originally, the term kanban[3] was used to describe the workflow implementations developed in Japan by Toyota as early as the

[3]"Kanban" refers to the [software] development methodology, while "kanban" is the Japanese pull-oriented inventory management system or more broadly, a reference to the physical board.

1960's to limit work in progress in lean manufacturing facilities by relying on a demand-based "pull" strategy. Kanban has more recently been used to describe a flow-based project management methodology that adapts this "pull" strategy to improve understanding of the work and workflow, encouraging change and adaptation. The use of a kanban board to visualize work and flow aids in this understanding. The target domain of Kanban is services delivering intangible knowledge work products to customers.

The Kanban method rests on a foundation of nine values, three agendas and six principles. The values are transparency, balance, collaboration, customer focus, flow, leadership, understanding, agreement, and respect. In a kanban system, work represents a flow of value towards the goal of providing the customer with value. Three agendas provide high level purpose to driving change in the organization. The Sustainability agenda focuses on improving efficiency and achieving sustainable results within the organization. The Service Orientation agenda looks outwards and strives to deliver and improve upon services that meet and exceed customer expectations. The Survivability agenda is concerned with continuous evolution and embracing change to remain competitive. The six underlying principles of Kanban are divided into two groups. Change Management principles include starting with (and respecting) what the organization does currently, pursuing improvement through evolutionary change (using the starting point as a baseline for assessment), and encouraging acts of leadership from individuals at all levels within an organization. The Service Delivery principles are to understand and focus on the needs and expectations of the customer, manage the work and allow people to self-organize around it, and evolve policies to improve customer and business outcomes. These principles try to ensure the organization's management focus remains on the work and the flow of value to the customer instead of managing the people doing the work.

The flow of a kanban system is visualized on a kanban board, as illustrated in Figure 2.21. Key parts of a kanban system include the use of visual signals that limit the Work in Progress (WIP; the collection of and count of items in the system), as well as clear commitment and delivery points. Commitments are tacit or explicit agreement indicating the customer wants and will accept an item and the service provider will produce and deliver the item. Delivery points represent completion

of the item. Items for commitment may be selected through a process from a group of ideas or requests (similar to a Scrum product backlog). The Lead Time (LT) of an item refers to the time between commitment and delivery, while the Customer Lead Time of an item refers to the time between request and delivery, which may be different if the customer "pushes" request instead of exclusively accepting "pulls", or if there are additional services in between the kanban system of interest, customer requests and customer delivery. The Delivery Rate (DR) may be described using Little's Law, where the mean Delivery Rate is the mean Work in Progress divided by the mean Lead Time for a steady state system (i.e. between two points of same Work in Progress). The same law may be used to describe a number of other throughput metrics of interest in a kanban system, where instead of Lead Time, Time in Process, Test, Development, System or Queue can provide insight. Visually, a Cumulative Flow Diagram illustrates Little's Law, as shown in Figure 2.22, and it may be observed that limiting Work in Progress will optimize the Lead Time.

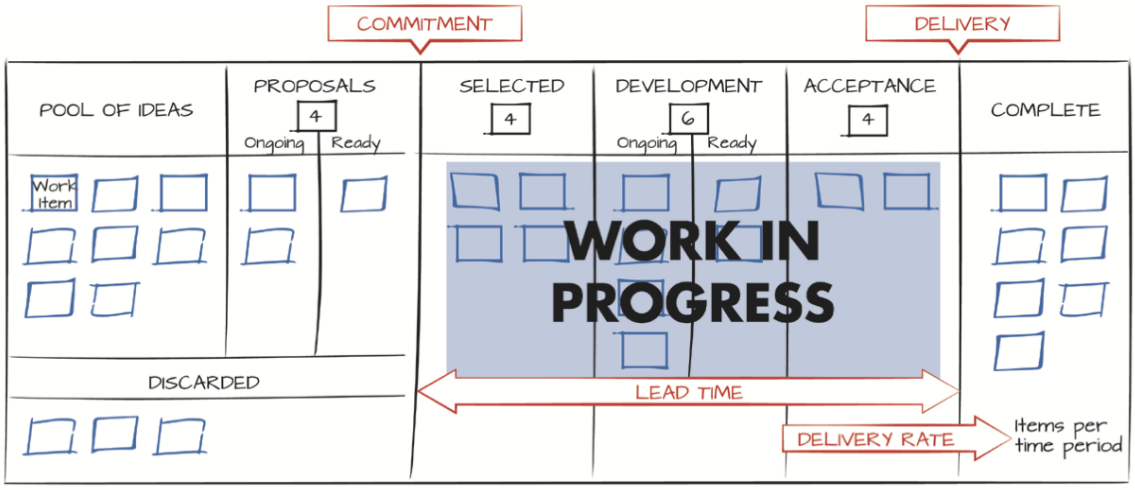


Figure 2.21: Example kanban board. Adapted from Figure 4 of [6]. Copyright 2016 Lean Kanban University Press. Reproduced by permission of the publisher.

There are six general practices in the Kanban methodology representing key management activities. These practices are quoted here:

- "Visualize"

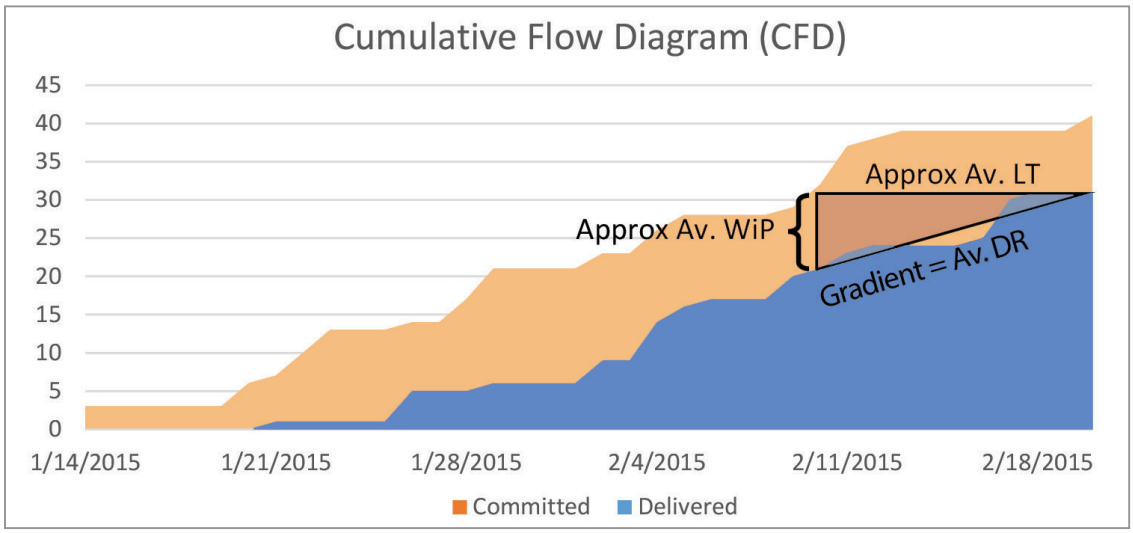


Figure 2.22: Example cumulative flow diagram. Adapted from Figure 5 of [6]. Copyright 2016 Lean Kanban University Press. Reproduced by permission of the publisher.

- "Limit Work in Progress"
- "Manage flow"
- "Make policies explicit"
- "Implement feedback loops"
- "Improve collaboratively, evolve experimentally"

Visualize is often accomplished with a kanban board showing the flow of work horizontally through a series of steps as illustrated above in Figure 2.21. It is important that the work items, commitment, delivery points, and Work in Progress limits be included. It is also good to visualize policies (especially regarding how the work items move between steps) and highlight any dependencies preventing progress. As mentioned previously, limiting Work in Progress helps optimize Lead Time which improves customer responsiveness and the ability to adapt to changing circumstances. Managing flow in a kanban system to maximize value, minimize lead time and provide some predictability can be difficult, and attention must be paid to bottlenecks and dependencies.

The delay cost and urgency of work items help prioritization. Different delay cost models may result in the application of different classes of service (with varying policies). Simple and minimal process policies help constrain and guide work within the flow, and should always be applied, but also be modifiable such that they may evolve to provide greater value. Examples where policies may be applied are for Work in Progress limits, definition of done for delivery, capacity allocation, class of service selection and work item replenishment. Kanban implements feedback loops with Cadences, referring to both meetings or reviews, and the time period between reviews. A minimal Kanban implementation may use replenishment and daily kanban meetings (like a daily scrum), while a larger organization may implement a more complex scheme, like the example in Figure 2.23. The final practice reflects that Kanban is designed to help an organization improve, and to encourage continuous evolution and adaptation. A Kanban methodology can not be considered "complete", rather it must constantly change to help an organization fit into and excel within its environment.

There are several steps to introduce a Kanban system to an organization, including how to evaluate the adoption and effectiveness of the system in key areas. Probabilistic forecasting tools and complementary monitoring methods can improve predictability and aid planning efforts. Finally, the authors explain approaches to scaling Kanban across different parts of an organization.

2.3.4 Scrumban

Scrumban [51] originally started as a method to transition from a Scrum methodology to a lean Kanban approach, but became a distinct methodology in its own right. In fact, Reddy [73] suggests it is a good method of facilitating change and introducing Scrum and Agile to teams. Scrumban draws on elements of both Scrum and Kanban to achieve a flexible flow-driven lean approach within a more structured development framework. It is primarily targeted at software development, like other agile methodologies.

Compared to Scrum, Scrumban differs in several ways. While self-organization principles encourage

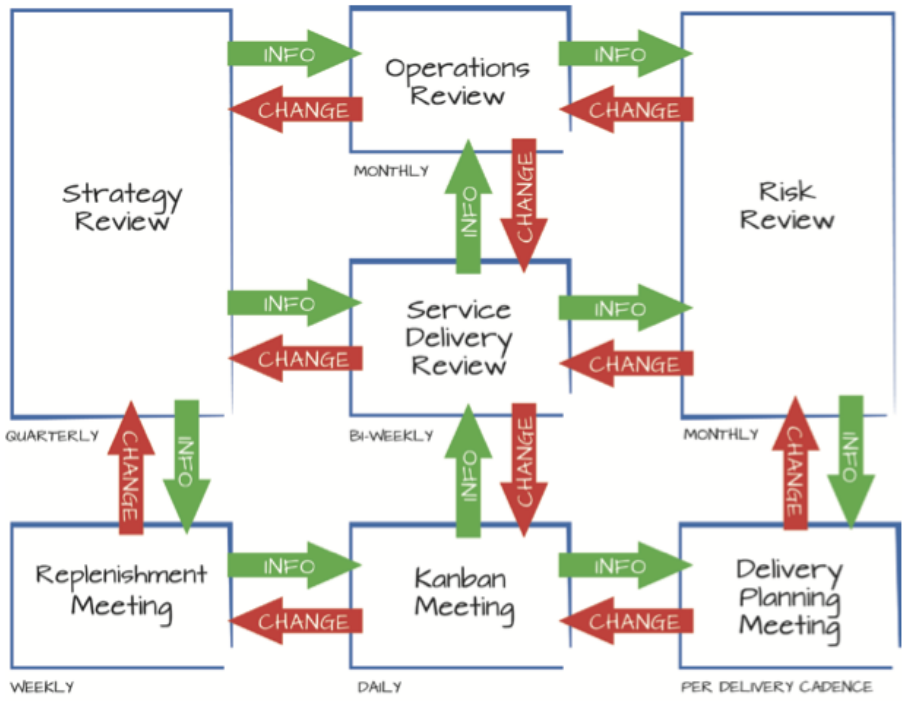


Figure 2.23: Example of Kanban cadences (feedback cycles) for a large organization. Adapted from Figure 10 of [6]. Copyright 2016 Lean Kanban University Press. Reproduced by permission of the publisher.

by Scrum remain important within defined boundaries or domains, Scrumban asserts that organizational management avoided in Scrum can still provide value. Scrum emphasizes a team of generalists, but Scrumban allows and sometimes encourages specialization with teams and functions (though does not enforce this). Scrumban helps to define how to work with explicit policies, and applies the laws of flow and queueing theory.

Scrumban varies from a Kanban approach as well. It prescribes Scrum as a basis for the software development process and is organized around teams. While Kanban strives towards a fully pull-based flow system, time-boxed iterations remain a useful part of Scrumban when applied appropriately. Finally, specific ceremonies are used to give more structure to continuous improvement techniques.

Reddy suggests that Scrumban provides a number of opportunities to diverge from the standard Scrum process to address shortcomings or challenges that often arise when Scrum is implemented. Examples of these include addressing emergency or priority work mid-Sprint, integrating a self-organizing team with traditional management structures, better leveraging specialized skills of individuals or existing teams, and more effective continuous improvement through additional objective performance measures.

As the number of tasks that a resource is responsible for simultaneously increases, total productivity falls as a greater amount of time is spent in context switching [96]. Scrumban suggests improving the flow of work and overall productivity by using Work in Progress limits supported by work buffers to improve quality, focus efforts and constrain distraction and overload. As a result, risk management and prioritization should improve. The principle of focusing effort and prioritizing effectively using Work in Progress limits extends from the individual to the team and finally to the organization as a whole (to limit strategic focus and prioritize opportunities). It is noted that the goal of improving flow and productivity of the whole system (team or organization) may result in a decrease in the utilization or productivity of individuals, which is contrary to the traditional measure of productivity being individual utilization.

Scrumban links risk with priority and the impact of not completing tasks by a time limit. Assessment of risk and impact is an important part of prioritizing work in a team. Four common risk profiles are suggested: urgent or emergency, fixed cost, standard cost, or intangible or investment cost. An urgent risk profile suggests a task is very high priority and will not be completed under normal circumstances—a team needs to stop other work and expedite these items. A fixed cost risk profile is medium to high priority, and failing to meet a deadline is detrimental. However, it is expected the team has time to complete these tasks within a limit if planned for accordingly. A standard cost risk profile indicates immediate but shallow impact where tasks do not have an explicit deadline but still incur cost due to delay. It is suggested these tasks are addressed by a first-come, first-served approach. Intangible or investment cost profiles do not typically have significant direct impact but are necessary tasks to support the team's progress (for example, addressing technical debt or

improving the architecture, i.e. rework).

When implementing Scrumban, the first step is to create a kanban board representing existing workflows and practices, considering inputs, outputs, process and links to outside stakeholders or systems. The initial board design should be simple; it can evolve over time. The team should begin using the board to reflect actual work items, and flow can begin to be measured and evaluated during daily stand-up meetings using metrics and visualizations of Work in Progress, Blockers, Lead Time, and Throughput. A cumulative flow diagram also provides insight. Next, introducing limits on Work in Progress can help stabilize the system, a necessary step before significant improvements can be made. Additional improvements to the workflow system can be made through the understanding and management of risk, of which the Cost of Delay is particularly important to address. Risk management methods improve the delivery of value to the customer with the reduction of wasteful activities. Further insight and improvements can be obtained by quantifying the value of work to be performed in terms of cost and revenue, and using this as part of work prioritization decisions. The visualization of the value and various system metrics helps bring understanding to the whole team and should be integrated with the kanban board system flow visualization. Finally, continuous improvement and evolution should become part of the culture of the team, aided in part by the use of team retrospectives to reflect and introduce new concepts.

2.4 Analysis and Quantification of Design Methods

Previously, we have seen a range of design project management methodologies, both traditional ones that are typically applied to physical products, as well as agile methods that have been primarily targeted at software development. Each of these methodologies may be categorized by a number of methods, one of the most important being the role of iteration. The methodologies may also lend well to models and simulations to help understand and improve the processes, especially with regards to coordination amongst the team. Furthermore, design methodologies fit into the overall product lifecycle, and tend to have similar design stages regardless of discipline or industry.

2.4.1 Investigating Engineering Design Process Iterations

Safoutin [80] investigated the use of iteration in the design process empirically by instrumenting engineering design software to investigate the use of iteration by engineering students in a simplified design class. The author observes that others consider the ideal engineering design pattern has no iteration (i.e. a waterfall model), and that iteration is generally viewed in a negative light as inevitable corrective action to fix mistakes that cause delays. Safoutin asserts that iteration is necessary and can be beneficial, especially as a way to provide feedback. A model that describes three different types of iteration seen in design is presented: repetition iteration, progression iteration and feedback iteration. Figure 2.24 illustrates the iteration approaches described. Repetition iteration is described as successive increments of the same activities leading to the final outcome with no intermediate outcomes. Applied to design split into features, this represents a top-down, depth-first design approach. Progression iteration looks to create intermediate outcomes of the whole at increasing levels of detail. This is a top-down, breadth-first approach, covering an entire design with progressively more detailed results. Feedback iteration extends progression iteration to explicitly integrate results of or information from outcomes at each level of detail to achieve greater accuracy in the final design outcome.

From the standpoint of mechanical product development, Safoutin brings to light that iteration is inevitable and even beneficial in the face of the traditional waterfall mindset. Safoutin also suggests that each approach, depth or breadth first, provide different advantages in the understanding of a complex design. The suggestion of using a progression iteration approach with feedback between levels of detail still reflects a traditional waterfall design project methodology, following concept design through detailed design. This is to deal with the potential interdepencies between features, where completing one feature completely may have negative impacts on others. The recommendation to include feedback is helpful, as is the idea of intermediate outcomes (i.e. a completed concept design or initial prototype). However, it is possible to approximate an Agile approach to the same model. A feature-driven methodology (as is adopted by most Agile approaches) is similar to repetition iteration, where a feature is rapidly brought to deliverable completion. To

address challenges of interdependency and incorporating future knowledge into an existing feature, Agile uses two approaches. First, rework is encouraged. Integrating new features may necessitate a change in an existing feature. The cost of this rework is reduced by considering this possibility in the original design, but it is also acceptable for time and resources to be allocated to this activity. Second, features and subsystems are designed to be as modular and decoupled as practical, which reduces impacts of changes in one feature on another. While some projects may not be usable until fully complete, many can begin showing value with partial completion. This is especially the case with software projects, but may also be present in other disciplines.

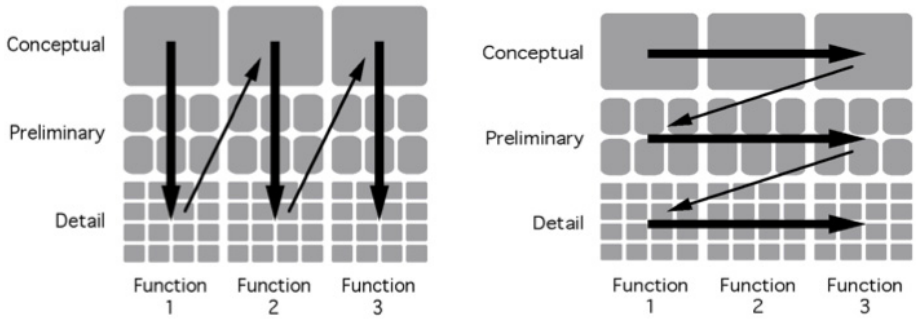


Figure 2.24: Incremental *(*left) and progressive *(*right) design approaches. Adapted from Figure 3.4 of [80]. Copyright 2003 Michael John Safoutin.

2.4.2 Design Process Improvement with Model-based Approaches

Wynn [104] presents a detailed formal modeling framework for the design process targeted at the planning and analysis of complex product development. This framework provides the basis of custom software developed by the author to aid in the planning and simulation of large projects with many hierarchical, sequential, relatively independent task streams. This model-based approach tries to address limitations of a traditional Gantt chart-based methods around task ordering and resource allocation, especially in the face of uncertainty and iteration. The author used product development program at Rolls-Royce to make observations useful in creating the modeling framework and specifying the software (now known as Cambridge Advanced Modeler, or CAM [18]), and later in applying the software to a real case. The software and modeling approach was deemed suc-

cessful in reducing process modeling cost, improving stakeholder communication regarding project management, and providing insight based on analysis into making the management process better. It is noted the software is limited and does not provide for concurrent or strongly linked tasks or dependencies, nor does it consider adaptive task selection based on feedback of the current project state.

Three perspectives for classifying models of designing are used by Wynn. The first classifies design processes into either stage or activity oriented. Stage-based approaches are more structured and tend to be sequential and chronological with limited feedback. Activity-based design models tend to be cyclical and rework intensive. In the second perspective, problem-oriented approaches typically focus on heavily on problem analysis before moving to generating multiple solution options. Solution-oriented methods move to a single solution early, which is then analysed and modified in an iterative manner during exploration of the design space and requirements. The third perspective categorizes design models as abstract, procedural or analytical. Abstract methods provide high-level, general guidance but provide limited support for specific process improvement. Procedural approaches provide more relevance to practical situations and specific aspects of the design process, and may be either design-focused or project-focused. Analytical approaches look to model a design process instance in detail with the goal making analysis possible that can be used to improve the process. This method leverages a modeling framework in conjunction with techniques, procedures or computer tools. The author suggests stage-based approaches are most often solution-oriented, while activity-based may use either. The abstract methods are often activity-based, but may be either problem- or solution-oriented. Procedural approaches are most often problem-oriented and stage-based. An analytical method can be designed to fit any category.

Wynn furthers the characteristic categorization of design methodologies. The desire to improve project planning and feedback capability beyond what is possible with a Gantt chart reflects a shift to greater process control. However, Wynn's target environment and the explicit limitations of the modeling framework and software implementation still suggest a traditional management approach for independent, sequential, lightly iterative work products. The analytical method is targeted at

prescriptive and procedural project management methodologies and does not provide the ability to adapt to the project and team. Agile approaches may be generally categorized as activity-based and solution-oriented, but certainly bring elements of abstract, procedural and analytical approaches. Further, stage-based and problem-oriented elements may be present or useful in some cases, but are perhaps less suited to exploratory problem domains. Agile methods try make practitioners see the system as a whole, while providing guidance and practices for specific situations and making sufficient analysis and data possible for feedback and continuous improvement.

2.4.3 Coordination in Complex Product Development

Suss [87] investigates the coordination between and within teams involved in large product development projects through modeling and simulation in an effort to improve engineering design methodologies. The author's approach is similar to that of Wynn [104], but extends task modeling to consider uncertainty and knowledge generation within a task, as well as coordination and interdependence between tasks from an information standpoint. This addresses some of the shortcomings outlined in previous work. Using the higher fidelity stochastic modeling approach allows the Suss to investigate relationships between and impacts of coordination strategies, iteration and rework, and uncertainty when trying to reduce product development time. Large projects are broken down into small, specialized tasks connected by pooled dependence (independent, parallel execution contributing to end goal), sequential dependence, and reciprocal dependence. Pooled dependence is effectively handled with rules or standards, specifying the end result ahead of time. Sequential dependence is addressed with planning, while reciprocal dependence benefits from feedback and mutual adjustment. It is suggested that the less routine the task, the more a feedback approach will improve outcome in the face of diversity. Further, as uncertainty increases, coordination strategies should shift from rules or planning to more meetings and greater horizontal communication bandwidth. Suss models both epistemic and stochastic uncertainty (of information) of design process tasks. The epistemic model follows an S-curve to reflect a tendency of faster knowledge gain in the middle of a task. As part of the simulation series, the author attempts to model a Scrum style agile

product development process in addition to more traditional product development methods.

Based on the simulation results, Suss concludes that following a Scrum-like development process, with well-defined intermediate outcomes and short feedback and review cycles helps improve coordination and focus within a development project team, leading to reduced design rework and iterations, especially in the face of uncertainty. The author asserts that an Agile approach is beneficial to all types of design projects and product development, and not limited strictly to software development. Improved coordination and reduced development time is supported by a number of mechanisms outlined in the work. Project planning, policies and systems should contribute to the reduction of project risk by scheduling uncertain tasks early which reduces overall uncertainty sooner. They should also be designed to increase communication bandwidth and information flow within a project to reduce bottlenecks that can impact development time in a nonlinear manner. Overall, the results suggest that Agile methods of design project management offer advantages over traditional approaches.

2.4.4 Design Modeling and Functional Modeling Across Disciplines

Eisenbart et. al. [32] reviewed literature from multiple disciplines to develop a framework that can be used to compare design models or documents across disciplines. The design models from various methodologies are used to capture information about the design at a specific point in the progress of the design process, i.e. the design state. The authors identified discipline specific and generic design states that can be categorized as either problem states (generally solution agnostic) and product states (solution specific information). The design models are analysed using a modeling morphology proposed by Buur and Andreasen [17]. The work highlights the similarity in design states across mechanical, electrical, software and building design, but variation in design models used to communicate the information. It is suggested that modeling method suitable for use across disciplines, as in the case of mechatronic systems, is needed.

Building on this, Eisenbart et. al. [31] review a range of functional modeling methods used across

different disciplines, which showed a diverse range of approaches and perspectives. Further analysis of functional modeling suggested a transformation process perspective is central to modeling across multiple disciplines, which could provide a basis for an integrative modeling framework if other project-specific perspectives could be included as needed [33]. Finally, Eisenbart et. al. propose an Integrated Functional Modeling (IFM) framework based on the concept of multi-domain matrices that can be used for effective cross-discipline system conceptualization, with sufficient modularity and extensibility to adapt to different projects [34]. This framework is based around the transformation process perspective, but incorporates six other perspectives as well. The framework identifies related entitities and core views to support these perspectives. The IFM framework continues to be extended and is under active development.

2.4.5 Stages in Product Lifecycle

Qureshi et. al. [72] evaluated a wide range of design methodology and product lifecycle management literature to obtain a common framework of design stages useful in collaboration and communication during cross-disciplinary (transdisciplinary) product development. The design stages in this framework were split into two tiers, inspired by the major activities or stages more typically relevant to product lifecycles and the greater detail often presented in design methodologies, which typically only cover a subset of the product lifecycle. They were also influenced by the transdisciplinary design states suggested by Eisenbart et. al. [32]. These lifecycle and design stages are listed here:

- Establishing a need
    - Market Analysis and Forecasting
    - Identification of Need
    - Project Management
    - Requirements specification
- Design
    - Conceptual Design

    - Embodiment Design
    - Detailed Design
    - Production Systems Development
- Implement/realize
    - Manufacturing
    - Assembly
    - Systems Integration
    - Procurement
- Use/Support
    - Sales and Distribution
    - Installation
    - Operation
    - Service and Maintenance
- End of life
    - Retire/Dispose/Closeout

Using this framework, a transdisciplinary industrial empirical study was completed to understand design processes used in industry compared to those presented in literature [71, 39]. This study showed the usefulness of the common framework to describe design processes and product lifecycles in industry across multiple disciplines and contexts or markets. Participants were able to match the processes of their organization to the general framework stages quite well. Also, there tended to be a clear tiered approach to process flow with most organizations using a stage-gate approach at a higher level, while adopting a more iterative process within stages. It is noted study focused primarily on large organizations (17 out of 23 organizations) with large design teams (250 to 1000 employees).

2.5 Agile Applied Beyond Software

Limited examples are available demonstrating the use of Agile approaches in non-software design projects, though there are numerous explanations of how a team may attempt to do so in the form of Internet articles and blog posts.

2.5.1 Project Suitability for Agile Methodologies

Alite [5] argues that Agile methodologies are not suited to all projects. An overview of the characteristics describing a project that may benefit from an Agile approach is presented, as well as attributes or project types that are better suited to more traditional approaches. Characteristics of the project, the team, the client, and organization are all important. Projects often have high levels of uncertainty (especially in requirements), complexity and risk of failure, and the technical foundation or platform should be flexible. The team should be small with a large portfolio of skills, and committed to quality, disciplined testing, and communicating with the client. The client should be heavily involved throughout the project and comfortable with agile concepts. The organization must support the agile approach, and the development work should be completed within the organization and team (ideally the team should be co-located). Alite establishes several additional important characteristics including the project champion's high tolerance for risk, an experienced development team, the ability of the end user to adapt to change to follow rapid evolution, and good flexibility within the project budget.

Examples are presented of software development projects that the author suggests do not fit well with an agile approach, including databases, embedded software, computationally complex projects, and incident response teams. It may be argued that while additional care or modification of process may be necessary, these projects are readily handled by an agile approach. It is likely that Alite recognizes these projects may need additional upfront design and architecture work prior to coding. This is easily included in an agile process while keeping with agile values and principles. In the case of an incident response team, a Scrum method may not be effective due to the potential of rapidly

changing priorities. A Kanban or Scrumban type approach is likely much more appropriate.

Alite makes an interesting observation that Agile methodologies are more complex than traditional methodologies. It may be argued that while the overall process description is often more detailed for an Agile approach than a waterfall type process, the ceremony and detail given to intermediate artifacts and work product is likely much greater in the latter. It is obvious either method will be more complex than an ad hoc approach.

The author summarizes the process of selecting an appropriate methodology by suggesting questions that may be used to help in the decision making process, as quoted below:

- "Are the requirements clear, can they change?"
- "Does the client fully understand and are they comfortable with the method the project would employ?"
- "Is the project [champion] comfortable with the methodology and what is their risk attitude?"
- "How will the development process take place? Is it done in-house, outsourced or distributed development[?]"
- "What are the development team characteristics?"
- "How fast does the product need to be operational? Is full functionality required in the first release?"
- "What are the quality, time and cost requirements?"

These questions help a potential Agile practitioner determine project characteristics and evaluate suitability for an Agile methodology. Alite suggests that selecting an appropriate methodology may benefit from additional research in the influence of the different characteristics on project success, and a way to quantify an Agile approach against traditional approaches in terms of quality, schedule and cost for a direct comparison. Finally, challenges associated with working with clients new to Agile should be addressed.

2.5.2 Agile SYSTEMS ENGINEERING versus AGILE SYSTEMS Engineering

Haberfellner [42] tries to clarify and define difference between an agile design process (i.e. agile management) and an agile design (i.e. design features), especially in the context of systems engineering. It is suggested an agile design process helps address uncertainty during development (like ambiguous customer requirements), while an agile design considers uncertainty and potential changes throughout the lifecycle of the product or system. An agile process strives to be adaptive and responsive to new information throughout development, rather than freezing requirements and solutions early as a traditional approach may encourage.

Within the context of systems engineering, Haberfellner outlines how to determine the need for agility by understanding the problem domain, and then describes three levels of agility within the design process. Greater agility is needed depending on:

- if requirements are less stable,
- what requirements may change, how much, and in what direction,
- and the cost of achieving a broad range in the requirements at the outset (rather than being able to change focus later).

Three grades of agility are described with the first grade simply being the introduction of some agility and increased iteration to the existing systems engineering design process. The second grade suggests the use of piecemeal engineering the use of modular designs to encourage reuse and reduce coupling. The third grade introduces the use of set-based design, whereby multiple concept designs, perhaps developed by independent design teams in parallel competition, are brought as close to completion as possible in an effort to explore as much of the design solution space as feasible within the time, cost and requirements constraints.[4] These ideas are similar to those promoted by an Agile methodology, but certainly more limited within the existing frameworks that large systems engineering projects typically inhabit.

[4]Set-based design is further described by Singer et al. [83].

Drawing the distinction between agility in process and design is an interesting idea - the former is typically detailed heavily by literature supporting Agile methodologies, while the latter is referenced as part of the goals of technical excellence and quality and the ability to rapidly adapt the design to meet changing requirements and business values. The author explicitly describes some of the properties that an agile design should possess. An agile design should be flexible, reconfigurable, and/or extensible, which may manifest as the ability to add or change modules to vary performance, or be scalable meet capacity demands. Such features are important especially if the lifecycle of the product or system has uncertainty. Short lifetimes and stable operating environments both reduce uncertainty. On the other hand, systems designed for extended periods of operation may encounter an unanticipated change in requirements due to changing market or other parts of the environment, especially if the cost of replacing or upgrading the system is high. As mentioned previously, the cost of meeting a broad range in the requirements (i.e. designing for worst case scenario early) may be very costly. Haberfellner suggests instead designing for the flexibility to adapt across the range of worst case to best case. Such an agile design must have flexible elements to support fast and easy changes, a method of measuring when changes may be needed (e.g. sensors to monitor the system), and a method to help make decisions that balance the cost and benefit of making changes.

2.5.3 An Empirical Foundation for Product Flexibility

The concept of product flexibility is also explored by Rajan [68], who defines flexibility as "the ease with which a set of changes can be imposed on a given design solution." An empirical method was suggested to evaluate the flexibility of products, referred to as a Change Modes and Effects Analysis (CMEA), which follows the same concept as an FMEA. This technique looks to decompose a product into modules or parts and understand how each part may change to reflect variations in requirements or lifecycle, and how these changes affect other components in the system. This can be used to improve flexibility in the design. The author suggests the technique is useful in identifying key areas to focus on during redesign, as well as during benchmarking studies or market and competitor analysis.

A number of products were evaluated using the CMEA technique, and several factors were identified that improve product flexibility:

- Modularity and less integration
- Modules as part of the product, attaching externally
- Increased use of standard components and interfaces
- Directed partitioning of design into a greater number of elements (more components and functions)
- Reduced number of parts per module does not affect flexibility (but likely improves assemblability)

The author notes that the CMEA method can be time consuming due to the effort of customer review and company evaluation, and the lack of a generic metric for evaluating flexibility not specific to each case.

2.5.4 Applying Scrum to a Prototype Vehicle Controls Projects

Bonderczuk [14] describes the experimental adoption of a lightweight version of Scrum in a team of three developing software for use in a prototype vehicle control system. The team worked in a cross-disciplinary environment, including electrical and mechanical engineers working on other subsystems. The Scrum methodology was modified to fit the team. Changes included implementing the product backlog as a live specifications and requirements document rather than user stories. In addition, the team met intermittently as the project was extracurricular, making daily meetings impossible. Instead, sprint review and planning meetings were combined into a single event in conjunction with other subsystem team leads, while the communication achieved in daily scrums was instead accomplished through a collaborative whiteboard task list and SMS text messaging between team members. The author concluded that using a Scrum style agile methodology instead of the previous ad hoc method improved the communication, technical understanding, collaboration,

and clarity of goals. Also noted was that a large backlog task queue reduced the effectiveness of the development team, while the key collaboration advantage was in understanding current system state rather than referencing outdated documentation of requirements and specifications.

2.5.5 Scrum in Mechanical Product Development

Reynisdóttirr [74] explores whether an agile method can be applied to mechanical product development. This is accomplished through a detailed account of observing the introduction and application of a Scrum methodology to a mechanical engineering team working on a single project, jointly with a Scrum embedded software team. This qualitative study was carried out through observations, interviews and information conversations. The author attempts to understand if adaptation of the method is needed, what factors were critical to success, the challenges that were encountered, and the effects of two Scrum teams collaborating on the same project.

Overall, Reynisdóttir concludes that using Scrum to manage team carrying out mechanical product development is feasible with some adaptations of the Scrum framework, especially for projects with high risk, uncertainty and change. It was not clear if Scrum is better than other methods, but it was an improvement over an ad hoc approach. A particular challenge was in the delivery of a true working product each increment, which is far easier to accomplish in a software development project. It was also noted Scrum is not applicable to projects where little or no planning is possible, like in maintenance requests. A Kanban methodology may be a better choice in these cases.

The author outlines the adaptations to Scrum that were made by the Mechanical team. While Scrum encourages a fully cross-functional team of generalists, the mechanical team was more specialized and did not have all skills required to create final product.[5] Instead, the mechanical team worked with other teams assigned to the project. Sprint lengths were varied in contrast to the strict time periods of a normal Scrum framework to accommodate events such as lead times, field test campaigns and holidays. The standard product backlog was replaced by the team's release plan

[5]It was not mentioned whether the skillset was sufficient for a clearly bounded sub-project the team should be responsible for.

which was created collaboratively with the Product Owner. The Product Owner was responsible for prioritization and approval. Product demonstrations and reviews at the end of sprints included stakeholders outside of the development team and Product Owner for improved feedback.

A number of factors were deemed critical to the success of the project. Having an experienced Agile coach and dedicated Scrum Master helped the learning curve and effectiveness of the team. Support and understanding from upper management was also important; ideally, the push to adopt Agile starts at the top level. On the other hand, the team must also be committed to the method, and should contribute to the adaptation and improvement of the method, as suggested by Agile literature. The Product Owner should actively provide vision and direction to keep the development team focused. Finally, the colocation of the development team, as well as close proximity to other teams assigned to the project, improved productivity.

Specific challenges and shortcomings were outlined by the author. The development team found the work breakdown process difficult when taking larger design tasks and splitting into smaller items. Reviews needed improvement to be more effective. Team members were used to working more independently and required adjustment to the more collaborative approach. The Product Owner was not as available as the development team needed, and external dependencies were not always managed appropriately. Finally, the project was large and complicated, which is not necessarily the best time to experiment with a new methodology.

The mechanical product development team worked closely with an embedded software development team assigned to the same overall project. The software team was already using Scrum as part of their management strategy. Once the mechanical team adopted scrum as well, there was an improvement in coordination, communication and cooperation between the two teams. Synchronization of cross-team dependencies like testing was improved. Each team also attained a greater understanding of its impact on the other, and of how to achieve their shared goals.

2.5.6 Informal Online Resources

Using Scrum on Hardware Projects

Maccherone [53] claims that Scrum can be effectively applied in the the context of hardware development projects with some adjustments. In particular, expectations should be modified in the areas of "minimum marketable feature" and user stories.

Agile methodologies typically encourage the delivery of finished and usable features each iteration that cover an entire vertical slice of the product. The author suggests the "break-even" point between design and implementation favours more design due to the cost of manufacturing a physical product. Instead of creating a finished product each iteration, prototypes provide similar value. If this is challenging, even producing documents, designs, or experiments may suffice if they are used to elicit feedback from as close to the customer as possible.

User stories help manage requirements in Agile approaches. While these can be challenging to adapt directly to a hardware project, the underlying core of who, what, why in the story and conversations around creating the story are beneficial. Conversations among the design team and stakeholders help maintain vision and bring considerations such as cost into the design early. The who of a user story encourages thinking about both the end user and the user of the specific feature, and the value delivered. The what describing the feature must remain solution agnostic, while the why considers the underlying reasons for the feature to improve the flexibility and creativity of solving the actual problem at hand rather than meeting a specific specification. Hardware projects should prioritize features by business value first, rather than end user value. The user story method of describing features and specifications may also be augmented by other requirements management techniques. For example, (ab)use cases may aid in understanding security and safety concerns, protocols and interfaces may require greater definition and description, and operating environment specifications may be clearer expressed as metric/value pairs (if this is known).

The author encourages trying to reach a goal of single iteration prototypes with the aid of rapid prototyping capabilities (in-house or outsourced), though the process cost should be balanced with

the time cost. If this goal is unreasonable, breaking down prototype development into several stages may be an alternative. This helps to address lead times and shared resources. For example, prototype design and procurement may be completed in one iteration, and prototype assembly and test completed after an iteration is skipped to allow for fabrication and shipping.

Managing dependencies and critical path is typically handled with longer term plans and tools like Gantt charts, but this may not truly be needed. These activities are handled continuously through iteration planning.

Specialists or shared resources assigned to the project may not be available all of the time. Maccherone suggests that if possible, specialists be part of the team full time, and trained for more generalist tasks if the specialist task load is light. This reflects the Scrum preference of building a team of generalists and considerations of the cost of task- (project) switching. Accessing shared resources may benefit from an approach similar to that used for external suppliers.

Agile Development for Mechatronics Processes

Jackson [49] outlines a number of key points to consider when applying Agile methods to mechanical, electrical and systems design projects. Working software is analogous to functioning system prototypes. This must be supported by rapid prototyping practices to encourage fast iterations, experimentation and the potential to obtain quick feedback from failures and successes. Agile methodologies often encourage team accountability and collaborative design through interaction instead of documentation to achieve adaptability and speed. This challenges the "traditional accountability culture of engineering", where individual engineers take responsibility by stamping documents, which may lead to more conservative designs. A difficult challenge between software projects and hardware projects is that software may be continuously improved with updates and new releases, while hardware products are typically locked into a single configuration once shipped (except for software or firmware elements). This is a significant difference in paradigm, and the difference between a production version and prototype version of a product must be considered.

Thilmany [93] also suggests an agile approach is beneficial for product development, especially if requirements are not stable. Adopting agile is challenged by lead times and stakeholders that are not directly part of the design process (e.g. suppliers and manufacturers). The author suggests that agile may be adapted only to the initial design stages where the working product is centered around a CAD model, with analysis, simulation and virtual prototyping. Releases deliver the model to a larger project team for review and feedback, including manufacturing, marketing, and quality assurance.

Agile in an Embedded Development Environment

Myllerup [59] asserts that a Scrum style agile methodology is readily applicable to embedded systems development based on experience and provides observations and suggestions to ensure success. Traditional embedded systems development is done by functional teams (e.g. software team, firmware team, electronics team, mechanical team, etc.). A better alternative is to structure the organization into fully cross-functional feature teams with members from different domains. Teams are then focused on the finished product instead of just the goals related to their profession. Further, generalist skills and knowledge will be improved when collaborating on tasks outside of an individual's profession. The author provides two ways of dealing with lead times of fabricating printed circuit boards or other prototype hardware. As suggested by Maccherone [53], project prototypes may be broken into several clear and measurable sprint goals, such as "critical components sourced", "schematic ready for layout", and "floor planning done". When taking this approach, it is important to avoid "scrummerfall", with sprint goals being "phase 1", "phase 2", etc. A second option is to work to complete a subsystem (subcircuit) outlined by the block diagram to a testable prototype stage each iteration and integrated into the final design concurrently. Myllerup also suggests that the acceptance criteria (as related to quality and value) be consistent across all disciplines, even if the definition of done is tailored appropriately. Scrum normally only includes the Product Owner in backlog planning activities, but other stakeholders should be involved to obtain the sufficient level of knowledge and experience across all disciplines. Finally, the author suggests some additional up-

front planning through a number of spikes during the zeroth iteration targeted at delivering a set of rough block diagrams outlining the proposed architecture. These diagrams and the architecture will be refined in subsequent iterations, but provide a foundation for initial backlog development.

Landing Zones

Wirfs-Brock [101, 100, 102, 103] presents landing zones as an alternative method of framing requirements. This approach defines a range in measurable attributes to capture a minimum, target and outstanding results. Meeting a minimum set will provide a minimum viable product that supports the product vision. Narrow ranges may reflect requirements with either low uncertainty or high criticality. The method helps identify areas of risk, improve multi-dimensional trade-off decisions and show requirements trending towards or away from targets as a project progresses.

Within the context of Agile methodologies, landing zone-based requirements can help evaluate the releasability of a product by monitoring a small, meaningful subset. Minimizing this group improves understanding and visibility even in large projects. This follows the visualization principle of Kanban to help understand the state of the project. If the group of attributes representing the landing zone is too large, it may be beneficial to split them into multiple landing zones by category or functional area. The initial targets should reflect the short term goals of a project. Throughout an iterative development process, the attributes, ranges and landing zones should be revised with the rest of the plan. Balancing attribute precision and flexibility can be challenging, and should be guided by project vision and business value. Even though the requirements will undergo change, concrete targets help guide development and provide a basis for quantifiable verification.

Visualizing Agile Designs

Embracing change is a key agile value. Haberfellner [42] outlines features that make a design agile and easy to change. Wirfs-Brock [99] summarizes work of Foote and Yoder [38], which centralizes on the concept that some parts or layers of a software system are much easier to change than others,

and are also more likely to experience change. This idea was in turn borrowed from Brand's "How Buildings Learn" [15], where the concept was applied to buildings. "Shearing layers" illustrate the rate and ease of change of different parts of the system, as shown in Figure 2.25. The adaptation to software based on Foote and Yoder is presented in Figure 2.26. This characteristic of designs is readily extended to many other domains as well. To improve the change tolerance of a system, it should be designed such that the fast changing layers may be modified without being obstructed by the slower layers. Grouping together features and modules that change at similar rates will reduce effort required for rework (though changes to lower foundational layers will require significant effort). This insight supports the goal of modularity and reduced coupling within a design.

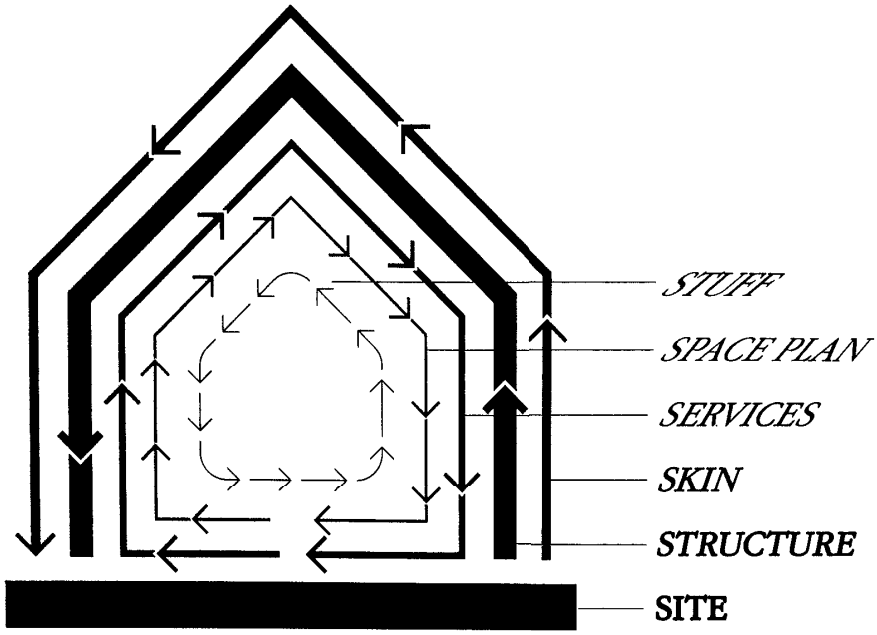


Figure 2.25: Brand's shearing layers in buildings. Adapted from "Shearing Layers of Change" by Donald Ryan in [15]. Copyright © 1995 Penguin Books. Reproduced by permission of the publisher.

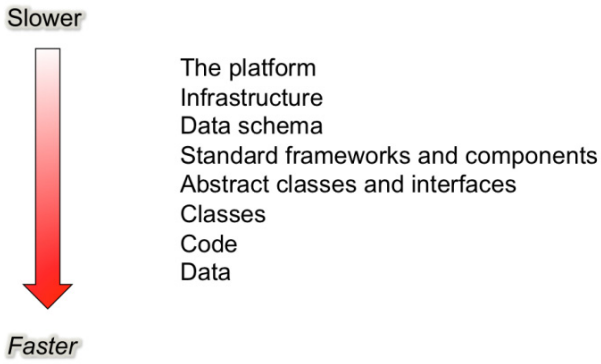


Figure 2.26: Adaptation of shearing layers to software systems. Adapted from [99]. Copyright 2011 Rebecca Wirfs-Brock. Reproduced by permission of the author.

Application of Scrum Methods to Hardware Development

Backblaze Inc. [9] presents some insight into the application of Scrum to hardware development. The example of the WikiSpeed project is described as a successful application of Scrum to a multi-disciplinary project, in which a prototype road-legal vehicle was developed in approximately three months with careful adaptation of agile software practices to automotive design [26]. Some of the challenges faced when adopting Scrum to hardware development are described, along with suggestions for adapting the Scrum method. The time and cost to create a working hardware prototype can be difficult to overcome when following an agile method in hardware development, especially when the per unit cost is high. In addition, a physical product is often less modular and more difficult to change, further increasing the cost and time of future iterations and rework. This results in a tendency to fall back to traditional methods in an effort to manage financial risk. The authors suggest this may be addressed by adapting the frequent release of a working product. Instead of delivering a functional prototype each iteration, a virtual simulation may provide similar value and opportunity for feedback while reducing the time and cost required to manufacture. This deliverable is also easy to modify. As an alternative, where modularity is sufficient, isolated subsystems or components may be delivered in each iteration, with additional sprints used for integration closer to the end of the project. Should the project or design have less modularity

than desired, it is suggested that the design could be changed to better fit an agile development methodology.

Helping Hardware Be Agile

Rothman [77, 78, 79] suggests approaching agile hardware development using a Kanban style methodology. There are different types of hardware present in typical product development cycles, including mechanical engineering, electrical and electronics engineering, and programmable logic (i.e. Field Programmable Gate Arrays). Each provides value to the project on different schedules and have different development cycle characteristics, heavily influenced by the cost of the physical embodiment. The author suggests organizing the project into functional teams based on appropriate hardware or software domain, with each team using a customized kanban board and backlogs (which should include interdependencies). The set of kanban boards allow each team to understand both their own and other's progress. Cross team integration is planned with the help of an overall project roadmap outlining feature priorities, interdependencies and ever more complex monthly demonstrations as deliverables. Increasing the number of integration points and demonstrations encourages collaboration and knowledge transfer to make appropriate tradeoffs and improve understanding of the project as a whole.

Coordinating appropriate demonstration and integration deliverables between teams can be challenging due the varying time and cost to create prototypes. Software elements are much easier to iterate on rapidly compared to electronics and mechanical hardware. The iteration cycle times must be synchronized to ensure feedback propagates across the entire project. Rothman discusses a few ways to reduce the time and cost of physical prototyping so these cycle times may be brought in line with those of the softer project components. Different types of modeling and simulation are suggested as low cost methods, while cost effective prototypes, models or mockups can be designed to validate a subset of design parameters (e.g. size, form, subsystem operation). When managing the risk of building a prototype, several considerations can guide the decision making process. The interim physical form should provide value, especially when considering the cost of this form and

the timespan for which the prototype is valid or useful. If a physical prototype reflecting the current state of the design does not provide value, an alternative prototype, model or demonstration should be considered that will through feedback and risk reduction.

2.6 External Design Project Constraints

As mentioned numerous times by authors describing Agile methodologies, part of the design project methodology may be constrained or prescribed by external influences. Main sources include laws, regulations, certifications, codes and standards. Engineers or other professionals may also be required to practice under the guidance of a professional organization as part of laws or regulations. In Alberta, professional engineers are self-regulating through the Association of Professional Engineers and Geoscientists of Alberta (APEGA), created through the Engineering and Geosciences Professions Act (EGP Act) [92].

2.6.1 Laws, Regulations, Certifications, Codes and Standards

Dieter and Schmidt [27] provide a detailed view on codes and standards. Codes are a "collection of laws and rules that assists a government agency in meeting its obligation to protect the general welfare by preventing damage to property or injury or loss of life to persons", and are usually a legal requirement. Standards are a "generally agreed-upon set of procedures, criteria, dimensions, materials, or parts", and are usually a recommendation. Performance codes describe a specific requirement to be achieved regardless of method, while prescriptive codes focus on the method, leaving no discretion to the designer. Standards may relate to performance, test methods or codes of practice. Codes and standards come from a variety of sources, including government, military, testing organizations, industry organizations, or from internal development proprietary to an organization.

Design projects that fall into domains that require high safety and reliability are subject to strin-

gent certification with respect to both the design and the method used to develop the design. In particular, medical, automotive, aerospace and safety critical control or monitoring systems fall into this category. Certifications may demand requirements traceability, a variety of testing and detailed design review, risk management strategies, and contingency plans in case of failure. Organizations may also opt to follow a specific standard (like ISO quality standards) to meet industry expectations or to improve efficiency, quality or value delivery.

Certifications are used to signify a design or product has undergone review (typically by a third party) to meet certain standards or regulations. A good example is Part 15 B of the Federal Communications Commission (FCC) regulations which limits the allowable amount of unintentional electromagnetic radiation emitted by electronics devices in the United States. Equivalent regulations exist from Industry Canada and other areas of the world. Industry consortiums may also provide compliance testing or certification to ensure a specification is met. For example, the Universal Serial Bus (USB) Implementers Forum facilitates development and compliance testing of USB devices [95].

2.6.2 APEGA

Engineering activities in Alberta are regulated in part through APEGA, who publishes a number of standards and guidelines pertaining to different responsibilities and best practices when engaging in professional activities. These are targeted at both individual professional members as well as permit holders (organizations or companies that practice engineering). A number have significant bearing on the methodology implementation that may be used by an organization or team working on mechatronics design projects. Several of these are summarized below.

Practice Standard for Authenticating Professional Documents

An important aspect of engineering practice is the authentication of documents used in engineering activities, whereby a licensed professional member reviews and takes responsibility for the content

of that document by applying a stamp, signature and date. This APEGA standard [91] outlines the method to determine what documents should be authenticated, when, and how. A basic process is suggested for use by permit holders, with additional information provided on storage, use, duplication and revision of these documents. A wide range of design artifacts may require authentication, including software, firmware and manuals or instructions. There are differences in the ownership and copyright of documents prepared by employees or contractors. Further consideration is given to the treatment of electronics documents, which require digital signatures. The management process for documents that require authentication should be considered when developing, using and improving a design project methodology and specific practices and policies, especially with regards to documentation.

Guideline for Professional Practice Management Plans

A Professional Practice Management Plan (PPMP) [90] documents the policies, procedures and systems used by a permit holder when conducting engineering activities. It is further described as a "management tool to document procedures for planning, implementing, documenting and assessing effectiveness of activities" carried out by an organization. The processes and policies described should support the values and vision of the permit holder, continued due diligence and professionalism, and achieving quality within the organization. Furthermore, the PPMP should provide the framework necessary to measure the effectiveness of methods used and supply the organization's management team with the feedback necessary to make regular improvements (it is suggested that yearly reviews and updates are appropriate). There are five main areas which must be addressed in the management plan.

First, there should be a description of the organizational structure and management responsibilities, especially with respect to the authority of responsible members and the distribution of technical responsibility across all team members partaking in engineering activities.

Professional members and permit holders must abide by the Code of Ethics and its standards of professional conduct, as well as remain compliant with the EGP Act. The PPMP must include a

clear commitment to this, and these underlying goals should be reflected in any of the organization's policies, procedures and systems.

A description of the available professional and technical resources and how they are managed should be present. This includes an inventory of facilities, equipment, reference materials, computing resources and software, IT policy, and business documents like contract templates. An inventory should also be taken of competencies and expertise of personnel. Furthermore, the methods of hiring, skills assessment, verifying appropriate levels of professional supervision, and training should be detailed. The method of handling work completed by others (i.e. outsourced work) should also be included.

The quality control section should cover a variety of topics that contribute to providing a professional, quality service. Quality control in professional business practices includes policies on intellectual property, confidentiality, promotion, contract negotiation, team coordination, and Loss Control and Risk Management plans or formal quality management plans. Quality control in technical work covers due diligence requirements, methods of documenting and verifying limitations in software, concepts, processes and procedures. Project management quality control outlines processes and procedures for scope, schedule and cost estimation, as well as project and organizational structure, and handling changes and reviews. Finally, the quality control of outsourced work should be described.

The last area a PPMP must cover is the controls related to professional documents and records, which represent critical information that requires review and approval by a professional member with appropriate experience and responsibility. These controls must consider the identification of, preparation, review, issuing, using, and revising of such documents and records. Further, documents must be archived and secured appropriately. These controls may also be impacted by codes and standards. The responsibilities of different team members for managing the documents and records must be communicated.

It may be argued that many elements of a PPMP closely resemble documenting a design project management methodology and ecosystem. Many of the suggestions for content and format given

are biased towards traditional management methods, but a number of underlying themes fit well with agile principles. As an example, the focus on creating estimates for scope, schedule and cost estimates and formally handling any changes suggests a rigid model, while the desire to document methods of monitoring effectiveness and providing feedback for improvements aligns with agile approaches.

Guideline for Management of Risk in Professional Practice

The management of risk is important in all parts and activities of an organization. This APEGA guide [88] suggests that risk management within an organization should be documented, even if the method used is simple and informal. It provides a detailed overview of the steps and methods to assess and manage risk as it pertains to the responsibilities of professional members. The responsibilities include those outlined by the Code of Ethics, and of due diligence and standard of care. The risk management process is illustrated in Figure 2.27. Risk management is an ongoing activity within an organization and across the lifetime of a project. In this sense, it reflects the goals of continuous feedback and improvement central to agile project management methods.

Guideline for Professional Responsibilities in Developing Software

The development of software as an engineering activity is described as inherently risky in this guideline [89]. The guide encourages carrying out software development activities using solid engineering and risk management methods, considering aspects of functionality, reliability, usability, security, and privacy. This is especially important in critical software or that used as a tool in other engineering activities. Software where errors or failures have significant consequences or may affect other work requiring professional member oversight should be treated in a similar manner, with appropriate review and authentication. Users of such software must be aware of limitations. Even if the application of the software does not make authentication a requirement, software development should be conducted using industry best practices and to appropriate standards. A number of resources are provided for reference. The guide also addresses intellectual property and

confidentiality. Finally, the guide provides guidance on adopting a software development methodology. While it is made clear different projects and cultures will required a variety of methods, it is required that "organizations must choose a reliable, proven methodology", and that risks associated with the methodology be actively managed. This requirement may limit evolution and improvement of methods, which is recognized, adopting an unproven approach will require greater risk management.

This software development guideline remains relatively neutral to the adoption of either agile or rigorous methods. Rather, it outlines the importance of understanding and managing the risk through the entire lifecycle of software. It highlights the responsibilities during software development should remain in line with any other engineering activities.

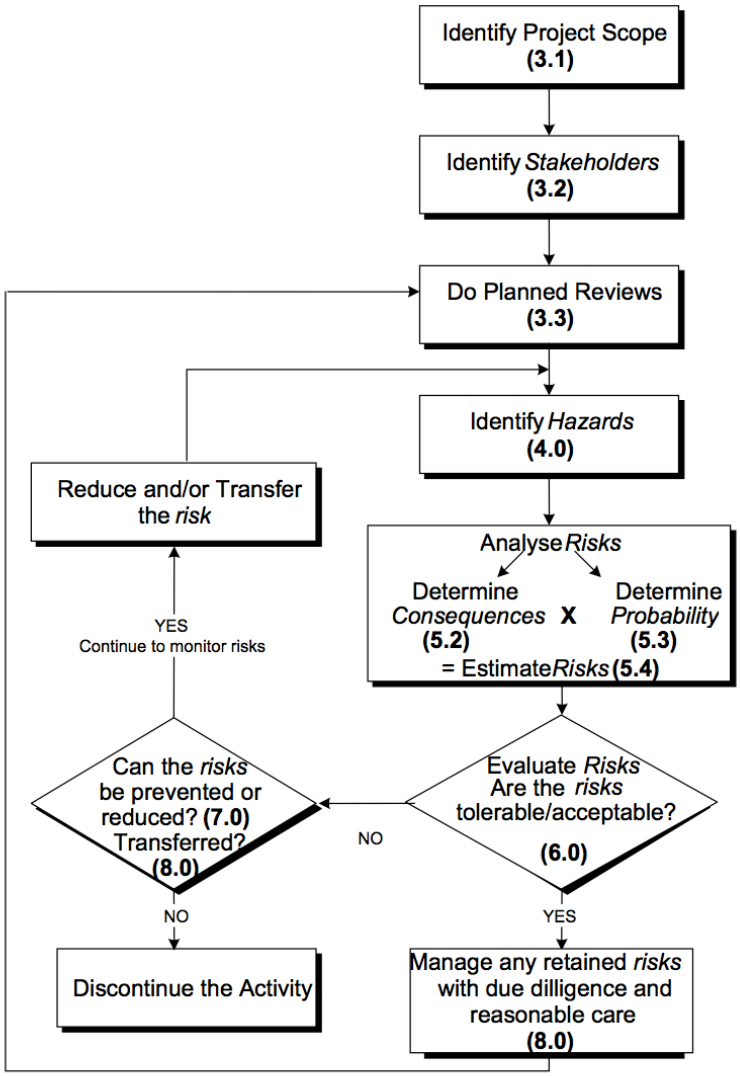


Figure 1 – *Risk management* flow chart

Figure 2.27: APEGA risk management flowchart. Adapted from [88]. Copyright 2006 APEGA. Reproduced by permission of the publisher. Note: currently under revision as part of APEGAâĂŹs recurring review process for all standards and guidelines.

2.7 Summary

There are a vast range of ideas on how to approach the design process and manage design projects effectively. New methods are continuously developed, and the open-ended nature of the challenges addressed by creative design activities can be handled in more than one way. Furthermore, the vast diversity in the applications of engineering design means the range in methods is justified.

First, a sampling of traditional design methodologies were reviewed. These methods are supposedly discipline agnostic, but tend to focus on the engineering design of physical products. Such products are typically mass manufactured or expensive to build, biasing to methods to address these issues with a primarily linear phase-driven approach.

Next, several approaches recommended specifically for mechatronics and embedded systems were considered. These approaches bring greater consideration to working with multiple disciplines in the same project. This includes understanding how to approach a problem in a way that does not commit to a specific solution, driving the design by feature and creating a modular architecture. Also of importance is understanding the interfaces and flow of energy or information between domains or modules. The methods described still use a relatively traditional approach to planning-oriented project management, but several alternatives—like the Spiral framework—are introduced, especially in the context of software.

Agile methods have traditionally been the domain of software projects as an alternative to rigorous software development approaches. Agile imparts an different view on project management, and strives for the regular delivery of value through working products, collaboration within the team and with the client, and adapting to change. There is a wide range of methodologies that follow agile principles and values. While often targeted at small teams, these typically do not consider the challenges of creating physical embodiments that are part of the outcome in mechatronics designs.

Efforts to understand the design process, especially the role of iteration and methods of modeling and simulating the process, are then outlined. Different types of iteration have been described, as

well as characteristics by which to categorize different design methodologies. Of particular interest was the conclusion obtained through simulation that agile-style methods with rapid feedback and iteration, along with good collaboration, are beneficial to reducing design rework and overall iterations during a design project, especially in the face of uncertainty.

Numerous sources describe how to apply an agile approach to a project with physical products. Topics discussed include evaluating project suitability, dividing tasks within a team, coordination between different teams, and alternative approaches to handling requirements. Other topics of interest are the meaning and application of agility in the contexts of both the design and the process of the project. Another theme is understanding the capacity for change at different layers, subsystems, or domains in a design, architecture or system.

Finally, methodology and design are impacted by external constraints like standards, regulations, certifications as well as laws or regulatory bodies like professional organizations.

Methodologies typically refer to the overall lifecycle of the product and where the design activities fit in. Key stages of a design project are also identified, with agile approaches at least describing an initial stage followed by a main iterative stage. Traditional methods typically break down the process into more phases related to the progression of the design.

In the methods reviewed, there is little discussion around explicitly incorporating risk and risk management into the design and management process. The exception is the Spiral design methodology framework, which focuses heavily on a risk driven approach. Agile Software Design (ASD) also incorporates risk into the process, while Scrumban considers risk for effective project task prioritization. Other methodologies integrate risk management more directly, due to its importance for safe, business-minded, effective engineering and project management. This is also a key topic driving external constraints.

Many traditional design process models trend towards addressing the problem and postponing the solution rather than finding a solution early and trying to improve it. The process is top-down, breadth-first, with attempts to finalize the design at increasing levels of detail. This is categorized as

progressive iteration [80]. Traditional design approaches do not expect that changes be achievable after the design stage of the lifecycle, or as the design matures. It is assumed that as time progresses, change is more difficult. While iteration in the design is deemed inevitable, it is not accounted for directly in most traditional approaches. Postponing the solution may also limit any insight or feedback available through early and rapid prototyping. Changes to a design later in the project used to represent high cost, but numerical simulation and inexpensive prototyping is shifiting the front-end load model for types of systems. Leveraging the feedback available from these methods early in a project may lessen the size and impact of changes later.

On the other hand, agile approaches recommend a top-down, depth-first method with vertical design slices (closer to incremental iteration [80]). The importance of iteration is outlined by most models, but while agile embraces rework as a way to link vertical slices, incorporate feedback and address change effectively, traditional methods discourage it, with most linear or progressive models limiting iteration to between adjacent phases. Linking vertical slices is also supported by effective modularization and a solid architectural foundation. The vertical slices or feature-based development favoured by agile approaches contrast with the phase-oriented nature of traditional methods. As agile methodologies have historically been applied to software projects, it is expected that the product or design can be changed easily during development, and possibly after release as well. The incremental iteration approach aligns with the goal of delivering a working product regularly that can provide value to and the opportunity for feedback from the client.

The iteration rate strategy adopted by different methodologies ranges from almost zero explicit iteration, to time-boxed iteration with various periods, through to a maximum level of iteration. Some key examples of iteration in several well known design strategies are described below.

The traditional waterfall approach strives for as little iteration as possible, although some have highlighted that this is not necessarily realistic [19, 35]. Of note is that the waterfall process describes the phases that a design should progress through with little emphasis on project management, which is built around the design steps and focuses on planning ahead in detail. This predictive approach can be viewed as a special case where the iteration cycle time is the maximum, i.e. the full

project period of interest. Note that the management approach and iteration rate may be different or separate from the design process cycle time. In some cases, design reviews within the waterfall cycle (e.g. between major phases) may be considered a type of iteration in the design process, as it is intended to provide some feedback in the project. The waterfall approach is readily applied to many engineering disciplines.

The spiral model, originally targeted at software development, more closely couples the design and management sides of a project, and certainly emphasizes an iterative approach to product development. Furthermore, it uses risk management techniques to drive development of a process model and decision making during the process. The model can be applied to generate design process models reflecting a range of methodologies (including waterfall) while still achieving risk driven goals. This model provided significant inspiration for the framework described in Chapter 3.

Scrum is one of the most well known contemporary examples of a highly iterative agile process for software development, as embodied by its short, fixed time-boxes. Scrum processes focus on project management, though the techniques and focus on feedback driven rework and change allow the process to support design iteration. While widely adopted for software development, the goal of releasing a working product after each iteration can be challenging to adopt in cases where physical products are being developed as in most mechatronics projects, especially when prototypes come at high unit cost. However, the underlying principles are readily adaptable to other disciplines.

The Kanban method pushes iteration to a maximum, and can perhaps be modeled as a special case of continuous, event driven feedback with a cycle time approaching zero. A paradigm difference exists with more typical time-boxed agile approaches where Kanban looks to pull tasks as capacity is available to maintain flow rather than committing to tasks (pushing) at the beginning of iterations. The pull paradigm may be likened to new iterations being triggered asynchronously by a variety of events like task completion rather than just triggering on time events like in a time-boxed methodology. Kanban helps support categorization of tasks more explicitly than Scrum which can in turn be used to manage the distribution of specialized skills. While the flow paradigm may be

effective within a team, time-boxing helps synchronize outside the core team. Collaborating with outside stakeholders and clients may benefit from having hard schedule points to plan around.

Iteration approaches can be different between design and management. Some methodologies focus more on either the design process or project management, while others consider both. The strategy and methods for each aspect of the project must be complementary.

Most methodologies highlight the importance of getting feedback and information from a variety of stakeholders. Traditionally, this is restricted to early in the project during requirements gathering or to later during testing after the design is complete. Agile methods encourage collaboration and feedback throughout the project. This follows the iteration strategies, as agile approaches strive to have a working, reviewable product that improves incrementally, rather than delivering a full system near the end of the project.

One of the main differences between traditional project management methods and agile or lean approaches is in the application of schedules, budgets and scope. Traditional methods try to monitor and control the project by evaluating against predicted cost and schedule. Agile approaches use value delivered as the core metric in evaluation, as predicting project progress is challenging, especially when the project changes or has uncertainty. Instead, schedules, budgets or other plans guide the project and help the team focus on the goals. This extends to driving the project with underlying principles and goals, instead of following steps in a process and delivering a set of documents.

Most methodologies target a specific project type or problem domain (for example, physical product design or software development). The methodology must be matched to the project, as well as the team. There is a large range in methodologies and in the characteristics of projects, and it is clear that a singular process or method will not be applicable to all cases.

A core characteristic of different methodologies is how change is considered. Projects that are exploratory have greater uncertainty and change, and are better suited to an adaptive approach, while projects that focus on optimization but have limited uncertainty and change may benefit

from a predictive, planned method. The agile principles embrace change rather than avoiding or controlling it in order to address the exploratory nature of projects, and this adaptive ability is incorporated into practices, design, management style, and other parts of the project. Design projects, especially for new and innovative products, are often rife with uncertainty.

Chapter 3

Proposed Agile Framework for Mechatronics Design Projects

One goal of this work is to create a framework that provides a foundation for applying an agile methodology to mechatronics design projects. The framework (and the associated methodology model) is meant to be flexible, drawing inspiration and values from various agile methods and mechatronics approaches, while still being able to deal with the realities of more rigid project constraints of physical products. The framework provides varying levels of guidance for implementing specific methodology elements and practices. Some parts are strongly suggested, while others encourage a mindset and thought process aligned with agile values. Examples of ways to implement a methodology are provided. The implementation should be customized and continuously adapted to match the team and the project. It is not suggested that this framework and methodology model will work for all projects or ecosystems, but is targeted at small teams working together on innovative and exploratory mechatronics design projects.

This framework focuses on several areas, including the product or system design process and lifecycle, the role of iteration, organizing the project team, the importance of project characteristics,

principles of agile design, achieving feedback and knowledge transfer, and incorporating risk management.

3.1 Product Lifecycles and the Design Project Cycle

Product lifecycles provide a high level overview of the general stages of a product, system, or project. The design process or design project cycle focuses on the scope of a single design project, and fits one or more times within the product lifecycle, usually closer to the front end.

3.1.1 Product Lifecycles and Business Models

In this work, the terms product and system both refer to the core deliverable outcome achieved in a design project and are mostly used interchangeably. A product is typically a singular unit where multiples may be produced to be used independently, while a system may reflect a collection of different elements that work together and may or may not be produced in duplicate.

Products and systems move through a range of stages as they progress from an idea through development to release and use, and later as they are removed from service. A set of stages and sub-stages that make up the product lifecycle in cross-disciplinary projects is described by Qureshi et. al. [72]. The complexity and interactions of the lifecycle for a single product or technical system is illustrated well by the lifecycle model proposed by Hubka [30]. Dieter and Schmidt [27] break a product lifecycle into two distinct phases: premarket and market. They suggest the premarket phase includes the generation, evaluation, research and development, and testing of a product with both technical and market goals, while the market phase sees the product enter, grow within, mature, and eventually leave the market, typically measured by sales. While the authors target manufactured hardware products in this model, it is at least partially applicable to other products and systems as well.

Typical business models seen in software projects (the historical domain of agile methods) include:

unit sale with no updates (except purchasing a new version outright), unit sale with periodic updates (which may be paid or free), subscription service with periodic updates, or subscription service with continuous improvements. Subscriptions may also be structured as pay-by-usage. Often a subscription service will rely on a central server controlled by the vendor to provide either core functionality or at least authentication. Unit sale software typically runs on customer controlled hardware. Update frequency may range between periodic and continuous, with continuous in this case referring to the practice of the vendor releasing new features and improvements as soon as they are completed by the development team, without queueing. Subscription models have become popular with improved access to the internet, and often present as a SaaS (Software as a Service) [84]. A final business model present in software products is based on indirect revenue, while providing the product or service free of charge or at a reduced or subsidized rate. Most commonly, the indirect revenue comes from embedded advertising or paid premium features.

Business models for physical products can be described in a similar manner. The most obvious is the traditional unit sale of physical products. As products incorporate more software and other changeable elements, or become reliant on an evolving system, updates and improvements become more relevant. There are a wide range of examples where unit sales with periodic updates occur, like the sale of vehicles (periodic maintenance paid or on warranty, recalls, aftermarket upgrades, etc.). A subscription service resembles a rental or lease arrangement, where the vendor maintains ownership and control of all parts of the system, and can update or improve over time. A good example is set-top boxes and modems for subscription television, phone, internet or other communications services, where the customer pays a small monthly fee to use the hardware, but the vendor may readily upgrade this hardware (or the software within it) as the whole system improves. Equipment rentals provide another example, where the vendor may maintain or replace assets regularly to ensure an appropriate minimum level of service for the customer. Finally, indirect revenue approaches typically see a subsidized or low cost product supported by the sale of paid, high-margin consumables that are required to continue using the product. This is readily seen in consumer printers (ink or toner) and single-serve coffee machines (single-serve packages).

A final indirect revenue approach is that most often taken with open source product offerings. The software and/or hardware design is offered freely and openly, with revenue coming from either the sales of hardware units or through value added services such as technical support, maintenance, consulting, training, commissioning, or customization.

The various approaches in software and physical products may be combined. In mechatronics systems, combining different models for various subsystems or elements may be beneficial to take advantage of differences in the ease of change of different elements.

One of the challenges with physical products is that when goods are manufactured, capital costs are incurred for both the setup and maintenance of the manufacturing system as well as the bill of materials that constitute each unit. Customizing a manufacturing system reduces the cost to manufacture each unit but makes changes more expensive and thus more difficult. This occurs primarily in the "market" stage and at the tail end of the "premarket" stage. When such an approach is taken, an agile development approach and this framework is better suited to the "premarket" stage, though some elements are applicable throughout.

On the other end of the scale, projects targeting a continuous release and improvement approach are well suited to the agile values of this framework. The lifecycle is supported by making it easier to change the product after release and throughout development, while also reacting to feedback from the market. Further, many agile methods push to see feedback from markets earlier and more rapidly, and more in parallel with development in a premarket stage, by releasing early and often with regular updates and benefits to early adopters and testers.

The business model that is used by the product being developed will impact where within the lifecycle the design project cycle will be. Some products and systems may undergo iteration in the design, leading to this framework being applicable at more than one point in the lifecycle. In some cases, the whole lifecycle may benefit from the application of this framework. It may be possible to adapt a project business model and management style to leverage this framework or other agile approaches throughout.

3.1.2 Design Project Cycle

The design project cycle is the interest area of this framework. The design cycle and the management of a design project is the topic covered in more detail in many of the references discussed in the Literature Review (Chapter 2). This framework draws inspiration from and adopts some elements from this previous work by others. A brief overview of the design project cycle in this framework is provided here, with a detailed account provided later in the Framework Model (Chapter 4).

At the most basic level, a design project has three phases suggesting a beginning, middle and end. These are described in a variety of ways and are often expanded on by others [29, 19, 44]. In this work, the three phases will be referred to as project initiation, project iteration and project conclusion, as illustrated in Figure 3.1. This framework identifies major milestones between each phases with go/no-go gates, with additional milestones guiding the progress of the project within the project iteration phase. The primary purpose of these milestones is to regularly evaluate how likely the success and continued value proposition of the project is in the face of a variety of risks and uncertainties.

The project initiation phase focuses on planning, research and knowledge input. This allows for the evaluation of project feasibility and suitability and some assessment of risk, considering both technical aspects and organizational compatibility. Planning activities create a foundation of understanding and an initial set of artifacts that are carried through the remainder of the project. A significant amount of stakeholder involvement and communication, especially with the customer or client of the project, is key in obtaining the information required. Project initiation transitions to the next phase with a go/no-go decision based on available information, after which the project either continues or is terminated with minimal cost.

The next phase is project iteration, which is characterized by continuous progression through iterative techniques in design and management to facilitate rapid feedback and adaptation. Throughout this phase, activities support both the design process and project management, and collaboration with stakeholders continues to provide status updates and obtain feedback that helps guide the

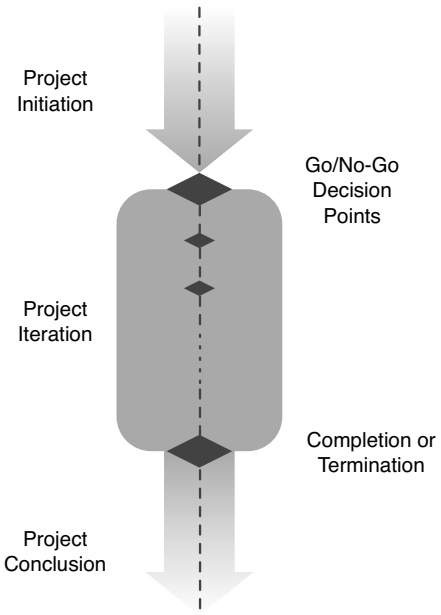


Figure 3.1: Three phases of a design project.

project. Regular go/no-go decision points allow a team to determine if the project is on track to provide continued value. If it is not, it should be terminated. The iteration phase continues until completion, reached when the scope and goals of the project outlined during project initiation have been met according to established metrics. In either case of termination or completion, the project moves into the final phase. Lastly, the team should review the process used within the project and the phase and adapt it to achieve the greatest value and performance. "Live" artifacts created during project initiation are carried through this phase to aid in managing and tracking knowledge generated and project progress.

The project conclusion phase centers on knowledge transfer and retention, and an outer feedback loop on the whole design project cycle. This phase should also strive to create an environment for continued measurement and feedback as the system moves through other stages of the product lifecycle. The activities and methods in this phase will depend heavily on project characteristics, the next stages of the project and the anticipated involvement of the current team and organization.

This is discussed in detail in Section 4.3. Should the project conclusion phase be reached through termination, the focus should be on meeting any commitments outlined during project initiation and on understanding the reasons for failure (as part of the outer design project cycle feedback loop).

The design project cycle is most often present in the premarket phase of the product lifecycle, and this framework is potentially best suited only to a subset, like the technical research and development phase. However, with products or projects that focus on continuous improvement (usually easier to change or more agile products like software, or those with a significant software component), the cycle may be stretched out readily into production or other phases. As suggested previously, multiple applications of the design project framework may be possible within the lifecycle as well. The design project cycle fits into the overall product lifecycle, and the boundary conditions (i.e. the inputs and outputs, reflecting the scope of the design project) should be considered with respect to the overall lifecycle of the product or system.

3.2 Iteration in the Design Project

Iteration in a design project is applicable both to project management activities as well as the design process. The feedback and review cycles in collaboration with a client and within the design team (retrospectively) are also both important. It is also beneficial to consider multiple levels or tiers of iteration within or across categories that may help organize and manage the project. The style of iteration applied at each point in a project may be of a different style. Existing methodologies exhibit a variety of different iteration characteristics, summarized as a single or zero iteration, time-boxed iterations, or a continuous or pull style of iteration.

3.2.1 Project Management Iteration

Iteration in project management is concerned with task planning, distribution, and deadlines. The order in which tasks, especially design tasks, should be handled is important, and is aided by the planning cycles of task selection and prioritization. These activities should consider overall project vision and goals, the critical path of the project, and managing project risk. Fully predictive methodologies like waterfall try to complete all of these activities once at the beginning of the project, while time-boxed approaches typically do so at regular intervals, with flow- or pull-based Kanban methods pushing for asynchronous cycles linked to task completions. Iterative planning cycles should be tiered, with high level goals planned and reviewed less frequently. Higher level planning is also done in less detail, as there is greater uncertainty. While detailed long term planning may be difficult due to this uncertainty, the high level planning can aid preliminary scope, schedule and cost estimates. As suggested by agile principles, planning provides guidance for the project, and plans should be able to change as a project progresses to reflect new information and better understanding. The exception to this would be when very strict outside requirements must be met, though introducing too many such constraints to a project is discouraged. A key part of understanding when an iteration is complete is determining how to evaluate if a task is complete through a definition of done or the acceptance criteria for a task.

3.2.2 Design Process Iteration

Iteration plays an important role in the design process. As discussed in Section 2.4.1, Safoutin [80] outlines two main types of design iteration: incremental and progressive, with additional discussion on the importance of feedback in iteration. Both types are top-down design approaches, with incremental being depth first and progressive being breadth first. Regardless of the form in which iteration presents itself in the design process, what is of particular interest is what the iteration supports: the evolution and improvement of the design throughout the design process and lifecycle of the product. The design is constantly being adapted and refined to fit the requirements of the

project. Iteration is a process where the design team can obtain feedback about a design and reincorporate it. This feedback is used to advance the design.

Design iteration may also benefit from a tiered approach. Tiers may be divided by feature, by subsystem and by product version, for example. Different tiers may be synchronized with different management activities or cycles, and reflect breaking the design tasks or system down into units appropriate for review or demonstration by the team, client, customers, or market. The iteration or evolution of the design will occur on different levels separated by feature, subsystem, domain or discipline, design phase, or total design revision or versions.

Scrum demonstrates an incremental focus to design iteration, where features represent vertical slices within a project. Each time-box sees features taken through all design phases to provide a working feature to the customer at the end of the time-box. Progression and feedback are implicit in the emphasis on rework of existing components as the design evolves. Designs with highly integrated subsystems or coupled features may be challenging to approach this way as effort spent in rework may become significant.

The waterfall design approach more closely resembles the progressive iteration as described by Safoutin [80]. The entire design is developed at increasing levels of detail through the design process phases. While this may seem beneficial, the ability to iterate and improve can be limited as there is little opportunity for high fidelity feedback obtained by bringing parts of the system to near completion early. On projects that have highly integrated (i.e. optimized) designs, this approach may provide greater benefit, especially if prototyping is costly.

Creating and testing prototypes introduces a more concrete, direct, first-order feedback capability into the iteration of the design. The prototyping approach is heavily linked to the design iteration approach, both in type and in frequency. The range of strategies available to create prototypes is presented later in Section 3.8. For prototyping to be effective, prototypes must be reviewed (for example, through use by the client or end user) and feedback sought for inclusion in the design.

3.2.3 Other Iteration

Tiered iteration rates in project management and design process improve the ability to synchronize between them and also with other distinct activities that are repeated regularly.

Customer interaction underpins agile approaches to development, and this must be accomplished regularly to keep a client informed and obtain feedback on value the team is creating. Activities that benefit from or require customer interaction include demonstrations, reviews and approval points (go/no-go decisions). Demonstration or delivery meetings release completed features or deliverables that provide value to the customer or users. This working product provides excellent opportunity for first-order feedback. In a Scrum methodology, this is tied to iteration time-boxes and task distribution and completion. Review and approval meetings may not see the release of a "working product" but provide useful second-order feedback on the progress of the project and the value it is going to deliver. The feedback from and collaboration with the client can help to make joint go/no-go decisions that determine whether the project will continue. The feedback discussion can also help guide the planning and prioritization of future work. If possible, customer interaction and iteration should be as close to instant and constant as possible through direct integration with client (for example, embedding a customer representative with appropriate domain knowledge within the team, like a product owner in Scrum). The realities of synchronizing different teams and organizations may mean that these iterative activities will be accomplished more effectively through time-boxing.

While the design process should integrate internal design review and feedback opportunities, technical reviews may require the presence of other technical experts or stakeholders outside of the core team. This is another example of where time-boxed or scheduled collaborative review meetings are helpful.

Retrospective activities allow a team to review how effective they are at delivering value and how well different elements of their methods and process are contributing to value delivery. This third-order feedback should also be explicitly included in the method, and carried out regularly. Examples

include retrospective meetings for the core team at the end of each iteration. Performance metrics of different types (especially visual ones) may help a team better understand their challenges or bottlenecks so they may move to address them.

Finally, other activities that are necessary for smooth project progression but do not directly impart value may fall into an iterative pattern. An example of this is regular payment for work completed when the client and the design team are part of different organizations. This should be detailed clearly in a contract early in the project. It may be linked to other iterative activities like go/no-go decisions and reviews. There is a range of options for contract structure and payment schedules, based around the flexibility or rigidity given to the scope, schedule or budget of a project [7, 10, 44, 54, 86]. It is recommended to choose a structure that encourages both parties to collaborate to deliver value.

3.2.4 Iteration Capacity in Different Disciplines or Features

Design work for each discipline will see differences in the ability to iterate and evolve. The capacity for iteration will also vary between features. This is especially the case when creating and iterating on prototypes. Delivering a "working product" requires different activities depending on whether it involves mechanical, electrical/electronics, software/firmware or other disciplines. Physical products must consider procurement, assembly, logistics and materials cost, while software often has no cost beyond development time. This makes faster iteration possible and continuous improvement easier. This is not only limited to different disciplines, but also different levels or "shearing layers" within the same discipline or across a single design. The shearing layer model can apply across disciplines as well as subsystems or features, but some disciplines tend to correlate more with inherently change-resistant layers. Figure 3.2 provides a basic example of this. For example, modifying an enclosure for a connected product that relies on an injection molding manufacturing process is typically more challenging to change than server side software acting as a backend for the product. At the same time, changing the database type in the server side backend software takes more effort than modifying an encapsulated data analysis algorithm. Improving change capacity is possible

with agile design, as discussed later in Section 3.6. A challenge with variation in the iteration capability is when a team must synchronize the readiness of different parts of a system or design for integration and testing (for example when developing full prototypes that incorporate multiple subsystems or features). This should be considered when structuring and potentially tiering design iteration during higher level planning activities. Furthermore, the resistance to change in the layer of the design impacts the task effort and should guide the amount of planning and architecture or concept design that will be completed.

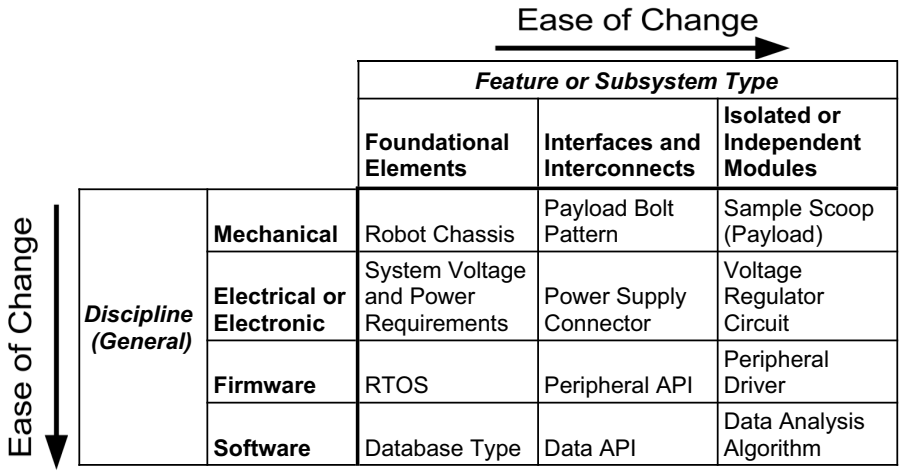


Figure 3.2: Shearing layers and relative ease of change in a mechatronics system, with example features or subsystems. Note that the examples may not be correlated across the vertical (e.g. a mechanical scoop payload may be easier to change than the power supply circuit).

3.2.5 Framework Recommendations

For the project management or other iteration categories, this framework encourages adopting a continuous pull approach to iteration similar to Kanban at lower (tighter) iteration levels. At higher iteration tiers or when it is necessary to synchronize with outside stakeholders or hard schedule requirements, time-boxed or time-fixed iterations or deadlines are desirable. For design iteration, the approach taken should balance between progressive and iterative styles based on project characteristics pertaining to specific technical and logistical challenges and the capital costs

of the system or product prototypes.

3.3 Stakeholders

Project stakeholders include any individuals, groups or organizations that may be involved in or impacted by the project. A list of potential stakeholders is presented below:

- Direct
  - Internal
    - Core project design team
    - Other supporting groups (e.g. Test, Quality Assurance)
    - Programme group, upper management, or organization as a whole
    - Marketing and sales
    - In-house service, support, and maintenance
    - In-house manufacturing
  - External
    - Vendors
    - Manufacturing
    - Shipping and distribution channels (e.g. wholesalers, resellers)
    - Service, support, and maintenance
  - Users/Customers
    - Users
    - Administrators
    - Organization
    - Parties involved in handling end of life (recycling, disposal)
- Indirect
  - Environment
  - Public

Traditionally, stakeholders are those outside of the core project team though this core team is easily considered part of the overall stakeholder group. Indirect stakeholders rarely obtain direct benefit from a project or product (or they would likely fall under the Users/Customers category), but may experience negative consequences if not included in the process.

Design decisions may impact some stakeholders only in later stages of a project or product lifecycle, and these stakeholders should be considered early. For example, a product may or may not be designed to undergo maintenance during its lifetime. If maintenance will be performed, it may be done by a variety of potential stakeholders [30]. One option is to only allow maintenance to be performed by the original manufacturer, to more tightly control outcome. Independent trained maintenance personnel often provide an alternative when system complexity is high. Finally, individuals with no specialized training may service their own system. Often, the target maintainer will vary for different features or actions, depending on complexity and frequency. The requirements for different maintainers will likely be very different. An example is the potentially conflicting goals of rapid and easy assembly during manufacturing, where fasteners may be minimized and adhesives substituted, with the ability of a consumer to disassemble and replace a component that deteriorates rapidly. Indirect or peripheral stakeholders like the public or environment are also good examples of those who may not be impacted until later. It is particularly true when considering the end of life and disposal or decommissioning and the difficulty or residual waste associated with this.

Stakeholder prioritization can not always be correlated directly to immediate profit. While financial success is important to any business, ethical considerations must be made when identifying and engaging stakeholders. There will almost certainly be benefit from doing so, though it may not be tangible or readily measured financially. These benefits may be related to trust, reputation and goodwill.

A situation where this is relevant is in the design of surgical instruments, as described by Dr. White [97]. The surgeon is the user, and the hospital is the purchaser of the instrument. These primary stakeholders will drive requirements and marketing. However, it is noted that the patient and their outcome is or should be the primary concern of the doctor and hospital, and thus designers of

the instrument should identify and prioritize patients as stakeholders. Working towards improving patient outcomes more effectively will provide clear benefit to other more obvious stakeholders as a result.

The goal of any project should be to provide appropriate and sufficient value to all stakeholders, while minimizing any adverse impacts. Understanding who comprises a project's stakeholders and how to deliver value to each helps make the project successful, and subsequently the organization as well. From an ethical standpoint, projects should deliver value to stakeholders, not just deliver profit to shareholders.

3.4 Team Organization

A mechatronics design team must have the appropriate skills, knowledge, and culture to match the project. Agile principles encourage a cross-functional team with a focus on generalists (as in Scrum), though specialists provide value as well (as suggested by Kanban and Scrumban). A mechatronics project will cover multiple disciplines, and it is unreasonable to expect all members of a team to be equally competent across all disciplines as would be expected in a team composed purely of generalists. However, this framework encourages knowledge transfer amongst a team and learning how to contribute to the project outside of an individual's comfort zone or specialty. Such collaboration and teamwork is important for the efficient distribution of resources and in managing bottlenecks. A variety of software tools can support collaboration and teamwork. Some examples can be found in Table A.2 with the tag 'Collab'. The size of a team may vary, though smaller teams are typically able to achieve better agility due to efficient and high bandwidth communication opportunities. Other agile methodologies suggest team sizes should be limited to around ten members [44], but of course this should be adapted to the specific team, organization and project. As a team changes size, reorganization may be necessary to maintain effectiveness. If possible, a team should be dedicated to a single project and remain as a single cohesive unit for the duration of the project. Team members changing focus to different projects introduces a context switching

cost, impacting overall productivity. The same principle applies on an individual task level within a single project, where multitasking expectations should be minimized to improve productivity and reduce distraction [73]. In addition, adding new team members mid-project without sufficient care may decrease progress [16].

3.4.1 Division of Work

Key methods for dividing project work and tasks amongst team members are by domain or discipline, or dividing by feature or subsystem. Furthermore, a team may be benefit from being split into sub-teams. Organizing a mechatronics team and task assignment is illustrated in Figure 3.3.

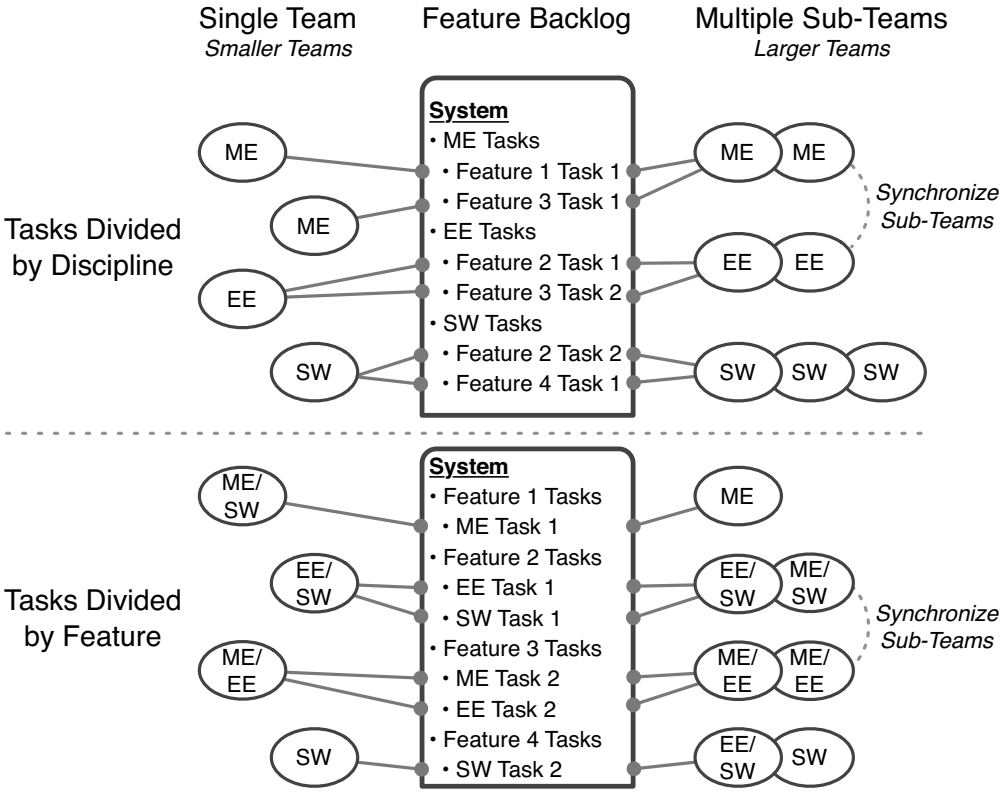


Figure 3.3: Organization and task distribution alternatives for a mechatronics design project team.

The foundation of team organization rests on how tasks are distributed to members of the team.

In this model, tasks are queued in a product backlog or task list similar to those found in Scrum or Kanban methodologies. Team members are matched with tasks based on skill and task priority. Team size will dictate whether sub-teams are necessary, and sub-teams may utilize a smaller sub-backlog created as an additional part of planning. Note that overall the sub-teams should act as a single cohesive team when viewed by outside stakeholders, even with the additional organization. The four basic organizational options may be described as follows:

Single Team, Tasks by Discipline Do not divide team and have each member work only on tasks related to their core competency. The team should be small enough that communication remains easy and efficient amongst the team members. Review and planning tasks should incorporate the whole team, and some synchronization is needed when integrating and delivering large feature or subsystem units. Tasks in the backlog should be broken down by discipline. Team members then draw only tasks from the backlog that fit their discipline or specialized skills. For example mechanical engineers will handle 3D modeling in CAD, while electrical or electronics engineers will be responsible for circuit design, and software developers will work on firmware drivers or algorithms.

Functional Sub-teams, Tasks by Discipline Divide into functional sub-teams that work primarily on tasks related to their discipline. This means all of the mechanical engineers will be on a mechanical sub-team, electrical engineers on an electrical sub-team, and so on. Each will have a backlog composed on discipline specific tasks, as described previously. This is a more traditional approach to project management and team organization and is perhaps easier to adopt when transitioning from a more traditional development methodology. This approach is suggested by Rothman to help manage the inherent development iteration cycle rates between different domains [77, 78, 79]. Reynisdóttir provides an example of this organization, where a mechanical and electronics team must work together, each following the Scrum methodology [74]. Note though that these two teams are more independent than is suggested by this framework.

Single Team, Tasks by Feature Do not divide the team, and expect team members to perform cross-functional generalist tasks. This approach is likely to work will with a strong pull-based iteration strategy, like in Kanban, where individuals may pull tasks appropriate for their specialization more often, but also act to support other team members when their work is forming a bottleneck and slowing progress. It may also work well when a time-boxed iteration strategy is used and the team is in fact very generalist. This is the approach suggested by Maccherone [53]. This method of team organization is probably most effective at improving knowledge transfer and education within the team. It reflects agile principles well, but may not work well as a team grows (which is handled by the next method of team organization). The balance between specialized and generalized activities can range, from being entirely generalist in nature to having team members be minimally cross-functional outside of their core competency. For example, a software developer may work on software and sometimes embedded firmware, while an electrical engineer may focus on circuit design and contribute slightly to firmware development.

Feature Sub-teams, Tasks by Feature Divide into feature sub-teams that are small and cross-functional, and built to have appropriate knowledge and skill for the feature(s) or subsystem(s) assigned. This approach follows agile principles more than dividing teams by function. It is especially appropriate when the subsystems or features of each are relatively modular and decoupled. It also handles a larger overall team size that may be challenged to work together as a single unit as in the previous option. This approach to team organization is suggested by Myllerup [59]. It can potentially help deal with some differences in iteration cycle rate between different features or "shearing layers". It is noted that the features or subsystems will often have some range of focus on different disciplines as well. For example, a chassis subsystem is primarily mechanical while a web application human interface has no direct mechanical work. This will skew the cross-functional composition of different teams, but the underlying agile mindset is still present. Furthermore, some skills will still be cross-functional and applicable. For example, perhaps a mechanical engineer has some insight into human interface design applied to the control of a mechatronics device even if the technical work is primarily in software development.

Recommendation This framework encourages dividing tasks by feature or subsystem over discipline to improve collaboration and knowledge transfer. The decision to use multiple sub-teams will be based on the size of the overall team.

3.4.2 Challenges with Specialization

Implementing a kanban style system may result in multiple task types (tasks for different design phases) within a team's area of responsibility. For example, there will be tasks related to requirements, architecture, detailed design, testing, etc. There may be team members that specialize in and favour one of these types of task over others, which can introduce bottlenecks. Team members should be encouraged to generalize more, and the team should support them through collaboration and knowledge transfer. Furthermore, creating teams composed of such specialized skill sets should be avoided to reduce task queue mismatches. The team should not have a large number of members specialized in both domain or discipline as well as design phase skills.

3.4.3 Synchronizing with Others

If a team is split into sub-teams, depends on outside stakeholders or is part of a larger group (i.e. there are information inputs and outputs), then synchronization between the activity and iterations of the team is necessary. Parallel activities like integration and testing between sub-teams, meeting external schedules, or demonstrating to clients or other stakeholders that are minimally involved are examples of where synchronization must be handled. Most of these activities occur at higher tiers in the iteration. Key factors to consider when planning how to synchronize include whether there are sub-teams and how the design iteration is split (by feature, domain, etc.). An appropriate way to improve synchronization is with an overall plan at the resolution of the higher iteration tier. This may be implemented as a roadmap outlining intermediate demonstration and delivery milestones as suggested by Rothman [77, 78, 79], or with high level product backlogs consisting of large or general tasks that should be reviewed and pulled from at a lower rate (appropriate for the

time resolution of the higher iteration tier). A roadmap is a recommended planning artifact for this framework and methodology, and described further in Section 4.2.5. Such an approach does apply additional constraints to the lower level iterations, but it should be possible to match the strategy to have minimal negative impact. This is further mitigated by encouraging adaptability in the planning artifacts. The higher level planning should include input from all teams or team members as having a team lead and higher level management handle these activities falls back to a more traditional hierarchical structure (though this is still an option if it works well for the team). The goal is to avoid impeding the teams completing the work and ensure that communication, collaboration and synchronization between sub-teams or outside teams is clear. The higher level planning must also follow the overall vision of the project. Note that synchronization may not require precisely matching the schedules of sub-team tasks, unless these tasks are critical for the whole team to move forward (for example, shifting to full system integrated testing). Effective synchronization looks to avoid bottlenecks, ensure effective flow of information and help to organize the team or sub-teams with others.

3.4.4 Development of Skills and Knowledge

Successful teams should strive to continuously learn about design processes, project management and technical aspects of design. Theory, methods, and techniques relevant to design projects may be learned from a variety of sources.

Academics The first exposure to understanding design projects is through academia. A typical engineering or computer science curriculum will include some components of design and project management. Unless a student is enrolled in a design or management specialization, the experience is likely limited. Accredited engineering schools in Canada are only required to have design as 12 percent of an engineering program [12]. It is noted that extracurricular activities may provide additional opportunities for growth.

Industry On the job training and experience significantly supplement the brief introduction to design process and project management obtained during postsecondary education. While ad hoc learning through experience working on design projects will inevitably be a primary source, there may also be more formal in-house training programs. Many organizations develop their own methods and strategies that are customized to their specific problem domains and business activities.

Professional Development Outside of an employer, individuals may pursue additional training opportunities targeted at their professional development. Professional organizations like APEGA or industry organizations like the Institute of Electrical and Electronics Engineers (IEEE) are two sources of education. These types of organizations help facilitate consistency and improvement across professions and industries and help transfer knowledge between organizations.

Most education and training is job- and industry-specific. The focus is on either technical or management topics, and is usually discipline specific. There is limited cross-discipline and cross-functional training from a holistic standpoint, though this is understandable, as industry- and domain-specific training is usually sufficient and appropriate for most jobs. Teams are traditionally composed of individuals responsible for specific parts or aspects of a project, each with experience to complete their part. Such teams need to be (and usually are) large enough to have a sufficient collective skill set and experience to meet all project needs. However, working in research, consulting, product development or at a "start-up" may all benefit from a generalist approach, where smaller cross-functional teams are necessary to achieve success across a wide range of tasks. In these cases, continued learning and skill development is even more important.

3.5 Project Characteristics

It is necessary to understand the characteristics of the project when deciding on a methodology and how to adapt it. This framework is particularly applicable to projects that benefit from agility and rapid feedback. Project characteristics will impact how "heavy" or formal the method should be,

or the level of complexity and detail in the process, elements, activities and artifacts. In general, a lighter approach will improve agility as a project evolves. Different characteristics are more important to the acceptability of a method for a project. The weight of characteristics will also vary between projects. For example, cultural characteristics are likely to be more polarizing in whether an agile approach is possible, while a methodology may be more easily adapted to address other characteristics. It is important to evaluate characteristics of the project itself, the project domain, the team, the organization, and the client. The underlying themes of this activity is to understand project feasibility and areas of risk.

3.5.1 Exploration Versus Optimization

The first step in evaluating the suitability of a project for an agile methodology is to understand how exploratory a project is. Agility and adaptability are an excellent way to mitigate risks and challenges associated with uncertainty and a lack of information in the project. On the opposite end of the scale to exploration, a project may focus on optimization. Such a project will typically have a low level of uncertainty and the team will have a clear understanding of requirements, goals and solution approach. Here the advantages of agility and adaptability are diminished. There are a number of questions to consider when evaluating the exploratory nature of a project. Has the team or organization completed a similar project in the past? Has a third party completed a similar project? Is there an existing market? If it has been done in the past, this should increase confidence that the project is feasible, or at least provide a benchmark as to the level of competency required. If a similar project has been demonstrated inside or outside of the organization in the past, there may be an existing process that has been demonstrated to work. Following an existing process will certainly reduce the exploratory nature of the project. A good example is the use of standards for designing pressure vessels. A design project focused on optimization will take an existing proven design and rework it. There already exists a pattern or template that can be adapted to requirements that are important, like reducing costs, optimizing for mass manufacture, or improving design efficiency. Should the team have limited experience in the design project at

hand, iteration, experimentation and feedback will help improve the design. Projects with areas of high uncertainty or risk (especially technical) will benefit from a greater focus on risk management throughout. The design should be approached in a way that reduces this risk as early as possible. A project that is exploratory as suggested by the level of risk and uncertainty will benefit from an agile methodology.

3.5.2 Criticality

Criticality in a project is tightly linked to risk management. As the consequences of failure in the design project increase, so too does the criticality. The consequences of failure may extend beyond technical risk to other areas of the project (for example, the consequence of missing a deadline). Typically, a project with greater criticality benefits from more formality. This may be accomplished with an increase in volume and detail in documentation and other deliverables. Increasing the artifact load will reduce agility to some extent as design changes require greater effort to propagate. Criticality and the demand for greater formality may come from external pressures like standards, regulations or industry guidelines. Such external influences must be considered early in the design project to ensure any requirements that arise from them are included. The type and degree of criticality of a project may impact how different activities are carried out. For example, a project where technical failure may result in loss of life, the type and method of prototyping should be carefully considered. Another example of criticality is in the ability for a design modification after release. Products or systems that are difficult to change after release require greater effort in reducing defects. A project with high criticality does not discount an agile methodology like this framework from being used, but it may need to be adapted to increase the formality or include extra deliverables in the process, especially with an emphasis on test and documentation.

3.5.3 Team, Organization and Client Cultures

The team working on a design project must be willing to embrace an agile approach. Each team member should be motivated to contribute value. Team members must be able to collaborate and communicate well, and not simply work on the same project in an independent manner. A team unwilling to approach the project with agile principles in mind will not be successful with this framework or with other agile methodologies.

Beyond the core development team, others in an organization should support an agile approach to development. This is especially true for management, who should support the project team and help remove impediments. Two important areas of consideration are planning and feedback. Management must be willing to put aside fixed schedules, budgets and estimates in a traditional sense to allow the plan to guide the project and evolve with it. Forcing the team to meet hard goals instead of pursuing value will negate the application of agile principles. The organization outside of the project team should also be willing to collaborate with the team and provide feedback, especially in areas where the project team is interacting with other parts of the organization (for example, with manufacturing, procurement, sales, or marketing departments). Introspection will be an important part of the team and organization evaluating how easily an agile methodology will fit.

Client culture is important on a project basis especially if clients will vary between projects. If the team is only working on projects internal to the organization, then client culture is less relevant. However, in larger organizations, the client team may be sufficiently removed that there are culture differences within the organization and the client team should be treated similarly to an external client. A client must be willing to be more involved in the project than would be typical in traditional approaches. They should be committed to collaborating and communicating regularly to provide feedback, answer questions, and address issues early. Ideally, a client representative will be embedded in or dedicated to the project team, and available all of the time. If this is not possible, at least there needs to be a rapid response time between the client and project team. Without this, feedback and progress will slow down. Sufficient domain knowledge on the part of the client and

the team is important for making design decisions and tradeoffs. Also, an understanding of how different aspects of a project or a design provide value to the customer is important, and how this changes and evolves. As more information is obtained throughout a project, there may be a shift in what the client deems most valuable. Communication between client and project team benefits from face-to-face or similar high bandwidth methods. A client should be present for activities of more formal feedback, like demonstrations, and acceptance and approval meetings. While this is present in even more traditional project methodologies, it is likely the frequency will be higher in an agile approach. The client, like an organization's management team, must be willing to work within a less traditional environment of budget, scheduling, contract. Alternative contract and billing structures will provide greater support to an agile approach and encourage both parties to strive for the delivery of value [7, 10, 44, 54, 86]. Should a customer demand a fixed price estimate, implementing agile effectively will prove more difficult in all parts of the project. It is still possible to apply agile principles and values, but the project will require greater upfront planning, making this framework much less appropriate.

The implications of client and organization culture are also applicable to other stakeholders, such as manufacturers and suppliers, but typically to a lesser extent. As the amount of information transfer between parties increases, so does the importance of a compatible culture.

3.5.4 Problem Domain and Market

The problem domain and market of the design will impact how effective this methodology will be. Key areas of interest are the business model applied to the project, external requirements of the market, risks in the market, and cultural values of industries within a market. Various business models that may be applied to mechatronics products or systems are described above in Section 3.1.1. Service-oriented or other models that are compatible with design change throughout a lifecycle can benefit from agility in the product, as do products or services where early release for feedback is feasible. These models and lifecycles may even have intended periodic iteration in the design of the product. This is explained further in Section 3.6. Different markets or industries may

necessitate specific requirements and adherence to standards or certifications, either by regulation and law or through convention and industry acceptance. This is closely related to the criticality of the project, as outline above in Section 3.5.2. Finally, risks (like competitors, customer acceptance and adoption, etc.) of varying degrees are present in different markets. If the market and problem domain area experience high change and uncertainty, and the risks are high, an agile methodology like this framework will be very beneficial.

3.5.5 Technical Risk

The level of technical risk in a project is tightly related to how exploratory a project is. Of all risk types present in a design project, technical risk is most directly managed through design decisions and development activities. Areas of technical risk should be identified and addressed early in the project to reduce uncertainty and potential impacts on schedule or budget. This can be done by prioritizing tasks that progress the development of technically challenging design elements. Other techniques include breaking down features or subsystems into smaller steps or units to improve the task resolution, or to increase the effort spent on requirements, analysis, functional modeling and architecture activities. As an example, Scrum suggests carrying out exploratory "spikes" to improve understanding and reduce risk, often using prototypes for direct feedback. Rapid iteration and feedback in areas deemed to be high risk can help quickly identify which of these areas need more attention. This will help the team refine estimates of effort or change feature priorities to avoid those that will impede progress. Finally, increasing the formality of relevant design activities can improve the breadth and depth of information about project elements and help mitigate technical risk.

3.5.6 Cost of Prototyping and Testing

Prototypes and testing help provide feedback about the design. It is important to understand the cost of prototyping and testing early in a project, particularly for cross-disciplinary projects and

when each project has different domains and requirements. The cost may refer to capital costs and time, both in the sense of team effort which impacts both cost and schedule, and in delays from procurement, shipping or other external dependencies which only impact schedule. The cost will govern how rapidly a team can iterate on full prototypes or tests. An increased cost or time to prototype will reduce the agility that is possible in the project. Solutions to address these issues (discussed in Section 3.6) can allow a project to shift back towards greater agility. It is noted that an agile approach focused on iteration can still be applied to other parts of the project, like project management. An increase in cost or time of prototype development increases the consequence of the prototype not working properly. Managing the risk in prototype development and testing will require greater effort to ensure these activities will provide value for the project.

3.6 Agile Mechatronics Design Principles

It is possible to achieve agility during the development process, and it is also possible to achieve agility in the design of the product [42]. In most cases, both provide benefit. If the product is targeted at a stable market with a business model that does not allow change after release, or the industry culture restricts change after release, then the benefits of agility in a product are minimized. Even in these cases, agility in the design is helpful during a premarket stage in prototypes. Reasons that make product or system agility desireable include changing market conditions, changing requirements (customer, client, external), and changes in underlying technology that a product relies upon. Business models and lifecycles with intended periodic iteration in the design of the product provide good opportunity to leverage an agile design principles. Those with no opportunity to change the design after release (form of higher criticality) may require different strategies. Agile design principles for mechatronics projects are value driven and try to create rapid and useful feedback as a method of measuring this value. To maintain agility, a team should avoid locking into specific paths as much as possible to rapidly adjust a design based on feedback to improve value. Most agile principles and values outlined for software development are applicable across the domains and situations of mechatronics, and should be part of the team's mindset during development. This

framework outlines seven key design principles including simplicity, modularity, feedback by design, customer engagement, feedback through prototyping, keeping documentation relevant, and disciplined development. Most are tightly linked and described in further detail below.

3.6.1 Simplicity

The design team should strive to keep the design simple and indicative of design intent. This will make maintaining, improving and adapting the design easier, both in the current design project and in other parts of the lifecycle. A clever design that improves the simplicity or understanding of the design adds value, but clever tricks that reduce clarity should be discouraged. For example, creating a code "oneliner" does not add value if it takes more effort to understand the behaviour. The design can be kept simple by minimizing the feature set at the beginning of the project and making it easy to add or improve as it evolves. Adaptive design should be preferred to anticipatory design, though a balance is needed. A sound technical foundation for the design must be created for this approach to be effective. The principle of simplicity is especially valued in agile methodologies [24, 44]. Simplicity of design is also encouraged by Noergaard [60].

3.6.2 Modularity

Modularity can be applied to all areas of a design project, including the design itself, tools used, components, cross-project inventory, templates and processes. Modularity helps improve the agility and flexibility in the design [42, 68]. Increasing the modularity of a design can be achieved by improving the cohesiveness within subsystems or features (keeping similar or supporting elements together), and decoupling between subsystems (reducing dependencies and links between elements), as suggested by [69]. Modularity applies to software and to hardware. It is noted that increasing the modularity of a design comes at a cost of a less optimized design, but for exploratory projects, this is acceptable. Once a project is more certain and mature, then the lifecycle can shift to a phase of optimization. An optimization stage can still follow an agile management process, but will ben-

efit from an integrated progressive iteration strategy in total system design. Examples of modular designs include a software module or library with a clear application programming interface (API) and dependencies, electrical subcircuits like power supplies with clearly specified interfaces, or a mechanical design like a skeleton chassis to which other components may be readily mounted. It is important to keep functionality constrained to where it makes the most sense (cohesive) and avoid spreading it between different features or subsystems. A different modularization strategy involves standardizing interfaces and clearly defining functionality so that improvements or substitutions can be made in the isolated part of a system to make it applicable to different situations. Examples of this include a common connector, signal and mechanical mount pattern to make robotic payloads interchangeable, a power supply section that provides the same output of power, voltage and current but can handle alternative input specifications, or software APIs that make it possible to switch between different wireless technologies or even a wired solution (Bluetooth, WiFi, 802.15.4, USB, serial, etc.). This sort of modular thinking is demonstrated by the 7 layer Open Systems Interconnection (OSI) model [48].

The concept of modularity can be extended to include reusability [69]. There are several possibilities including reusing design elements in a different project, using commercial off the shelf (COTS) components, and reusing techniques or templates across projects. As a team and organization works through different projects, they may create a library of modules for use in future projects. This builds value by reducing future technical risk with working elements and accelerating development of new projects. The same benefits apply to using COTS components. Development time can be reduced by picking standards components that are readily available. Even if these components are not optimal at the beginning, the team can always iterate later and reimplement in-house to optimize the design. Some examples of reusable or COTS elements are listed above:

- Software libraries with consistent APIs
- Electronics blocks (schematic and maybe also printed circuit board (PCB) layout) with clear functionality and interfaces for microcontrollers, power supplies, analog frontends, communications, etc.

- Common mechanical interfaces and connections, including consistent fastener selection and patterns
- Payload or module interface standardization (mechanical, electrical, power, connectors, communications, and ideally software API or similar). Examples like USB are complex but very adaptable.
- Reducing number of unique parts (discrete or passive components in electronics and type and sizing of fasteners in mechanical)

The processes and procedures used in one project should be reusable and adaptable to other projects. The breadth of areas is large, but examples include custom tools for assembly or test, documentation templates, rapid prototyping methods, PCB design and bring-up guidelines, design review methods, or simulation approaches.

The examples of reusable elements, processes and procedures above are tightly coupled with approaches to prototyping, which is discussed further below in Section 3.8.

One of the underlying goals of modular design is to make change easy [68]. This improves flexibility and the ability to evolve and adapt the design rapidly. The team should be able to readily modify or adapt parts of a design to meet changing requirements. This means it should be possible to isolate parts that need improvement. Standard or reusable items also makes documentation or other secondary tasks easier, as does experience implementing a specific element in the past.

It is important to understand that increased modularity and design agility is more beneficial in layers, subsystems or features within the design that are expected to change more often, either as project requirements change or through the lifetime of the product or system. Potential for change in different parts of the design should be evaluated to focus effort appropriately and provide the greatest value.

Implementing a modular design is not without challenges. Maintaining modularity can be difficult across the life of a product. This is especially true if the product or system is continuously evolving and improving. Modularizing the system to allow low impact changes to only parts of the system

indicates that the interfaces between modules should be defined and stable. This means improvements must be backwards compatible if isolated updates are to remain possible. Unfortunately, this can restrict improvement at these interfaces, which represent a layer that is more challenging to change. The balance of maintaining backwards compatibility with potentially greater design improvements must be evaluated as the design evolves. A basic example is the development of a stable API, and understanding when new features require that the API be changed, impacting any downstream users of that API.Modularity focuses on ease of change and piecewise improvement or modification at the expense of optimal design. If attempting to optimize a design for one or more constraints or requirements (for example, power consumption, size, weight, speed, cost, etc.) in each unit, a modularized design approach will make this difficult to achieve. Such a design project is likely to have less technical risk and more stable requirements, and thus receive less benefit from an agile approach. For example, optimizing a system for manufacturing will undoubtedly sacrifice some ease of change to achieve lower production costs. On the other hand, in some projects, there may be some parts or subsystems or layers that require optimization, while others will still benefit from the flexibility and ease of change that modularity affords. A carefully architected system may be able to accommodate both.

3.6.3 Feedback by Design

Just as feedback is needed from customers to understand if the design and management process are meeting needs and creating value, feedback is needed from the design itself to understand issues within the design and isolate problems as early as possible.To enable direct feedback about the performance of the design, planning and infrastructure for feedback and test should be created early. Features improving design observability should also be incorporated [19]. Monitoring and feedback is useful in all stages of a product's lifecycle [98, 60, 69]. This includes detailed test and debug obtained during development, gathering information during manufacture for process improvement, real time monitoring and feedback in deployed products, and tracking why or how products are retired. Different products and systems will have varying opportunities of interest for

feedback. One technique for feedback and testing includes Test Driven Development (TDD) [40], which is typically targeted at software but may be adapted to other disciplines as demonstrated by the WikiSpeed project [26]. Designing tests and understanding the definition of done and acceptance criteria for portions of the design should be carried out early and iteratively in development. Insight into the success and value of a design can be obtained with an analysis of data collected during monitoring. Design reviews provide good opportunities to review and discuss such data. Examples of design features that provide data for monitoring and feedback include setting up debug, logging, and error handling in embedded firmware and hardware, instrumenting parts of code, mechanical, or electrical components for measurements, logging data in volume during development and early testing, and creating the capacity to log and potentially report issues back to the team even once a product is released to customers. These design features are also related to improving the fault-tolerance of a design. In the event of a failure, the design should fail safely and provide a mechanism to understand the conditions leading to the failure. Using firmware and software as an example, fault tolerance can be improved with techniques like Design by Contract [56], proper application of asserts and enabling debug features appropriately.

3.6.4 Customer Engagement

Collaboration with the client is a core value of any agile methodology [24, 44]. It is a recurring theme in this framework, as part of both design and project management. Obtaining feedback from customers early for both of these areas is important, as discussed in Section 3.7. Providing the client or customer with a minimum viable product (MVP) as quickly as possible enables this feedback [76]. Maccherone [53] similarly refers to a "minimal marketable feature" (MMF). An MVP allows a customer to experience and understand a concrete embodiment of the design so they may give appropriate feedback. This also starts the test and feedback cycle early, even if it is initially less formal. Delivering a working product or an intermediate with tangible functionality (for at least a portion of the project) readily communicates the value being generated to a customer or client. Of course, the client must understand the goals and expectations of this rapid, early prototype

process and that results or artifacts may not be carried further in the design. Incorporating the client into the feedback process early helps to identify issues related to technical risk, feasibility, and the ability to meet requirements. It is important to make sure the design being developed through the team's understanding of the project vision is in line with customer or client understanding of the vision as early as possible, and adapt the design or the vision as necessary. This may include improving the client's understanding of what is feasible within project constraints. A similar process of collaboration should occur with other stakeholders (like manufacturers, sales or marketing departments, vendors or suppliers), when it is appropriate (i.e. as their feedback and capabilities have increasing potential to impact the design).

3.6.5 Feedback Through Prototyping

In support of providing both the development team and the customer with a concrete design embodiment through which to evaluate value, progress and fitness, the design process should strive to develop prototypes or MVPs quickly and often. This is discussed in detail in Section 3.7 and Section 3.8.

3.6.6 Keeping Documentation Relevant

Many design processes focus on the creation of documents and artifacts to capture the knowledge generated. It is important to consider the goal and purpose of documentation before expending effort creating it, and to further consider the best medium for each case or artifact [25]. Documentation has limitations within the overall goals of communication, collaboration and knowledge transfer, especially as a project is constantly progressing and evolving [44]. Documentation is discussed in detail in Section 3.9.

3.6.7 Disciplined Development

A project team should strive to maintain a disciplined approach to development and to "do it right the first time" within an agile context. The phrase "do it right the first time" is often associated with traditional design that views iteration to fix errors as a failure of process and is to be avoided [44, 80]. In the context of this framework, the phrase should be interpreted differently. Rather, this framework takes the stance that tasks should be carried out properly and with discipline, within the existing framework or internal standard of practice, as appropriate to the work at hand and the value it is creating [44, 19, 60]. It is important to do it right the first time but not necessary to do it all the first time. It is not expected that a design be completed correctly in one shot, but the design must be created with value-driven work, technical excellence, best practice, due diligence, and professional intent. It is important to avoid taking shortcuts that will be detrimental to key principles like simplicity and modularity. During development, the team should avoid the accumulation of technical debt (i.e. a backlog of things to improve or fix because it was not done well). If the team does see evidence of this, they must be sufficiently disciplined to spend the time to work against this regularly. It is not acceptable to skip steps or accept many mistakes, the team must still carry out checks, reviews, and inspections. Doing it right the first time may seem at odds with the concept of bare sufficiency, but is instead complementary. When planning and executing tasks, the team must consider the value generated relative to the effort put in. In some cases, a quick hack or shortcut provides rapid feedback or answers to questions to continue making progress. The goal of such activity must be understood, as well as the limitations of the result. Consider the activity of creating a low feature or low fidelity mockup prototype. If it will provide value by helping to further specify the project, demonstrate an idea to the client or prove a technical solution, then it is worthwhile, if the limitations are noted. What is dangerous is propagating these limitations to a final product where they are unacceptable (do not meet requirements or do not provide longer term value to client). Avoiding the propagation of limited intermediate results is important in all parts of the design. Other examples include using a back-of-the-napkin calculation for initial estimates but not using the same result in detailed analysis or simulation, or using a short bit of code to test

an idea but not using the same piece of code in production without adding appropriate error and corner case handling. The accuracy and precision must be matched appropriately between outputs and inputs. If a project team is trying to reduce low value work by streamlining process, reducing upfront planning and limiting documentation, they need to be disciplined to at least carry out the small tasks that are necessary to maintain the minimum level of sufficiency and ensure collaboration is effective (for example in software, follow a coding standard, maintaining up to date comments or automatic documentation support, and consistently use revision control in lieu of detailed external design documentation). Disciplined but lean development may be supported by adopting tools, procedures, and processes that target bare sufficiency.

Maintaining disciplined development can be challenging. Team members will be tempted to skip steps or avoid updating artifacts if it appears to be impeding progress. This can be resisted by making the right thing to do the easiest thing to do. Keeping undesirable tasks fast and easy will help the team follow accepted process. For example, implementing a revision control commit policy that requires a detailed commit message and provides a template will encourage the team to document code changes properly. It is necessary to regularly evaluate different practices and procedures through retrospection, and ensure they continue to provide value. They should be modified or removed if they do not.

3.7 Feedback and Knowledge Transfer

Design activities generate knowledge. This knowledge can be used to understand the design, improve the existing design, capture the state of the design, and obtain insight into the design activities themselves. The knowledge must be transferred and propagated within the project team as well as to other stakeholders to ensure sufficient understanding. This is readily accomplished through collaboration and communication. Fast, high bandwidth options like face-to-face meetings are preferred over slower, static approaches like documentation, as suggested by numerous resources on agile methodologies [24, 44], to increase the rate of feedback. Faster, higher frequency feedback is

of greater value than slow or delayed feedback as the design or process can converge to an effective solution more quickly. Documentation remains useful for long term archivability, but should be leveraged efficiently and in a way that provides value to the project. Prototyping provides a direct path to understanding the design with first-order feedback. A prototype as an artifact is not appropropriate for direct knowledge transfer. Prototypes embody knowledge but are potentially short lived or inaccessible, and therefore are not a reliable archive of this knowledge. However, the experience associated with a prototype, both through creating it and testing it, leads to excellent understanding and opportunity for design feedback. Furthermore, documenting the production and testing of a prototype creates an opportunity for a far more permanent knowledge archive that can reliably capture the state of the design and facilitate understanding throughout the team both now and in the future. Review and documentation of the design without an immediately working product can be thought of as second-order feedback. While design feedback aids in the improvement and evolution of a design, third-order feedback about the process of the project can help improve a team's approach to delivering value. Incorporating the client into the feedback cycle allows a team to prioritize activities that will create the most value. Leveraging quantifiable metrics for monitoring and feedback of both the design and the process will help a team identify areas for improvement, and understand how changes that address these areas are working in an objective manner. In summary, three types of feedback are helpful in a design project. First-order feedback leverages a working prototype or minimum viable product. Second-order feedback arises from the review of progress using intermediate deliverables like design documents, without a working product. Third-order feedback uses retrospective activities to understand the project process.

3.7.1 Process and Management Feedback

The project should regularly use retrospective meetings or other activities to obtain feedback on the design and management processes. Retrospectives should be conducted regularly as part of the review stage of the project design cycle. There should be an appropriate amount of feedback

for each level of iteration in the cycle. For example, as tasks are completed, there should be a brief discussion about how well team accomplished the task and if there were any challenges, while reaching high level milestones should be accompanied by a review of overall strategies used to reach that point. The retrospective activities may not need to be formal, but must be part of a culture of seeking to improve the ability to deliver value. A team should use the feedback to enhance their understanding of how they work together and within the framework of their methodology and project. Identifying areas that are challenging or restricting progress is the first step in making changes to the methodology to improve it. Each team and project is different and always changing, which means the methodology must be continuously adapted to suit, both through the course of a single project and over the longer term existence of the team. As described in Chapter 4, a kanban board as part of the project management process provides valuable feedback regularly. A team should consider other practices that allow data-oriented feedback to be collected and analysed more formally.

An important aspect in formalizing methods of feedback is in using design project process metrics. These metrics allow quantifiable analysis of the performance of the team and the method, which makes improvements measurable over time. A team needs to determine what metrics will provide feedback such that they can improve their value proposition. Collecting metrics must be easy, as with other practices, if the team is to do so with discipline. Building a collection of long term data can provide unique insight for future improvements to the overall methodology, even if short term data is less useful.

This framework suggests tracking the key metrics of time and defects. While critical if an organization is charging hourly rates, time tracking can also help a team understand project and task effort estimation or uncover bottlenecks in processes. It can also be used to provide more quantitative feedback in planning and review activities like in Kanban or Scrum. Defect tracking helps to measure quality, and tends to be more qualitative, though some may be quantifiable. Traditional defect tracking in software means documenting any bugs or errors that need to be fixed, which can be fed into the task queue during planning. This can be extended to other disciplines by clearly

describing shortcomings in a design, ideally by comparing to requirements. Useful defect tracking information includes when it was discovered, how it was discovered, and how to recreate the issue. This will provide the foundation for improving the design. An alternative approach would be to log the rate of acceptance or reasons for a client refusing to accept a finished task.

Two additional metrics that are more traditional but still provide necessary feedback are tracking costs and expenditures, and monitoring if high level budgets and schedules are being met. Traditional management strategies prioritize these metrics as a key way to evaluate performance, but this framework suggests using them only as a guide and an alternative way to visualize and quantify value delivery, rather than a key planning and control tool. A client may also prefer progress feedback in this way. Traditional project management artifacts like Gantt charts, PERT charts and accounting and budget sheets can be used to provide an alternative view to project progress, but using them heavily for planning and controlling is discouraged.

3.7.2 Prototyping for Design and Customer Feedback

A primary value in agile methodologies is that delivering working software is more important than comprehensive documentation [24]. Having a working product or prototype allows feedback from customers and the design team directly from the system rather than the abstraction layer represented by the description of the system. Adopting this attitude and looking to deliver a prototype or an artifact that represents direct value, not a description of future value, to the customer and stakeholders regularly is key to applying agile to the cross-disciplinary nature of mechatronics.

As mentioned previously in Section 3.2, prototyping across different disciplines requires varying commitments of time and both capital cost and cost of time spent. Time cost comes from the team members or other resources being compensated on the basis of time, and also the cost of delay (where delivery of value later impacts revenue or other measures up until delivery, and potentially beyond, for example if a competitor develops a larger market share during the time to delivery).

Part of effective prototyping is understanding how to create the greatest value (usually lowest cost and time), especially early.

The underlying goal of prototyping and testing is to increase the iterative rate of direct feedback to the design team as to the fitness of the design or current embodiment. When planning prototyping and testing activities, it is important to consider what information is being gathered. The team should figure out how to setup the "experiments", with a testable hypothesis or goal and way to evaluate the results (like the acceptance criteria of any task). Leveraging the direct feedback available through prototypes can significantly help to reduce technical risks, and to some extent other risks as well (like market risk, when customer feedback as to suitability can be obtained early). It is also important to consider feedback on the effectiveness of prototyping practices, which is part of process and management feedback.

3.8 Prototyping Methods for Mechatronics Design Projects

Prototyping creates an embodiment of a design idea that allows some interaction with developers, users or the environment outside the system, with the goal of obtaining feedback on the fitness and functionality of the design. Software prototypes are seen typically as a partially finished product that can be release for use and will evolve, as the build time and cost is almost zero beyond the development itself. Physical hardware is not so simple, as the cost to create an equivalently functional prototype can be very large. It is suggested that the team broadens what it means to deliver a working product. Rather than creating an actual embodiment of the current state of the design, the goal should be to deliver a result that provides value and the possibility of feedback. This idea can be extended to encompass any work completed, but the team must look to balance different approaches and consider what feedback provides the most value for the project to progress. This framework encourages the delivery of prototypes that will provide direct feedback as often as possible over simply documenting progress on a design. The latter may still have value for specific tasks or parts of the product or system however, and selecting appropriately is part of the

challenge of completing the project. A combination of techniques or methods is needed to achieve the goals of the project and work around some of the challenges present when dealing with physical products. Two key strategies applicable to all elements in a prototype include creating a custom implementation or incorporating COTS or outsourced components. Throughout prototyping and testing, principles of agile methodologies and agile design should be considered. There are a variety of ways to reduce time and cost when creating prototypes, though a development team should always understand the goal and how to evaluate the result. To aid in testing and evaluation, a prototype should be designed to enable observability, accessibility, and adjustment [19]. Examples of possible prototyping approaches for software, firmware, electronics, mechanics and integration tasks, which are all prominent in mechatronics design projects, are presented below.

3.8.1 Software

Modern agile software development regularly leverages numerous techniques including unit testing, test-driven development, continuous integration and continuous deployment, static analysis, etc. Describing all of these techniques is beyond the scope of this work, though some prototyping methods are described.

Creating a mock-up of human interfaces, graphical user interfaces or other parts of the user experience can help fine tune the design before significant effort is placed on creating the real implementation. These parts of a software system can represent significant effort to create or change, and can be difficult to evaluate objectively. Using a readily debuggable, sandboxed environment to test software internally or with a small stakeholder test group before release can help contain issues early. When possible, leverage readily available software frameworks and use included example projects or tutorials to rapidly deploy a prototype. Examples of such frameworks can be found in Table A.3 under the tag ‘SW’.

3.8.2 Firmware

Firmware development encounters different problems than on regular software stacks. Dealing with the hardware aspect can be more challenging as issues may arise in a greater number of places, making them more difficult to isolate. While on a software stack, the hardware is usually reliable and known to be working, an embedded firmware system can not always expect the hardware to work properly, especially in the prototype stage.

Vendor or third-party development or prototyping boards can allow for early firmware development with real hardware before product hardware is available. Ideally, a hardware prototype should be built early, even if this is simply an in-house set of development boards or similar generic tools. These generic tools may be reusable in other projects as well. Consider using a test-driven development approach for firmware, such as Grenning's [40]. Mocking or simulating hardware or parts of software can help isolate certain modules for testing before the hardware is available. Platform agnostic code modules can allow testing or use without real hardware. Ideally, these modules can be applied across different hardware or even compiled for a desktop machine, especially when combined with mocking or simulating dependencies. Using alternate non-optimized frameworks or platforms for demonstration purposes or to create a minimum viable product can speed development. This may sacrifice some requirements that are not critical for a basic demonstration, but should be solvable in future prototypes. Examples of suitable rapid development ecosystems can be found in Table A.3 under the tag 'FW'. Building out a custom set of modular libraries applicable to the current project may allow reuse in other projects. At the very least, some of the design patterns used should be reusable, but platform-specific drivers, helpers, debugging or utility functions can often be carried between projects, especially if the platform is the same or similar (same microcontroller, same microcontroller family, same microcontroller architecture, platform agnostic code that can be compiled for a different target). As a team gains experience with multiple projects, this reusable collection of modules will grow. Purchasing tested and documented software packages can speed development and/or address technical risk in subsystems. Examples include tested communications protocol stacks and real-time operating systems.

3.8.3 Electronics

Most electronics prototypes are relatively inexpensive and fast to develop compared to mechanical prototypes, at least when staying within relatively standard technology domains. It is important to consider the cost of printed circuit boards and components relative to the time cost of development. While the cost of a prototype may be astronomical relative to a volume production device, creating multiple prototypes to accelerate other project tasks, like firmware development, may be well worth the cost.

Simulation is a good option for subsystems with little code, like power supplies or analog circuits. Examples of simulation software tools can be found in Table A.2 under the tags 'Sim' and 'EE'. Other electronic design automation (EDA) packages are available for simulation as well, many with integration between schematic capture, PCB layout and simulation. Consider integrating EDA packages with mechanical computer aided design (CAD) packages as well, to gain insight into mechanical form and behaviour. For example, the geometry of a PCB may be all that is needed to verify mechanical fit or design an enclosure, and this information may be easily transferrable.

Use existing development boards, breakout boards or prototype boards from the component manufacturer or different third-party vendors to have an example of working hardware. Examples of the wide variety of vendors providing these products can be found in Table A.1 with the tag 'Elec'. This will reduce effort in testing and debugging a simple prototype platform, and leave more time for prototyping the actual product. Furthermore, consider modifying or hacking these existing products to meet the needs of the prototype (swap out components to meet different specifications, use green-wire fixes, combine multiple items, etc.). Using solderless breadboards or solderable protoboards can speed fabrication of simple, testable prototypes. Creating custom breakout or prototype boards for the project can be done if commercially available ones are not available or to gain experience designing for and assembling the components. Different design characteristics (high speed, low noise, radio frequency (RF)) may make custom PCBs the only viable option. Many component suppliers provide application notes or example designs that can be leveraged to quickly create a working design appropriate for current project. It also allows different subsystems to be tested

independently, either on separate PCBs or by logically isolating them on a single prototype board. These subsystems may be electrically isolated, even on a single PCB, by linking subsystems with jumpers. This approach also makes the integration of test equipment easy. Custom breakout and prototype boards should be designed to make modification and green-wire fixes easy, and ideally their applicability outside of the current project should be considered. Of course, a balance must be struck between prototype speed and how finished, complete or flexible the design of the prototype or testing aid is.

Use rapid and inexpensive manufacturing options for printed circuit boards that are readily accessible online. Examples of PCB vendors are list in Table A.1 with the tag 'PCB'. This is applicable to both custom breakouts as well as full product prototypes. It is also supported by online sources for solder paste stencils if hand-soldering is not practical. Externally manufactured items will incur a lead time. It may be beneficial to stagger the development of different subsystems so that the lead time for one may be used to design or test another, leading to a constant progress flow. As an alternative to outsourced PCB manufacturing, prototypes may be created through in-house etching or CNC milling of blank copper-clad board. While turnaround for simple projects may be rapid, complex designs with two sides, vias, plated holes, small package technologies and small and dense features can be challenging. The characteristics and tolerances of these boards are also difficult to control, and may be unlike a commercially manufactured equivalent, which is especially important to consider when dealing with high speed signals or RF features. Finally, the solder mask and silkscreen on a commercially available board can significantly aid assembly.

Consider the components used in manual assembly and for test. These may not be the same that would be used in a final product, but the design for each should be considered. The requirements for reliability, ease of manufacture, and ability to make adjustments or changes will impact the selection. For example, a prototype may use screw terminals or 0.1" headers for connectors, while a final product may use a board-to-board or board-to-wire small-pitch, machine-crimpable connector family, or an FFC (flat flexible cable). The board space and enclosure requirements are likely different for each.

Hand assembly is likely appropriate for small-volume prototypes, aided by soldering irons, hot air stations, and basic reflow equipment. Small surface mount packages may be assembled by hand-spreading solder paste with a prototype stencil, placing components manually with tweezers or a vacuum pick-up, and reflowing with an inexpensive reflow oven, modified toaster oven, or even a hot plate. Where a greater quantity of prototypes is needed (for example with multiple identical components combined to create a system, or for small-scale alpha or beta test deployments to customers), short-run contract manufacturing may provide a greater value. Assembly of even a small quantity of prototypes manually can become very time-consuming (in some cases as few as five or ten, depending on complexity). Often, vendors offering small run services will have a library of standard components that, if used, can significantly reduce cost and lead times. Prototype designs may incorporate these components early to efficiently leverage these services. There are a growing number of companies providing PCB assembly services (details in Table A.1 with tag 'PCBA'), some with the capacity to completely manufacture and fulfill low quantity production orders. There may also be opportunity to split assembly, with standard components handled commercially, and the development team adding the small number of unique parts afterwards.

The quantity of prototypes created should be appropriate for the stakeholders who need them. For example, once a hardware prototype is tested, it may be then used for firmware development. Additional hardware testing may benefit from additional prototypes. Some testing may be partially destructive, which will increase the number of identical prototypes necessary.

Different subsystems may benefit from different prototyping approaches. For example, microcontroller development boards will help early firmware development, while power supplies or analog front-ends may benefit from more detailed design and simulation. Note that subsystems may have unintended influence on each other, and they should be developed and tested in an integrated manner to complement independent testing, especially later in the development cycle. Consider applying the principles of test-driven development to electronics systems, as discussed in greater detail below in Section 3.8.6.

3.8.4 Mechanical Elements

Mechanical prototypes can become very costly very quickly, depending on the scale and complexity of the system. While simple enclosures or small, low-precision devices are more readily built, large assemblies, exotic materials or precision machining requirements can cause the capital cost and time of prototyping to balloon. Alternative techniques can control cost by prioritizing similarity over true reproduction, and may be described as mock-ups (non-functional but geometrically similar) or models (mathematical and behavioural similitude) [43]. Described below are a number of mechanical prototyping techniques.

Simulation and modeling provide an excellent opportunity to avoid the capital costs of prototype fabrication. It is important to ensure that such approaches provide value and useful information that is similar to the direct feedback obtained by a physical prototype. This is aided by defining a clear way to assess and communicate the value and goals of the simulation and modeling. Some of the advantages to simulation are that changes are inexpensive and relatively easy with great similarity to software, and that a model can start as being approximate or partial and evolve with the project. Mechanical CAD can provide a very detailed visual understanding of the design and offer feedback with built-in tools. Team collaboration is readily achievable, especially with the aid of some newer online tools (for details, see Table A.2 with the tags 'ME' and 'Collab'). Areas of simulation typical of mechanical design include finite element analysis, computational fluid dynamics, thermal studies, motion studies, interference and tolerance analysis, and assembly and maintenance understanding. It is also possible to simulate some manufacturing processes (like injection molding and machining) to improve the design. While many simulation packages offer excellent methods of visualization (especially those integrated with CAD packages), simulation may be entirely numerical, especially when custom solutions are used. Various simulation software tools are available, some examples of which are listed in Table A.2 with the tags 'ME' and 'Sim'.

Physical prototypes can use multiple approaches. They may be scaled, representative of only shape or other physical characteristics while compromising on other specifications (lower strength, for example), or demonstrate functionality without being optimized for aesthetics, size or cost.

Fabrication can be accelerated with rapid prototyping, applying different methods to different types of prototype. Additive manufacturing methods or 3D printing (fused deposition modeling (FDM), stereolithography apparatus (SLA), selective laser sintering (SLS)) can be accomplished either in-house or outsourced. This approach can provide fast, inexpensive, geometrically accurate models but often do not reflect the material properties of a final product. However, there are growing examples of directly using the output of SLS processes in final products [85, 50]. Subtractive manufacturing allows the use of a wider range of materials. Tools like CNC or manual mills or lathes, or CNC laser cutters or waterjet cutters are similar or the same to what would be used in a production environment. There are prototype services available locally or online to reduce lead times (see Table A.1 with tags 'CNC' and '3DPrint'). Small teams may find high quality subtractive or additive manufacturing equipment expensive to bring in-house, though smaller scale options like desktop milling machines or small laser cutters may provide some benefit. The value of purchasing tools that require significant cost, training and time will have to be considered carefully against the ease of outsourcing. Developing an effective collaborative relationship with the fabrication team can improve the efficiency of low quantity, quick-turn prototyping requests. This will also help the development team enhance their understanding of design for manufacture.

Alternative prototyping approaches include stacking layers of 2D material (i.e. lasercut or waterjet cut) to create 3D embodiments, or using alternative materials like cardstock or wood to create low fidelity mockups. Modular assembly products like aluminum T-slot extrusions and accessories are readily adaptable to prototypes (and sometimes low quantity finished products). One example is the the product line from 8020 Inc. (details in Table A.1). When precision parts are required, try to use COTS components to reduce machining costs. A good example is the use of mounted bearings instead of machining custom housings. Taking advantage of features like captive hardware or keyslots will reduce assembly and disassembly time. Incorporate adjustability into features where practical, for example by using slots instead of holes. Finally, consider applying the principles of test-driven development to mechanical systems, as discussed in greater detail below in Section 3.8.6.

3.8.5 Integration

Mechatronics projects must consider integrating different subsystems and also different domains throughout the prototyping process. Simulation provides some opportunity for this, especially with the possibility of multi-physical simulation. Hardware-in-the-loop (HITL) simulation can overcome some challenges when testing is difficult or risky in a system's regular configuration or operating environment. Packaging prototype electronics into a mechanical prototype must be sufficiently robust to survive a test environment, but it should not be expected that the design be finalized. An effective prototype should be easy to assemble, disassemble, access and fix, as this will happen much more often than in a final product. For example, making a small adjustment to the system should be achievable without removing many fasteners. Otherwise, too much time will be spent making adjustments, or adjustments will be avoided, perhaps to the detriment of a well-tested system. Integrating parts of a design as early as possible is important to understand the interaction and interference of different subsystems. Understanding these limitations early provides ample opportunity for this feedback to be integrated into the design.

3.8.6 Cross-domain Test-Driven Development

Test-driven development (TDD) is a technique in software development where the unit test code for a specific function or module is written before the implementation code. This forces the developer to actually write the test, and also to think carefully about the purpose, behaviour and interface of the function. It also encourages only implementing the minimum functionality and focusing on the task at hand, rather than introducing unneeded features that increase complexity and opportunity for failure. Often, writing unit tests for embedded software or firmware that relies on hardware means this behaviour must be "mocked" or "stubbed out" so testing can be done in an isolated manner. These goals make sense in an agile development paradigm, and also fit well into the pattern of understanding the goal or hypothesis of a testing or prototyping activity and how to measure its value (i.e. does it pass the test). Test-driven development may be applied to electronics

and mechanical prototype development, as demonstrated by the WikiSpeed project [26].

By considering energy and information flow into and out of different subsystems, it is possible to isolate them and stub out the input/output dependencies using a testing framework. This can be aided with functional modeling captured in appropriate artifacts. There are numerous approaches to functional modeling across different disciplines [31, 30], while the Integrated Functional Modeling framework may provide an opportunity for a more inclusive method [34]. In electrical systems, test equipment like function generators, DC loads, power supplies and HITL simulation can be used. This would be especially effective coupled with individual breakout boards or isolated subsystems on a single prototype. Computer controlled test equipment can make test suites repeatable and automatic. Mechanical systems may be more challenging as energy has more forms and these forms are often more difficult to accurately control without expensive equipment. A good example would be the use of an instrumented dynamometer with controllable load profiles for angular or linear motion along with a controllable motor to test a gearbox, clutch or transmission. Different sensors and actuators can help apply different types of load to mechanical components. Sometimes, matching the interface is sufficient. For example, a dummy payload with the correct size and weight may be appropriate for testing a robotic actuator, while a simulated motion rig might be appropriate for testing a payload. This is especially true when coupled with appropriate electrical stub elements. The goal is to create mockups or stubs with the correct interfaces and some valid behaviours to test subsystems. A full physical system may not be necessary, just enough to realistically simulate a behaviour of interest. Like other areas, an electrical or mechanical test framework can be designed to be reusable across multiple projects.

3.9 Design Project Documentation

Traditional design processes and project management often encourage significant documentation as a key part of the deliverables guiding the progress of the project. Agile methods focus instead on effective communication and collaboration which may partially be accomplished with documenta-

tion, but also through face-to-face discussion and other rapid, high bandwidth techniques. Before documenting, consider the goal and purpose of the documentation and consider the best way to accomplish this. Generating documentation may not be the approach that generates the most value, or if it is, there are different media appropriate for the task. Some of the key reasons for documentation include:

- Enhancing the understanding of a design or process.
- Providing some record of design process and changes.
- Sharing knowledge amongst the team and with the client.
- Recording knowledge generated by the design team that may be useful in the future for the current project or other projects.
- Creating a snapshot, frozen instance or reference frame (a version) of the design or a set of known information to compare against in the future (for example, to compare test results against for evaluating improvements).
- Creating a reproducible process, procedure or embodiment of the design.

It is important to understand the limitations of documentation within the overall goals of communication, collaboration and knowledge transfer. Creating good documentation that is effective at communicating knowledge can be difficult, and a team cannot rely solely on this approach. It must be supplemented with other collaboration practices like informal discussions and team meetings.

Documentation becomes rapidly outdated if not maintained to reflect any changes in the design. The effort put into documentation that is inherently static then loses value. One of the key challenges of documentation is ensuring it reflects the state of interest, especially if that state is the ever-changing current state of the project.

The project team must balance documentation activities with overall agility. Documentation can be time consuming, and documenting all aspects of a design and process in detail takes major effort

that will slow down progress. This framework suggests using a minimum set of documentation artifacts to reduce effort of creation and maintenance [75]. The team should tailor the minimum set to ensure all aspects of the design and process are documented with minimal but sufficient detail. A recommended minimum set of artifacts are outlined in Section 4.2.7.

Documents should match the project characteristics, especially criticality and culture. For example, a project that implements a continuous release policy for software will benefit much more from automatic documentation generation than a project where improvements or revisions are difficult after the product release. Even in projects without a clear need for agility, it is unlikely documentation with agile characteristics will have a negative impact.

Larger teams or projects may need increased formality in their methods of communication [44]. Part of this communication is documentation, and it should be adapted appropriately. This framework focuses on smaller teams, and suggests several practices for improving the agility of documentation. Documentation is often viewed as final, stamped or permanent, but keeping documentation as "live" as possible and letting it evolve rapidly with the design can keep artifacts up to date and synchronized. Live documents can be supported by archiving or snapshotting documentation representing revisions, versions and a history of progress. Methods suggested for keeping live documentation up to date include:

- Keep document formats simple.
- Make updating the documents easy by minimizing the process to do so.
- Keep documents coupled with the design source (for example, using automatic code and comment documentation generation tools, linking documents directly into source files, creating inline notes in source files).
- Create templates that are fast to access and duplicate.
- Leverage automatic documentation tools to make documenting activities easy and coupled (for example, automatic source code documentation generation, tools that propagate changes in 3D CAD models to 2D drawings).

- Adopt methods that reduce duplication of information (for example, using model-based definition to store manufacturing information directly in 3D CAD models to eliminate the need for 2D drawings [4].)
- Choose tools that make documentation and archiving possible and easy. For example, archiving and searching even informal discussion can be easy if carried out on digital chat or other collaboration platforms (see Table A.2 with tag 'Collab').

Selecting appropriate tools is very important to ensure a team achieves sufficient levels of documentation efficiently. Easy to use tools will improve the ability of the team to maintain artifacts regularly and with discipline. Ease of use is subjective, and the team should work to find solutions appropriate for the whole team. While experience makes using a tool easier, different projects may benefit from alternative tools that better fit the specific project characteristics and domain.

The alternative to live, living or working documents is fixed, permanent or hard documents. The goal within this framework is to only create fixed documents for parts of a project that are not supposed to change once it has been released. A good example is a contract with a client or hard external requirements documents, like summaries of standards or regulations, or internal standards or procedures. Improving the agility of documents often requires more thought and setup, and thus this effort must be balanced with the value it will provide throughout the project. An alternative to a truly fixed document is to spawn versions from a live document template that is more readily updated. Even a contract can be a live template that gets improved regularly, and then copied into unique, project-specific versions.

Part of documentation activities is the management of collective organization knowledge for future use. As a team generates knowledge during a project, it is important to communicate, store and track what is known by the team. Making design elements reusable as part of an agile strategy means greater effort in documenting and generalizing the elements may be necessary. This is especially important if the usability is to extend beyond the current team to others in an organization or outside an organization. For example, a team may wish to open source a portion of work or a tool for the wider benefit of others, or to aid in future use across other closed projects. The preparation of

documentation includes determining the value of the information and an appropriate level of detail for the information so it remains usable without needing too much effort to create. A bare level of sufficiency should be targeted especially if immediate use of the documentation is not expected. Documents should be suitable for storage, but also suitable to aid communication amongst a team or to bring new team members up to speed. It may be useful to support the documentation with practices such as weekly or monthly seminars to facilitate transfer of knowledge within the team. Finally, knowledge and information should be stored in an organized fashion. Building a knowledge repository contributes to the organization's intellectual property portfolio. Part of this repository must be a way to catalogue and find information within it.

While a design is expected to evolve and the documentation must be able to reflect this, some changes may originate externally. For example, vendor may release errata or standards may be updated. If possible, documentation that captures this externally supplied information to be coupled as closely to it as possible, and avoid duplication where possible. The value of reproducing information available from the original source is likely to be low, but it is still important to sufficiently document information like why certain COTS components were selected, such that they can be replaced or substituted if necessary (for example, if a product transitions to end-of-life).

A major part of documentation is intended for customers or users. For example, user manuals, API information, instructions or datasheets are all targeted to outside stakeholders. Documentation like this should be considered a feature or subsystem of the product, and be treated as a deliverable like any other part of the project. It is not the same as internal documentation that is readily supported by other types of interactions and collaboration. The time and effort for completing these deliverables must be included in the project planning and estimation.

As mentioned previously, the formality and detail of documents should be appropriate for the culture, criticality and market of the project. Some projects and documents demand greater formality when meeting different specifications or standards, especially in medical, aerospace or automotive markets. Accounting and tax advantages may also drive greater formality in some documentation. While the resulting documentation may need to be formal, there is usually the possibility of getting

to the point of document delivery in an agile fashion. It may not be necessary to create documents in a traditional waterfall fashion even if this is alluded to by the specification or standard. For example, it may be possible to reverse engineer the final design back to requirements to meet traceability needs. Such requirements can again be looked at as tasks or deliverables for a project instead of an internal part of the design process, as the documents are now also targeted at third parties in the event of review or audit.

In summary, there are several types of documentation artifacts. Fixed documents should be avoided as most will undoubtedly need to be updated over time, and it is better to shift away from this mindset. While documents like contracts may be the exception, even these can be treated in a more agile manner. Live documents should be the goal of the team, especially when used internally. They should be guided by development and support agility. These documents should be simple, easy to keep up to date, and coupled to the source of information. This may be supported by automatic the generation of documentation where possible. The project development history can be investigated by regularly creating an archived snapshot of the current state of the project. Documents should be customized to a project, perhaps with the aid of some standard templates. Documentation targeted at users, clients or other outside stakeholders may evolve from live or fixed internal documents, but should be treated as part of the project's deliverables and tasks. Overall, documentation should strive to provide information of high quality over high quantity.

This framework encourages a minimal set of relevant documentation artifacts, customized to the project at hand. A sample set is described in Section 4.2.7.

APEGA outlines a number of requirements for documentation practices [91]. To meet these, the organization and project team should create a process or procedure that is expandable, with supporting tools to handle the variety of document types that may be encountered. In particular, a team working on mechatronics projects will work with a wide range of engineering documents that may need authentication, including software or firmware source code and binaries, electronics schematics and PCB layouts, wiring diagrams, mechanical drawings, instructions or procedures, etc. The methods and tools used for authentication practices should be documented in the PPMP.

The practices should try to meet the goals of simplicity and being the easiest option, and should be adaptable to meet unforeseen needs or as the practices prove insufficient. The Professional Practice Management Plan has elements that are tightly linked to the methodology, and is described in *Guideline for Professional Practice Management Plans* [90]. An inventory of resources, tools, and equipment should be included, which can tie well into other parts of organizing a business and project team, like accounting and shared resources. It will also help new team members become familiar with what is available to them. There should also be an inventory of technical capacity and how to evaluate it, which ties in well with knowledge sharing, storage and team collaboration. A key part of the PPMP is to outline the quality control and risk management practices of an organization. This framework tries to integrate a basic level of this into the general practices and attitudes of the team by focusing on value, feedback, customer communication and risk-oriented planning. Some additional discussion is also presented in Section 3.10 below. Overall, the PPMP forces an organization to explicitly explain its activities and processes in certain areas, which forces a team to better understand how to carry out its project design methodology. Furthermore, it is expected that a PPMP be updated and improved regularly, which aligns very well with agile principles and methods. A live document system outlining a team's methodology that evolves with the team would be appropriate, with an appropriate level of formality and detail for practices relevant to the PPMP as well as the feedback process used to update and improve it.

3.10 Risk Management Considerations

Risk management is integral to the decision making process guiding a design project from beginning to end. It is an ongoing process, and varying types of risk need to be addressed differently, often at different times. During project initiation, the team should develop an understanding of risks in the project and any key high-risk areas. This is tightly linked to the evaluation of project characteristics. Areas of uncertainty should be identified (particularly for technical or market risk) as well as known, characterized risks and how the organization handles these.

Outcome- or stakeholder-oriented risk categories include occupational health and safety risks, environment and public health risks, legal and financial risks, and reputation or societal risks. These risks are often linked to standards and regulations, and impact the criticality of the project. Some understanding of these is necessary in order for a project to proceed, and the level of risk should be compared to the risk tolerance of the organization, team and client. The ability to mitigate identified risks should also be evaluated.

Source-oriented or internal risk categories include technical risks, organizational risks, market risks, and external risks. Technical and market risk may have significant uncertainty, as is typical of exploratory projects. Part of the effort in creating a basic draft architecture early is to help understand technical risk. Understanding at least what kind of market the product targets helps, as discussed by Highsmith [44]. Additional market research may be necessary to reduce the uncertainty and risk to an acceptable level. The organizational risk, how the organization will be able to handle the project content and strategy, is part of the analysis in determining how well the project matches the organization and team culture.

If an area of risk or uncertainty is deemed significant in its impact on successful project outcome, it should be considered as early as possible. As these areas of high risk are addressed early, the overall project risk will be reduced more quickly. Visualizing total risk over time may help the team understand project progress. This could be accomplished with a risk burndown chart (like in Scrum) or a diagram similar to a cumulative flow diagram (CFD) (like in Kanban, see Figure 2.22). Completing tasks helps address technical risk if uncertainty is reduced and organizational risk of meeting deadlines if remaining effort is reduced. Managing risk helps to improve the probability of technical and commercial success.

Risk and uncertainty should be considered regularly as iterations and cycles are planned and reviewed, with the recommendation that task priority be linked to higher risk parts of the project. Part of the feedback in projects is understanding how to monitor and measure the risk to be meaningful and helpful in decision making. The team must identify risks that need controls, and implement controls to mitigate this risk. The effect of controls should be measurable to understand

their impact on mitigating the risk. The team should continue mitigating risk that remains in ongoing activities. A risk-oriented approach may be used as part of the strategy in creating definitions of done and acceptance criteria.

Higher risk areas (higher probability and/or consequence) will benefit from increased formality in analysis and management to ensure comprehensive treatment, using tools like Failure Mode and Effects Analysis (FMEA). An organization and team should become comfortable with understanding the threshold of when these should be implemented. A significant challenge exists in developing a basic qualitative risk assessment process that can help a team determine objectively the areas of higher risk. Risk management activities using more formal techniques should be integrated into the activities and tasks of a project and methodology. APEGA provides good guidelines on risk management tools and techniques, including approaches to mitigating risk in different ways [88].

3.11 Summary

A design project methodology rests on a lifecycle representing the entire lifecycle of the project, which will be similar across a range projects but linked closely to business model. The business model of a mechatronics project may vary or use multiple approaches, depending on the ability to make improvements after the release of a product or system to clients and customers, a practice that will improve the agility of the product and organization. The method should focus on iteration and feedback to ensure continued delivery of value.

Feedback driven iteration is present both in the approach to project management as well as the evolution of the design. Iteration remains a key focus while balancing incremental and progressive design iteration strategies, and effort should be made to create prototypes or leverage other opportunities for rapid, direct, first-order feedback. While the model of evolution and iteration of the design will certainly impact task distribution as part of project management, it is not directly tied to the overall management approach. The ease with which a design can be changed will have

an effect on design iteration rates and stategies. Visualizing change capacity in the design can use the concept of "shearing layers", which in mechatronic systems will depend on both the feature or subsystem, as well as its dominant discipline. The iteration style used in project management may be unique and should be appropriate for the project characteristics (described in Section 3.5) The same characteristics should be considered when structuring design iteration. Design iteration and improvement should not be limited, and instead should be agile, continuous and encouraged. The goal is to keep the design easy to change and adapt, as is discussed later in Section 3.6. A tiered approach to iterating during a design project can help a team implement pull-based methods at lower levels and time-boxed methods to achieve synchronization at higher levels. Furthermore, the formality of each tier and category of iteration can be adjusted to meet project characteristics, such as the size of the team and client requirements.

The team must be committed and organized to support the iteration and agility. There are several approaches to organizing a mechatronics team, and ideally should divide into subteams and distribute tasks by feature or subsystem instead of domain or discipline. Challenges around dealing with individual specialization and synchronizing the team with other stakeholders must be considered. It is desireable that a design team continues to grow, which may include more formal learning methods through industry or professional development.

The project characteristics should match the ecosystem and culture of organization, client, market, outside requirements or stakeholders, and must all fit into the agile approach. It is important to consider all stakeholders early in the design project, even those that will interact only in later stages of a lifecycle. Large impediments or differences will reduce effectiveness. Exploratory projects are well suited to this agile framework, though higher criticality or greater prototype cost will require additional strategies or formality. Understanding the market and the business model applied to the product or system under development is important as well.

To complement an agile approach to the design process and project management, the design itself can be agile. Keeping a design simple, modular and reusable helps let the design adapt to change and uncertainty, while feedback can be achieved with the help of design features, close collaboration with

the customer, and leveraging rapid prototyping and development techniques. This is all supported by a minimum of relevant and up to date documentation, and the team maintaining a disciplined approach to development.

There should be special consideration of the methods of feedback and knowledge transfer. Feedback from both the design and the process help a team stay on track to deliver value, and is obtained through prototyping and review and retrospective activities. Wherever possible, quanitifiable approaches to feedback should be taken.

Rapid, direct feedback obtained from prototyping helps improve the design. The team should strive to improve the rate and reduce the cost of prototyping physical hardware to approach iteration rates possible in software development. Techniques for doing so are relevant to both individual disciplines as well as for system integration and testing tasks.

Knowledge propagation and archive is supported by documentation, which can take significant effort to create and maintain. While documentation is still important, it should be approached with the intent to be more agile and improve value proposition. The methodology must be adapted to meet external requirements imposed by standards or regulations.

Risk management helps address uncertainty throughout a project. It is important to evaluate risk early during project initiation, and continue to monitor and mitigate it throughout project. Formal tools should be used when risk is deemed to be above a threshold acceptable to the team and organization.

A team creating an effective methodology for working on a mechatronics design project should consider some key ideas. It is impossible to select a single methodology applicable to all problems, and there must be some adaptation to fit each situation. There should be a clear way to evaluate the value being added by elements and activities. Retrospectives provide an opportunity for the team to ensure every element and practice of the methodology adds value. Adaptive strategies should be favoured over predictive or anticipatory actions, unless there is a high possibility of future value through these actions. The methodology should be able to be changed and improved easily to help

the team be agile and evolve with adaptability. The team should avoid locking into a single path or trying to anticipate too much – anticipate needing to change, not what will change. Finally, it is important to strive for ease and simplicity in all elements, and fall back to the basic goal that every part of the methodology should help the team deliver value.

Next, the ideas and principles described in this framework will be applied to a methodology process model that can be used to manage a mechatronics design project.

Chapter 4

Proposed Agile Methodology Model for Mechatronics Design Projects

The framework described in Chapter 3 outlines a number of principles and topics to consider when a small team implements or follows an agile methodology for carrying out mechatronics design projects. In this chapter, these ideas are used to create a specific agile methodology model describing process, practices and artifacts. The model is structured with three stages as illustrated in Figure 4.1. Project initiation encourages the team to create a solid foundation of understanding and high level planning, while the project iteration stage allows the team to carry out development in an agile fashion. Project conclusion will vary depending on the outcome of the project and the next steps. The model has some implementation flexibility so it may be applied to different projects exhibiting varied characteristics, and balance the level of optimization planning and exploratory adaptation that is appropriate. Futhermore, the model is not designed specifically for mechatronics projects because it is discipline-agnostic and can accommodate a variety of discipline focused activities and elements relevant to each phase of the project. This approach fits well with mechatronics projects, which each have unique combinations of solution-oriented tasks.

Each project requires customized practices and artifacts pertaining to that project and team. The format and content of the artifacts should be practical for the team to maintain and update. The team should keep documentation artifacts to a minimal, relevant set. A suggested minimal set is described below in Section 4.2.7. There are numerous examples across a range of resources for different options, and the team may have existing templates that can be adapted. The artifacts and practices are expected to evolve across projects.

The model uses go/no-go decisions allow the team and client to decide whether the project should continue based on a review of current progress and value. The first go/no-go decision occurs at the end of the project initiation phase, at which time the team decides whether it will accept the project or not. The cyclic nature of the project iteration phase provides periodic opportunities for additional go/no-go decisions. The outcome of a go/no-go decision determines the project conclusion phase activities and goals.

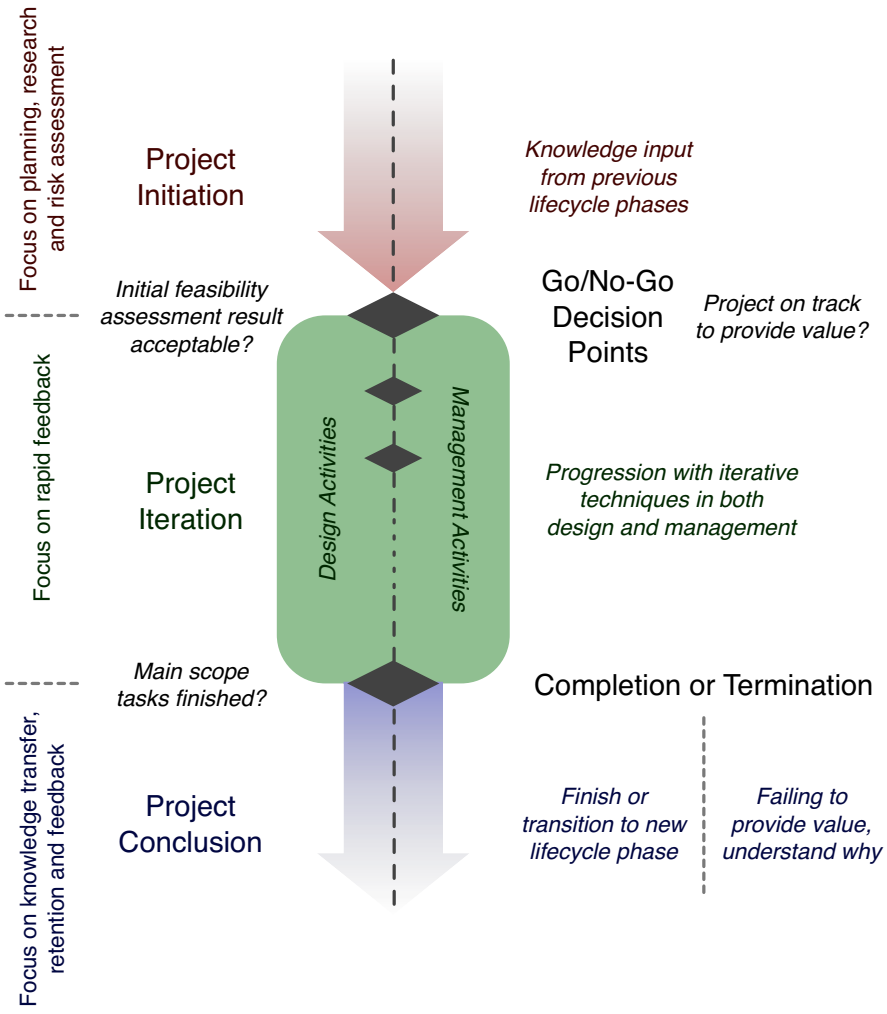


Figure 4.1: High level view of methodology model.

4.1 Project Initiation

The project initiation stage focuses on building a foundation of the knowledge and goals needed to make the main development stage successful. The team must understand the goals of the client and the key characteristics of the project. It is similar to the initial stages described by other methods and has similar activities (problem definition, requirements or information gathering, planning, project definition, feasibility assessment, market analysis, product concept, etc.) [29, 27, 43, 60, 67, 44]. At a high level, the team should try to get baselines or initial estimates for scope, schedule and cost, and develop a succinct understanding of the project from different viewpoints. Correlating these viewpoints to different stakeholders can provide additional insight. A basis of the practices, interaction, live documents, and other elements of the methodology should be created for use in the main phase. These are not expected to be completely correct or permanent, and should evolve with the project. The project initiation phase has three primary activity categories: stakeholder collaboration and engagement, planning and knowledge generation within the team, and creation of an initial and minimal draft set of artifacts. The effort spent on these activities will vary depending on the characteristics of the project, team, client and organization. The project initiation phase is visualized in Figure 4.2, and described in further detail below.

The information gathered in the project initiation phase should be carried out in parallel, as each area of interest will partially rely on the others. The order of tasks below provides a reasonable starting point, but the team should expect to iterate and jump between them. Different projects will focus more on one area or another. Each of the three main activity categories will be applicable to all tasks. Throughout project initiation, existing knowledge should be considered. This may come from the client itself, the collective knowledge and expertise of the team from previous experience, or through existing examples and templates of artifacts. The product or system itself may have pre-existing documentation outlining the current state of the design. This will vary depending on how the current project fits into the overall lifecycle of the product.

The end goal of the project initiation phase is to have enough information to make an informed

go/no-go decision on whether the team should continue with the project, while also creating a base set of requirements and goals reflecting the vision of the project.

4.1.1 Evaluation of Project Characteristics

Project characteristics play an important role in the approach to design and management. The characteristics are described in further detail in Section 3.5. The information about project characteristics can be gathered using several methods. Direct discussion and interviews with the client or other stakeholders (for example, potential customers) is one method. Reviewing any existing documentation prepared by the client (for example, a Request For Quote) is another source. Additional research by different means about the market, industry or technical domains may also be necessary.

Areas of risk and uncertainty that are uncovered should be investigated further throughout the project initiation phase. The team needs to understand what may be done to control and mitigate risks, or if the risks are deemed too large to make the project feasible.

Artifacts that can be created to start capturing the information and requirements being gathered include a project vision and scope document, as well as a feature backlog to organize requirements, features and specifications. A project vision and scope document should outline the high level goals of the client in a manner that both the client and the team understand, providing high level details about the initial scope of work involved in the project. It is also important to outline the project lifecycle boundary conditions (known inputs and expected outputs of the project) and to describe the definition of done or acceptance criteria that help the team and the client evaluate if the project is complete within the boundary conditions. These should be quantifiable if possible, or at least a method of evaluating the results should be defined. The feature backlog is used to organize requirements, features and specifications. If possible it can be directly integrated into the kanban board as discussed in Section 4.2.1, in which case it may be appropriate to include general tasks in the backlog. This artifact can be modeled after a number of options. Software requirements

are often recorded in use cases like those of Unified Modeling Language (UML) [69] or user stories [44, 53]. There are a variety of software tools that provide relevant functionality. Examples including Taiga, Trello and Pivotal Tracker can be found in Table A.2 with the tag 'Collab'. Projects with physical products may use a specifications matrix spreadsheet to capture this information. The QFD method represents a more formal approach [43, 55, 67]. It is suggested that a lightweight system similar to user stories is implemented, with consideration for the information presented in a specifications matrix and use cases. Of significant importance is to ensure the feature backlog contains acceptance criteria or definitions of done for each item. The purpose is to provide a metric or method to evaluate the item against during review. As a result, this metric will be a key requirement or specification making up part of the item. Ideally, the criteria will be quantifiable and testable, or otherwise provide a concrete goal. As an example, a feature may be to design a voltage regulator that can deliver a specified voltage at a specified current with a maximum noise figure. The acceptance criteria can be that the design is tested by prototype under the specified conditions and the noise figure recorded in a test results document. This document is then readily reviewed and the feature or task can be accepted. Other cases may be simpler, but the overall goal is to provide clear direction on what to review and how it should be evaluated. Throughout the project initiation phase, these artifacts should be added to and revised as requirements and goals are understood and clarified.

Culture

It is important for the team to develop an understanding of the client and the problem through discussion with the client. Additionally, the client must be made aware of the general approach to project management that is being applied with an understanding of expectations and responsibilities of both the team and client This includes outlining the agile nature of the framework and how iteration and feedback play a role in the project. Clear communication regarding estimation and chargeout practices is also important for a value-driven project, which benefits from flexibility in two or more of cost, schedule, or scope. Understanding the expectation of the client here will drive

the level of detail needed throughout the initiation stage. A client willing to embrace the agile approach will indicate a more minimal initiation phase and allow the team to move quickly to the main iterative phase.

Market and Business Model

Initial understanding of the market includes information like the expected size and the level of expertise of the users (how much knowledge or training they have had). It is also useful to know how the client envisions the product or service interacting with the market and the business model. The team should focus on aspects of the market that impact the design process and impose different requirements. For example, if the market is broad and large, the team should consider how to get feedback from the market early through early minimum prototype testing, customer interviews, etc. The level of expertise of the users will influence features and tasks related to user documentation, manuals, instructions and training material. An understanding should be developed around the criticality and culture expected in the market. For example, a medical or aerospace market has a different expectation of reliability and formality than a consumer or research market.

Technical Risk

The project team needs to understand what the technical problem and scope of the project is, which drives initial technical requirements. Any hard or critical technical requirements should be identified early, along with technical uncertainty or inexperience, as these represent areas of greater risk. The team should be able to identify what disciplines may be involved in the project, as well as what technologies will play a role. This helps the team understand what types of prototyping may be required and the prototyping iteration rate that may be possible. It is important to begin exploring any domain or market specific regulations or standards that may be applicable, as these will be hard requirements. Based on the information gathered here, the team can identify any preliminary concerns with competency. This may be supported by a review of the expertise inventory present in the Professional Practice Management Plan, described earlier in Section 3.9. Areas of technical

risk and an understanding of requirements should be re-evaluated after some initial architecture development.

Hard Requirements

Any hard requirements related to project management need to be identified. This is typically limits on budget and schedule deadlines. These need to be evaluated against the capability and available resources of the team and organization. For example, a large project with a short deadline may not be achievable given the team's size and other commitments. This is also related to matching the team's methodology with the expectations of the customer. If the customer is expecting a hard budget estimate, the team may need to reconsider accepting the project, or take a different approach than that outline by this model. For example, perhaps the project could be structured by completing an initial feasibility study (possibly paid) that involves some planning and analysis, then only pursue the project if there is sufficient indication that an agile approach is not better over a more traditional, predictive approach.

While both the budget and schedule requirements can be captured in the requirements artifacts, they should also be reflected in a preliminary budget estimate document and the preliminary project roadmap. Note that in this methodology, these documents are used to guide the project and help gauge progress rather than be a primary method of monitoring and control (which should focus on value and quality instead).

4.1.2 Type of Project Lifecycle

The team should develop an understanding of which parts of the product lifecycle this project will cover, based on the market and business model if this information is available. The goal is to help define the boundary conditions and scope for the project. The inputs and outputs of the project should be identified to outline what the state of the product or design is at the outset of the project, and to understand what the outcome and deliverables of the project need to be for it

to be successful and useful.

This can be completed through an analysis of initial information from the client. This is used to create an initial description of the the "definition of done" for the overall project, including any deliverables or key questions that should be answered. This should be included, along with a description of the high level scope of the project, as part of the project vision and requirements (feature backlog) artifacts. The resulting summary of project vision, goals and boundary conditions should be reviewed with the client for feedback and approval.

4.1.3 Team Organization and Responsibilities

The team should form a strategy on how tasks and work will be divided. How the team will be organized will depend on the skillset of the team and the technical content of the project. The options available to the team are described in Section 3.4. There should also be a discussion around the distribution of technical responsibility and document authentication as required by APEGA. While it is not necessary to assign individual responsibility at this stage, it is important the team has the capacity to authenticate documents that require it at the end of the project. This is helped by understanding the competency of the team and if there are any shortcomings.

The roles and associated responsibilities of different team members should be understood. Key areas of management responsibility include the client point of contact (similar to a Scrum product owner), and any other important stakeholder points of contact (for example, for communicating with a specific vendor or manufacturer). It may also be useful to designate a team member in charge of facilitating the design project methodology and guiding internal feedback activities (similar to a Scrum master role).

Decisions about distributing responsibility will likely be based on the experiences the team has had in the past both together and as individuals. The team may also wish to test a new or modified approach and evaluate effectiveness. As with any other part of the methodology, adjustments can be made throughout the project with appropriate feedback and review. Specific responsibilities of team

members may be documented internally in a live document that has a level of formality reflecting the criticality of the project and the culture. Key points of contact and methods of communication should be made clear to the client to support collaboration throughout the project.

4.1.4 Clarifying Collaboration

As mentioned in Section 4.1.1, the client and the team (and any other critical stakeholders) should have a mutual understanding of the collaboration activities, communication methods and commitments that are expected throughout the project. This may require compromise between parties to identify practices and schedules that will be acceptable for everyone. Recommended areas of collaboration include informal email or in-person discussion, regular formal reviews and planning sessions, demonstrations and tests, and high-level planning reviews and planning. An appropriate frequency of different collaborative activities should be determined (as needed, maximum time between, minimum total quantity, key times matching milestones or other events). The expected communication format and duration should be outlined, as well as any tools that will be used to support the collaboration. The scheduling constraints should also be discussed, and whether meetings must be planned well ahead of time (and how far ahead) or if on-demand discussion is feasible. Important and formally scheduled meetings or collaboration events may be integrated into the team's roadmap, task list, or other temporal planning artifact.

4.1.5 Preliminary High-level Architecture

The design team should begin drafting a preliminary functional model and high-level architecture as part of project initiation. In this work, the architecture and functional models are treated as a similar artifact, and be generally classed as architecture artifacts. Typically a functional model will reflect solution-agnostic information and the architecture will be more solution-specific, though this will not always be the case. Varying levels of detail or specificity are possible with both. The early architecture should remain solution-agnostic if possible, while falling within the design

space outlined by the project vision and initial features and specifications. It is noted that some requirements or client visions reflect specific technologies or techniques, automatically limiting the design space. Creating this architecture helps a team identify and clarify requirements, features and preliminary subsystems in a cyclic, iterative manner. The activity helps to understand possible technical areas and disciplines that will be present in the project, and also anticipating some of those across the broader project lifecycle. The preliminary architecture helps a team begin visualizing the design solution, and help highlight any areas of technical risk or competency limits.

The team should use techniques like brainstorming and client discussions to improve the requirements, scope and vision. As questions or additional requirements emerge, they should be discussed within the team as well as with the client to obtain feedback and strengthen understanding as quickly as possible. Artifacts that help capture this requirements information includes the feature backlog and specifications list, while the project vision and scope documents are also improved. It is noted that the team should draw on previous projects and experience to incorporate suggested requirements that are applicable across multiple projects. As a team builds experience through multiple projects, an understanding should be developed of what requirements should be focused on early and have greater impact on the progress of the project. It may be beneficial to create a "library" of requirements that the team can review during project initiations. This library can be added to during reviews carried out as part of the project conclusion.

Finally, the preliminary architecture (or functional model) should be recorded with living documents like a feature tree, a functional hierarchy, block diagrams or boundary diagrams [98, 69, 43]. An alternative that may be particularly useful for mechatronic projects is to use the IFM framework [34]. Different projects will benefit from different document types depending on the maturity level of the existing design and the clarity of the project vision and requirements, as well as the size, complexity and formality of the project. Examples of architecture artifacts including a high level boundary diagram and a hardware architecture block diagram are presented Chapter 5, Figures 5.1 and 5.3, respectively. It is expected that these documents will evolve with the project, and should be continually updated. As the number of documents increases, so too will the effort taken

to maintain them, and so the types should be carefully selected to most efficiently communicate the design. The architecture will grow as the project progresses, and creating a set of architecture documents that can be linked in a tree format may be an effective approach that also encourages modularization. Similarly, the IFM framework can be adapted with different modules or views added as necessary to reflect additional design information.

4.1.6 Iteration Strategies

As the team begins to understand the project requirements and characteristics, a strategy for the iterations can be developed and appropriate time-box target periods can be identified. There is a range of key iteration loops and activities that define the iteration. This is described in detail in Section 4.2.4. The iteration is reflected in several artifacts, particularly the kanban boards (see Figure 4.7) and roadmaps (see Section 4.2.5) that describe workflow and process. Iteration loops that primarily involve the main design team carrying out day-to-day activities represent the tightest feedback, and should be targeted for a kanban-like, pull-style approach. Those involving high-level planning or synchronizing with stakeholders outside of the main design team are expected to have longer cycle periods and may rely more heavily on time-boxes for effective collaboration between all parties. Key iteration periods to consider include the internal workflow and review rate, the client review rate, the rate of creating prototypes, the release rate for projects using continuous release, retrospective process reviews, major design/test/integration milestones, high level project planning and review sessions, and finally the entire project time period. Depending on the nature of the project, it may be possible that almost every activity rate is almost the same (except the entire project). While this would be ideal but some characteristics may make this difficult. The team should consider prototyping capacity and lead times that fit with the preliminary architecture, as well as project formality and experience, when selecting target iteration time periods. The formality required in different review and planning activities will increase management overhead and make fast iterations less efficient. The time periods will also be impacted by the client timeline and their commitment to collaboration and feedback. Finally, there may be variation in iteration periods

between different disciplines, again related to both the capacity for prototype development and the organization of the team and distribution of tasks.

As a starting point, there should be at least four opportunities for iterations of client involvement that result in a go/no-go decision, beginning with the review included in project initiation and ending with the the final go/no-go of the main project iteration phase. Time-boxes for most iteration activities should not exceed one month. Iteration times should be identified as either fixed for flexible for each point. Those used to meet external requirements or with greater synchronization complexity are more likely to be fixed, while those without dependencies outside of the design team can be far more flexible. Note that regular retrospective activities will provide opportunities to adapt iteration rates and time-boxes to the project as the need arises.

A key aspect of providing the feedback necessary to review and improve project management is the time spent by team members on different activities. Time tracking should be carried out as a core metric of the process. Time should be recorded in an artifact, ideally based in software to allow easy and potentially complex analysis and review of time spent on different activities. While detailed tracking will help optimize the process in the long term, starting with even a basic time tracking approach will help the team improve over time. Client chargeout may also rely on tracking time accurately.

Iteration details may be recorded on internal documents to guide the team. The kanban boards used by the team help visualize and provide feedback on iteration. The kanban board(s) should be created during project initiation to guide the team during project iteration. The initial high level roadmap should also be developed to record and visualize key activities supporting iteration. This may include hard schedule requirements and initial high level feature, delivery or demonstration goals. Dates may change as the project evolves and progress is made. Review and feedback with the client and the team is formalized with regular go/no-go decision points.

4.1.7 Review and Go/No-Go

The end of project initiation is marked with the formal project review and go/no-go decision with the client. The team should consider project characteristics when making a go/no-go decision. In particular, concerns that may trigger immediate project rejection include an incompatible culture or development style, major technical risks or areas where the team lacks competencies, or hard requirements that are not achievable. The review should cover a number of areas and artifacts:

- Project vision, scope, boundary conditions and definition of done
- Preliminary requirements, features and specifications
- Proposed high level architecture or functional models
- Initial high level roadmap and target delivery points
- Initial budget
- Understanding of collaboration commitments and activities

The review process may indicate that some artifacts need improvement to reflect the understanding of the project, or that the team needs more information about the project. This should trigger an iteration of the project initiation phase to address the shortcomings.

At the end of the phase, the team should have initial living documents for the project vision and scope, the requirements, features and specifications, the architecture, the roadmap, the initial budget, the time tracking method, any kanban board(s), and finally any additional internal documents or collaboration documents detailing additional practices or information specific to the project.

If the project involves a client external to the organization[1], the final artifact needed is a contract between the parties. Ideally, the contract will be split into several parts that include a boilerplate

[1] As opposed to a client that is part of the same organization. Even if this is the case, the client should ideally be at arms-length to the design team. It is important that the client prioritizes the goals of the project and what will create the greatest value for the organization. While possible, this may be challenging if the client is also part of the design team.

template supported by project specific modifications and appendices. Contracts should be appropriate for an agile project methodology, and there are numerous sources describing this in greater detail [7, 10, 44, 54, 86]. Any contract should undergo review by a qualified legal professional. The contract should include a clear description of:

- The development approach
- The scope of project
- The go/no-go decision process and resulting commitments from both parties, especially in the case of termination
- Chargeout and payment details (if necessary)
- IP ownership or licensing

Finally, the team and the client can make an informed go/no-go decision about whether to proceed with the project or to cancel it. Continuing with the project will trigger a transition to the project iteration phase, supported by the set of living artifacts created. A cancellation should the followed by a review and analysis of the reasons leading to this decision.

The development team may benefit from organizing tasks and managing progress using a kanban board suited to the activities and workflow of the project initiation phase. The task list will be fairly generic and core tasks will be applicable across most projects. This task list can be carried between projects, added to, and adapted as appropriate. The key activities of stakeholder collaboration and engagement, planning and knowledge generation within the team, and creation of an initial and minimal draft set of artifacts can be readily captured in a kanban such as the example illustrated in Figure 4.3.

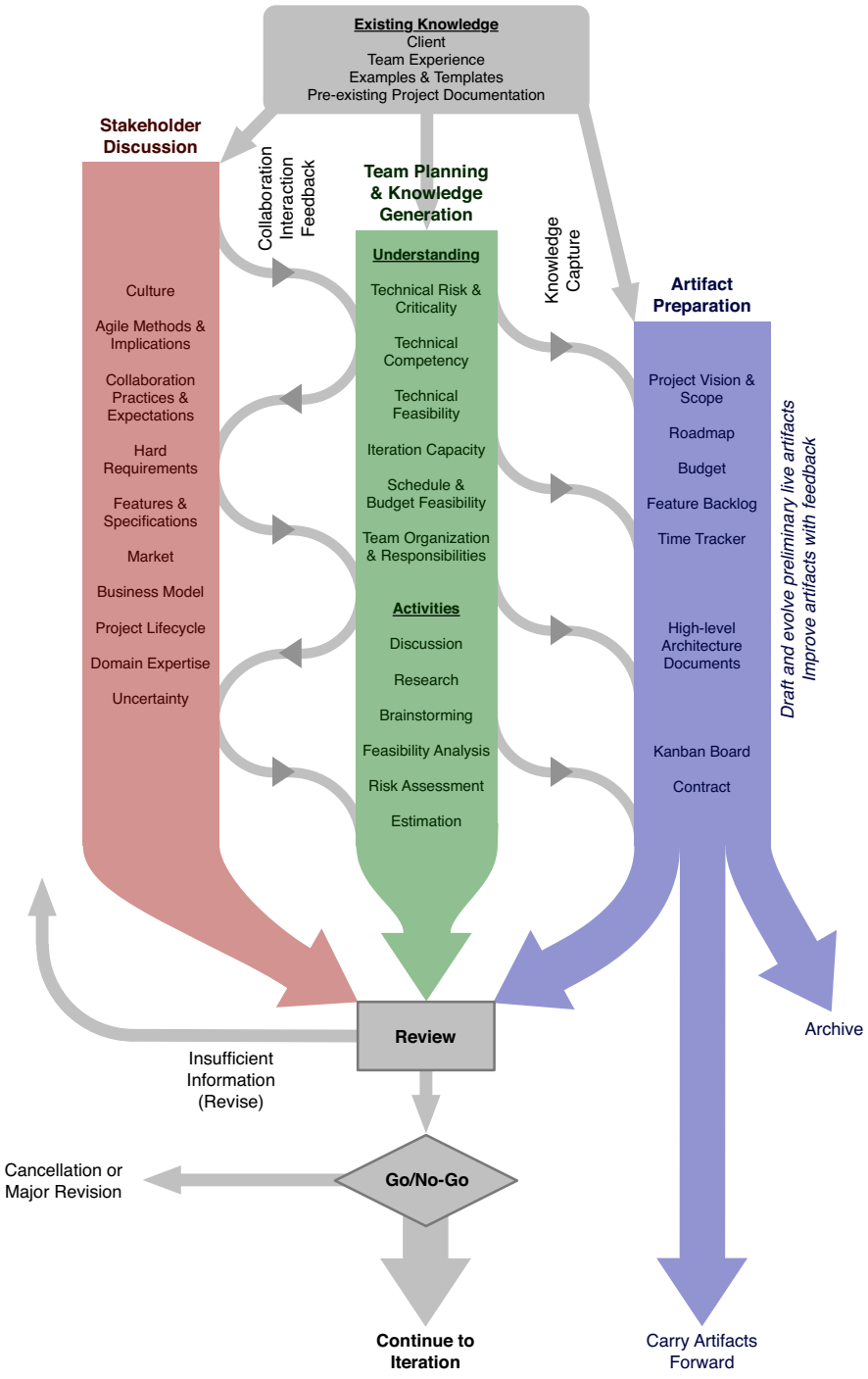


Figure 4.2: Project Initiation phase.

	Planning		Execution		Review	
	Task List	Analysis	Stakeholder Discussion	Document	Team Review	Ready for Final Review
WIP Limit	*None*					*None*
Priority Items						
Normal Items						
Link to policies (transition trigger, acceptance criteria, etc.)						

Figure 4.3: Project Initiation sample kanban board.

4.2 Project Iteration

The main iterative stage of the model is meant to reflect the iterative, feedback-driven nature of design projects with the goal of constant forward progress and delivering value, while maintaining continuous interaction with stakeholders and team members. The model is presented in a way to avoid the impression of doing work over again, but rather building on what was done previously. This may mean scrapping some parts of a design, reworking extensively, and changing course as new information is obtained, but the goal is still to move the entire design forward towards delivering value. The project iteration phase builds on knowledge and live artifacts initially created in project initiation stage, and this phase will see the creation of additional artifacts related to both design and management as project progresses. The iterative model is illustrated in Figure 4.4.

Project iteration is represented by a helix. The vertical axis represents the progression of time. As time progresses, value and effort increase, while risk is reduced. There are two sides to the helix, with one side representing the management, planning, and review activities, and the other side representing design execution and development activities. Live documents improve, grow, and are refined as time progresses. The state of the project should be representable by a horizontal slice across the helix and a snapshot of the artifacts.

Viewing the helix down the time axis illustrates the iterative workflow and management approach with the goal of regular feedback, monitoring and improvement towards goals. This is clear in Figure 4.5. The main activity categories in this circular workflow model are planning, execution, review and go/no-go. A high level view of how information is used and represented in the artifacts of the project is visible. Interaction with the artifacts is not limited to what is suggested in the illustration, though these are the times in the workflow where the artifact is most relevant. There will of course be variations in the quantity, type, importance, and formality of artifacts with each project, and at different times throughout the project. The beginning and end of each workflow cycle is indicated with a go/no-go decision.

It is noted that the half split of the circle between design and management activities is not necessarily

representative of the time spent carrying out each, but rather that they are both of importance. The execution phase may consist of other activities reflective of the specific workflow of a project or task.

The feedback-driven nature of the model is obvious if the workflow cycle is unwrapped to show the time progression and feedback loop analogous to a control system, as presented in Figure 4.6. This analogy was alluded to by Highsmith [44].

4.2.1 Workflow

The detailed workflow overlayed onto the basic plan, execute, review and go/no-go loop as illustrated in Figure 4.5 is a generic, discipline-agnostic example of the main design workflow activities carried out when distributing tasks by feature. The management half of the workflow circle will remain fairly generic across any type of task, while activities in the execution half of the circle will vary considerably depending on the type of task. Some tasks support the project but do not contribute directly to design, while other, larger tasks may span more than a single phase of the cycle. Typical activities are described in detail below. The workflow is reflected directly in the design of a kanban board. An example of a detailed kanban board with multiple workflows is shown in Figure 4.7. This example shows two workflows, one for generic tasks that are relatively simple, and another specifically for design tasks when broken down by feature or subsystem that details some of the activities needed during execution. Ideally, the tasks on the kanban board can be further categorized with a visual aid like colour or symbol to identify additional characteristics like discipline, design phase, task type, assigned team member, time in process, or any other descriptor that helps the team manage the tasks.

The left-most two columns of the kanban board represent the feature backlog. Ideally, this is identical (and ideally live linked) to the artifact used to document requirements, features, tasks and specifications. The second column identifies tasks or features that have been prioritized as most important. Remaining columns represent each subsequent activity that a task or feature

undergoes, with the right side of the kanban facilitating review, approval and potentially release activities. There are many variations of a kanban possible, and the team should adapt the kanban to the project at hand as well as their preferred workflow. Furthermore, the kanban board may need to evolve as a project progresses (though the team should not make drastic changes regularly). One of the strengths of using a kanban board for project management is the easy visualization of progress and task status. A kanban should show tasks that are blocked, differences in priority, type of task, or any other information deemed useful. This goal of visualization to aid communication may be extended to other management artifacts as well. Evaluating workflow effectiveness, project status and project progress can be aided by Kanban-style CFDs or Scrum-style burndown charts. The use of collaborative software tools can provide extensive opportunity to gather information together in the kanban and offer unique methods of visualization and analysis. Examples like Taiga, Trello and Pivotal Tracker can be found in Table A.2 with the tag 'Collab'. Furthermore, software tools are readily applicable to distributed teams. A co-located team may still opt to use a physical kanban board.

Planning

The planning stage of the workflow covers the review, editing and prioritization of the feature backlog or task list. New information is gathered as a project progresses, and this should be used to clarify the requirements regularly. Planning activities also involve detailing existing tasks and potentially breaking them down into subtasks that are of appropriate size and scope. Almost like a work breakdown structure, the overall tasks, features, or subsystems will have smaller stages of work that may need to be split across iterations or across teams or resources. The team should understand uncertainty and risk in tasks and use this information to break them down accordingly. If the primary reason for risk is uncertainty, tasks should be introduced or structured to obtain the understanding needed to reduce this uncertainty. Early in the project, the team may need to start with additional architecture investigation and technology investigation ("spikes" in Scrum) as a way to obtain this understanding.

A key aspect of making a task or feature ready for the execution stage of the workflow is to outline the acceptance criteria or the definition of done. This will be different for different types of tasks and even different tasks of the same type. Ideally, there will be quantifiable metrics or specifications that can be measured against during testing or review, but this is not always possible. Some general acceptance criteria include making sure related documentation is sufficient (architecture up to date, source documents up to date), the requirements or specifications tied to the feature or task have been met, the feature or task has been tested and reviewed appropriately, and the result qualitatively meets goals or provides value. The team will have to determine the most effective way to achieve this. If possible, the acceptance criteria or definition of done should be recorded with the other task attributes as part of the kanban.

Prioritization of features or tasks can be a challenging task, and should incorporate numerous factors. The team should consider the high level goals outlined in the roadmap, the goal of reducing project risk early, and the critical path associated with sequential task dependencies. The value assigned by a client is also important. Tasks of higher risk (especially technical) should be prioritized to reduce overall project risk earlier.

The planning activities will likely benefit from collaboration with the client. The expectations of client involvement should be outlined early in the project, and adapted to meet the goals of both the team and client as the project progresses. This may include regularly scheduled meetings and discussions, or other means of communication. Ideally, the client will contribute tasks and features to the list, review those added by the development team, and provide input on what items represent the greatest value so the team may incorporate this during prioritization.

Adding tasks to the feature backlog is a constant process, and the team should expect to do so at each stage in the workflow. While the planning and review stages will be the primary time when tasks are identified, insight gained during the execute stage may introduce other tasks. If possible, the source of the task should be recorded, which may be the customer, the team, or through review to address a deficiency or defect.

When defining tasks, the team should strive to be feature oriented, with a clear understanding of

the task goals (definition of done or acceptance criteria). Ideally, tasks that last through a single workflow cycle should deliver a prototype, or at least a result that is demonstrable at the end of each task. Tasks should also push towards regular synchronization and iteration between features, subsystems or disciplines, with the goal of preparing for more comprehensive demonstrations and deliveries. These higher level goals are based in part on the roadmap planning aid. Part of delivering a tangible result regularly is to create short term demonstration goals for tasks. While a prototype is not always feasible, it is also possible to demonstrate or review deliverables like CAD models, calculation packages, simulations (mechanical, circuit, or other), interface specifications, schematics, architecture documents, or software modules or features. If a feature needs to be broken into multiple tasks that are not sequential (e.g. tasks of different disciplines), it is better to carry out multiple tasks from a single feature at the same time rather than having team members simply pick a random discipline-specific task. This helps reduce the work in progress when considering tasks grouped by feature. In some cases, a task may need to be broken down by design phase, in which case each task should have a clear, demonstrable deliverable ("show design feature feasibility by calculating these items and comparing against these specifications" rather than "finish calculations").

One additional activity that the team should consider including in their planning is that of task effort estimation. This provides an additional metric that, when combined with time tracking, can help the team understand how accurate their estimates are. The goal is not really to meet the effort estimate (i.e. hours to complete), but rather to improve estimates so a team is better able to predict high level schedules and deliver value to clients by completing a project in the time that is expected. A traditional project management strategy would be to control the project by looking at how closely a team's actual effort (measured) meets the estimate (setpoint). This methodology uses estimates as a guide and time tracking as a feedback mechanism to improve the estimate, though meeting deadlines is also a part of a team's value proposition.

Execution

The execution stage of the workflow is where tasks are carried out and features are implemented in an effort to progress the design project. While there will be a range of task types, this section will focus primarily on design tasks. This methodology outlines a general approach to the design of features or subsystems, consisting of requirements, analysis, architecture, concept or high level design, detailed design, implementation and testing. These activities are not expected to be perfectly sequential, and need to be adapted to be appropriate to the task or feature at hand. Requirements is primarily a part of planning (and project initiation), but is mentioned here for completeness. Analysis generically describes any additional research or other activities needed to create an appropriate design. A key part of the design is architecting the feature or subsystem and understanding how it will integrate with the rest of the system. This helps define bounds that will lead to more successful integration of the new part of the design. The architecture artifacts help to capture the overall system architecture design in a practical way that is communicable across the whole team and across disciplines. While technical details and expertise will vary between disciplines for these activities, the approach will be very similar across all disciplines. In contrast, the strategies for detailed design, implementation and testing will vary drastically depending on what disciplines or technologies are being used. Some examples of activities or tasks for different disciplines are described further below.

Examples of detailed design activities for mechanical aspects include 3D CAD modeling, calculations and simulation. Electrical or electronics subsystems will rely on schematic capture, wiring diagrams calculations and simulations. Software and firmware may use model-based design and code generation, a test-driven development approach, defining APIs, design-by-contract, algorithm simulation, or defining data structures or flow. It is noted that while test-driven development may seem a part of implementation and test (in reverse order), it is reasonable to consider the design of the tests part of the design phase, while writing tests is part of implementation and running tests is part of testing, thus fitting the general design approach model.

Examples of implementation activities for mechanical, electrical and electronics elements may include simulation studies on models, and fabrication of partial prototypes and full prototypes using

a variety of techniques outlined in Section 3.8. Software development activities include writing code (for both the implementation and the tests), generating code, and various activities to prepare for test and release.

Testing approaches will vary, but are all based on the same goal of obtaining data that helps evaluate if the definition of done and the acceptance criteria have been met. The test plan and associated activities should be prepared earlier as part of the requirements, analysis or design activities. Typical electrical or mechanical tests include evaluation of performance and failure modes across a range of environmental conditions aided by various test equipment. Examples of useful test equipment include environment chambers, shaker tables for testing vibration, water and dust ingress test systems, EMI (conducted and radiated) test setups, controllable power supplies and loads, and measurement equipment like multimeters, oscilloscopes and current meters. Software testing may involve unit testing, integration testing, stress testing, or usability testing, for example. A key part of the test activities is demonstrating functionality and results to the customer and the entire team. As a project progresses, there may be more dedicated workflow tasks that support testing with user groups, including from early prototypes through alpha and beta testing. The more a product or system can be tested, the greater the chance of identifying defects before release.

As a project progresses, cross-discipline integration and testing tasks will become of greater importance. This may require more complex test methods and procedures, but should generally follow a similar workflow. For example, testing an integrated basic prototype may mean running software unit tests on an embedded hardware platform while connected to real hardware, mechanics and sensors, and monitoring the test at multiple points across the range of disciplines.

For some projects, prototypes or tests may be large, expensive, or high risk. The risk may arise due to safety concerns, high cost coupled with uncertain outcome or possible loss, difficulty or complexity in fabrication, or high lead times. This is especially the case if trying to test the limits of a design or understand failure modes and the system has significant energy or cost. In these situations, it is prudent to spend more effort in preparing which will lead to less frequent prototypes. Alternatives to full prototypes should also be considered.

As mentioned previously, it is important to deliver a result that can be reviewed at the end of the execution stage of the workflow. Understanding how to evaluate this result should be considered during the planning stage or early in the execution stage, but this may need improvement as a greater understanding of the task at hand is developed through execution. The details of a specific test procedure will not be possible to define without the detailed design. Finally, it may be appropriate to break significant and complex test activities into multiple tasks. For example, one workflow cycle will cover the design and implementation of a feature, another for designing and implementing a test plan and test fixture, and a third to carry out detailed testing on the feature. If this is taken to an extreme level, the full design iteration and workflow will reflect a waterfall approach. While acceptable for some projects (those with extremely high prototyping costs, for example), the design team should favour vertical feature slices that are small enough to carry through the full design process in a small number of workflow cycles.

For some features there is a high likelihood that the workflow will be interrupted due to external dependencies and wait times. This is particularly noticeable during the implementation stage when creating physical prototypes or any other embodiment using outside elements (e.g. server or client test hardware for software). Ordering COTS, getting components manufactured, in-house manufacturing, assembly, and any other similar activities will all incur delays. This is the case even if component sourcing or manufacturing is tied to in-house services (unless the project team is carrying out all parts of these tasks directly). Furthermore, different activities may require different amounts of attention or effort by the team—a simple online order is straightforward, but liaising with a machine shop for custom components may require significant effort. A strategy to mitigate the impact of these delays is to split tasks or features at points where delays are expected. The team members involved can carry out other tasks while the delay occurs, and carry on with the second part of the original task in a new workflow cycle once ready.

As mentioned previously, there will be a range of tasks that do not fit into the design workflow, and will have a workflow specific to the activity. The same overall goal of reviewable results at the end of an execution stage still applies. Examples include cost analysis or extensive research

tasks.

Review

The review phase of the workflow provides opportunity for demonstrations, test data analysis, reviewing the state of the project, updating artifacts to reflect new understanding, and continuous release activities. The formality and effort of reviews will vary drastically depending on the task and the state of the project. Reviews may be internal or involve other stakeholders.

The frequency of reviews that include the client will depend on their availability. Projects with high client commitment may be able to achieve immediate reviews, while a reasonable alternative is to schedule regular review meetings similar to the time-boxed nature of Scrum. Ideally, this will not interfere with the kanban-style pull approach to task completion. Tasks that have been completed and reviewed internally can be queued for customer approval at the next review meeting. Of course, this delays feedback from the customer on such tasks and may then delay any dependent tasks relying on the feedback. As a task result or feature is reviewed, it should be compared against the acceptance criteria. The value of the delivered feature, especially from a client's perspective, should be part of the review. This may be qualitative in nature. If possible, it should be compared to the initial value or priority placed on the task during planning.

A team should carry out internal reviews at the conclusion of each task. Internal reviews contribute heavily knowledge transfer within the team. Internal knowledge transfer may be aided by documentation (either at time of review or as it is generated by team members during execution or other workflow stages). A suitable artifact to document this knowledge should be selected, with suggested methods including a team wiki, or informal how-to documents or application notes. These reviews also provide an opportunity to carry out regular process reviews to adapt the methodology as the project and team evolve. Process reviews may utilize defect or time tracking artifacts. Other artifacts like the project vision and scope and the project roadmap should be reviewed and updated as necessary as well. The client may be involved in these reviews and updates occasionally.

During reviews, it is important to mark down reasons for rejection or changes. Should a task or feature fail to meet acceptance criteria, it can be added back into the feature backlog (likely with the highest priority, but not necessarily). Even if a feature is accepted, there may be new defects or feature requirements uncovered (related or not) that should be added to the feature backlog.

Defects should be added to a defect tracking system artifact. If an issue is discovered during a review of the feature, it is not really considered a defect, as this is the purpose of the review. The feature or task will however be rejected, or passed with an additional task to address the shortcoming added to the feature backlog. Discovery of an issue during use or after acceptance is classified as a defect or bug. This may occur either after release or later in the project after acceptance but before release. A defect indicates that the feature or task deliverable does not meet the requirement or specification, or a fault or error has been uncovered during use. An issue that can be classified as a change request can be identified as when the feature meets the requirements, but the requirements did not properly reflect what the customer wanted, or if the client or team have a new idea of what is wanted now that is different than before. Tracking changes in a similar manner to defects is also encouraged, but the issue should be categorized properly. Defects or changes may also benefit from further categorization of importance or criticality, which will help drive prioritization.

If the project is leveraging a continuous release approach to push changes to users, the end of a review is a good opportunity to carry out relevant release activities. Ideally, this will be automated.

Go/No-Go Decisions

At the end of each workflow cycle, the team and client together should use information about the current state of the project to decide whether to continue forward with the project, whether it should be terminated, or whether it has reached completion by meeting project objectives. This provides regular opportunity to cancel the project if the expected value being returned is not meeting expectations or if risks are not being mitigated appropriately. Typically these decisions will be made after reviews that involve the client, but the team to should at least consider the likelihood of success and continued generation of value after each internal review as well. The

activities associated with either termination or completion are outlined in Section 4.3 below.

Client and Stakeholder Collaboration

Client or stakeholder collaboration is beneficial throughout the workflow, but especially in the later part of the review stage, the go/no-go decisions, and the earlier part of the planning stage. These regions incorporate customer feedback into the evaluation of the project progress and planning the project direction to deliver the greatest value. This weighting of client involvement is illustrated as part of Figure 4.5. A more formal arrangement for customer involvement is important through meetings and approval methods, and should be described as part of the contract (especially regarding go/no-go decisions). It should be made clear to the client (and other stakeholders, if necessary) that rapid feedback will help guide the project more effectively and allow more efficient progress. If the client is committed to collaboration, access to their ideas and feedback even throughout the execution part of the workflow is beneficial, probably on a less formal basis. Note though that effective requirements management and clear goals is still necessary, and can not be entirely be replaced by informal face-to-face discussions on a day-to-day basis.

4.2.2 Handling Multiple Parallel Tasks

It is likely that working to complete only a single task at a time during an iterative workflow cycle is not practical. Based on the size of different tasks and characteristics like expected delays, as well as the size of the team, there are several strategies to manage multiple tasks.

In the first case, the team may complete multiple tasks within a single iteration. This is practical if planning and review overhead is significant relative to the size of each task. There is a high likelihood of this, as otherwise the tasks are probably too large and should be broken down further, or the review and planning activities are too minimal. The tasks may be carried out either sequentially or in parallel (or in some combination), and each may have a different workflow. Parallel execution of multiple tasks in a single iteration is achievable either through true multi-tasking or if there are

multiple team members or resources available. Parallel execution in the context of the helical model is illustrated in Figure 4.8. Sequential execution, either by dependency or as a single resource or team member is handling all tasks, is presented in Figure 4.9. These figures assume each task will have the same effort, which is not necessarily realistic, but the overall approach is still applicable. Care must be taken to avoid spreading team members across too many tasks at once in a single iteration, as this will incur overhead due to multitask context switching [96].

In the second case, the team may wish to draw a single task out beyond a single iteration timespan. This is more reflective of a kanban-style pull approach to task distribution. An example of skipping one review and planning session for a single task is illustrated in Figure 4.10, while fully interleaving tasks is demonstrated in Figure 4.11. The alternative is fixing the time between workflow cycles and always matching reviews (see simple first case above) and filling the available time with a set of tasks based on estimated effort, which is how Scrum approaches the problem. Looking at this second case from a different perspective, the goal is to have faster interleaved internal review and planning sessions on demand as tasks are completed, rather than fitting a set of tasks into a single time-box. This tightens the feedback loop and ensures information does not become outdated. The effort of different tasks may vary significantly for each stage of the workflow. Clearly, some tasks simply take longer to execute than others. Tasks with greater volume, criticality or potential side effects will demand increased effort in planning and review. This is also relative to the shear layer characteristic of the task and rate of change that is possible for the subsystem or feature. Some tasks or features will require rework of large or foundational aspects of a design. Finally, external dependencies impact the progress of a task over time. While one solution is to split such a task into multiple tasks surrounding the expected delay, this may not be desirable, or it may not be possible due to unforeseen delays or dependencies. Interleaved review and planning activities will require more team members than just those who executed the completed task, potentially disrupting the flow of others. Finally, as management activities are interleaved, synchronization will be lost between different tasks or subsystems (and potentially subteams working on them). The team should regularly synchronize the management activities through larger review and planning sessions for several reasons. Most importantly, scheduling constraints around client and stakeholder

collaboration opportunities can be addressed in part through this synchronization. Integration tasks will have dependencies that all need to be completed before the task can be carried out. Review, releases and planning can be aided by having a complete view of the project with no work in progress, which also facilitates archiving or freezing this state. Also, if a team is split into subteams or a similar hierarchy exists, a fully synchronized point in independent workflows improves information exchange across the entire team.

4.2.3 Sharing Tasks Between Multiple Team Members

The possible approaches for sharing tasks between multiple team members is described in detail in Section 3.4. Task planning and flow is described previously, with some additional considerations outlined here. Team members may work on different tasks in parallel as some tasks may not require continuous full effort. If possible, tasks like this should be parallelized to distribute resources efficiently. In addition, some tasks will require multiple resources (again, with varying levels of effort). Examples of tasks that require parallel effort by multiple team members include prototyping or testing activities that are physically easier with more than one person, or where health and safety concerns require multiple personnel (i.e. so team members are not working alone).

A possible strategy for task sharing is passing off tasks at different points in the overall design workflow to specific experts. For example, in a software project, requirements and architecture may be carried out by system architects, development by dedicated programmers, testing by quality assurance personnel, and release activities by an IT team. It is important to be wary of building a team composed of specialists, as task sharing becomes far more difficult to manage, with every task passing sequentially through each team member. While specialists may be an important part of the team (like those with specific prototyping skills, for example), this model avoids catering to a team of specialists.

4.2.4 Iteration in the Model

The task-level workflow approach described above is supported by other, higher-level workflows, like that outlined by the project roadmap. Workflows help define where iteration occurs, especially from a project management standpoint. Key iteration rates, as mentioned above in Section 4.1.6, include the client review rate, the rate of creating prototypes, the release rate for projects using continuous release, retrospective process reviews, major design/test/integration milestones, high level project planning and review sessions, and finally the entire project time period. In the simplest case, the levels of iteration in the model are driven by internal reviews, reviews that include the client or other stakeholders, high level roadmap planning and goals, and the overall project in the context of the product lifecycle. There may be additional levels as the team or project grows and incorporates other levels of organization, especially hierarchical in nature, though the scalability of this model may be tested. The iteration rate for each level can be different, and may not be constant through the project. Furthermore, iteration rate may be different across different disciplines or subsystems within a project, even at the same level, to reflect the differences in change capacity (as modeled by the shearing layers). While the iteration rate may vary especially with the strong use of a pull approach, it is likely to show an average magnitude in the long term if the team creates tasks of similar effort throughout the project.

Iteration rate targets should be developed during project initiation based on team experience and project characteristics. Most important is to match the commitment of the client to being involved in reviews. Iteration rates do not need to be fixed and represent a guideline, but the team should target an approximate rate of feedback. Ideally, fast iteration rates will be used with customers reviewing regularly and even integrating into the team. Fixed iteration rates to synchronize with customer availability or other external dependencies or requirements will likely force slower rates due to the overhead of planning and organization (aided by the roadmap).

4.2.5 High-level Planning with a Roadmap

A roadmap guides the high-level progress of a project, and describes the overall iteration approach in the design process. It is similar to a traditional schedule but should evolve with the project and provide an aid for iterative planning activities instead of being used to evaluate project success. Furthermore, a roadmap does not need to be tied to specific dates, though linking a roadmap to calendar dates may be helpful or necessary. Some estimation of time or completion dates during project initiation is useful for process feedback as well. The roadmap should show details for near-term activities and milestones, while detail will be minimal on longer time scales. As a project progresses, the roadmap should be updated regularly during planning activities to keep the near term sufficiently detailed.

The project may focus on a phase-based progressive design iteration approach or a feature-based incremental approach, or balance each to reflect project characteristics. The design process may also be varied across different disciplines or shearing layers. With any project that involves greater cost or lead times during prototype development, there may be a greater preference for phase-based design at the roadmap level. Early in the project, tasks will focus on requirements, analysis and architecture, while detailed design and virtual implementations will dominate mid-project, and physical implementations and testing are likely to be found near the end of a project. While this is not encouraged due to lack of early first-level feedback, it may be a method used to mitigate certain types of risk (especially if the cost to build a safe prototype early is high). The roadmap will indicate trends in the long term design process approach.

An example implementation of a roadmap artifact for a project focused on incremental iteration is shown in Figure 4.12. Note that the variation in line style between the date and the description helps to illustrate a greater confidence in the order and date of near term tasks. The continuous release milestone type is also greyed out, as an example of a project that will not release prototypes at this stage in the lifecycle. A roadmap for a progressive iteration approach will see variation in design-oriented milestones, replacing the prototypes and major design milestones with completed project phases instead (e.g. architecture tasks complete, detailed design tasks complete, single

prototype implementation complete, prototype testing complete). This vertical list schedule is one way to visualize a project roadmap. Others include calendars, horizontal timelines, or matrices divided by feature, subsystem, subteam or discipline. This roadmap sample is also easily mapped to Gantt chart, PERT chart or a calendar schedule if desired. The goal remains that the roadmap is simple and easy to understand and modify, provide visualization of planning and progress, and to be used as guidance, not as an absolute goal.

The roadmap is first created during the project initiation stage and is updated throughout the project to reflect the current state of progress and planning. Evolving priorities, requirements, features or overall strategy will cause a roadmap to change over the course of the project. It is expected that the near term plan have greater detail than the long term plan. One goal of the roadmap is to provide initial planning about when major review or synchronization events will occur, especially if a client or stakeholder availability is limited, or if specific deadlines must be met. Synchronized collaboration between different sub-teams or stakeholders is described in Chapter 3, Section 3.4.3. These and other key milestones of the project should be included in a roadmap. Examples of milestones include the completion of major testing or integration tasks for subsystems or the whole system, preparation or completion of external test campaigns, or demonstrations. Milestones typically correlate to the review and go/no-go portions of the management side of the workflow, where reaching the milestone helps the team evaluate progress and plan the direction for the project on a higher, more complete level. The roadmap helps the team understand critical path and drive prioritization in the feature backlog during the planning stage of the workflow as the team works to achieve milestones and synchronize efficiently. If possible, the roadmap should try to provide some indication and visualisation of the key iteration rates that are greater than the main workflow and internal review rate. These iteration rates are described above in Section 4.1.6 and Section 4.2.4.

The vertical time axis on the helix model can almost be written out as a timeline matching the roadmap. The period between the go/no-go points illustrates the iteration rates that should be decided and can match a number of iteration cycle levels or scales. This is illustrated in Figure

4.13. It is noted that the helix and timeline do not need to match or be linear, which would lead to a very complex diagram.

4.2.6 Developing Multiple Design Concepts

A standard technique used in design is to develop multiple concepts for the solution to a problem [19, 25, 29, 43, 35]. These are compared based on a set of relevant criteria to select the best option that will be carried forward through the rest of the design process. In a traditional design methodology, it is common for several complete solution concepts to be prepared early and only to the level of detail necessary to make a selection, and this selection then moves into a detailed design phase, with most methodologies describing concept design as a single stage in the overall development of the project [29, 25, 19]. In the context of a mechatronics design project that values agile, modular design and an incremental approach to the design process, this definition of concept development constrains the design. Instead, this methodology takes the view that pursuing multiple concepts may be applicable at any level and cover any range within the project scope or design space, drawing on the feature-based concept development approach described by Haik and Shahin [43]. Concept development strategies ranging from task scale to project scale are described below. A different strategy may be applied across different parts of the same design, especially when divided by subsystem or feature. It is part of the responsibility of the team to evaluate when working on multiple concepts is appropriate, and the amount of effort to spend on each concept. Carrying out parallel design activities increases overall project effort, and thus should be carried only as far as a clear decision can be made as to which alternate design will be selected to continue with.

At the smallest scale, there may be several possible solutions considered by a single team member during execution of a single task. This approach develops, evaluates and selects a concept all within a single workflow iteration prior to review. It is suggested that the team review the concept selection decision together during the review stage. At this scale, development of multiple concepts may be planned or unplanned as part of the task, and is not considered part of multiple tasks. The concepts help formalize design tradeoff decisions within the current task. An alternative to

having a single team member carry out concept development is for two or more team members to collaborate together informally during the planning or execution stage of the workflow to brainstorm and develop the concepts.

At the next work scope level, the team may carry out several concept investigations in parallel as separate tasks with the goal of making a decision on which concept to adopt by the next (synchronized) review. If the concept has been developed completely in the single iteration, it can be integrated, otherwise the team should take the selected concept through remaining design phases by adding a new task to the feature backlog during review. Larger concepts or shorter iteration cycles will likely only see each concept carried through an architecture or embodiment design phase, with detailed design, implementation and test carried out as a later task. For smaller tasks or longer iteration cycles, there is a better chance each solution can be carried further, perhaps through the whole design process, for a more direct comparison during review.

At the next level of scale or scope, the team can carry multiple concepts through multiple tasks or iteration cycles until appropriate decisions can be made and the design space is more thoroughly explored. Ideally, the team should try to reach a conclusion earlier as carrying multiple designs forward is expensive. The greatest value is delivered if the concepts are only taken as far as required to make a good decision. This balance can be a challenge to achieve, though a team will likely improve with experience. This scope is the closest to the traditional concept evaluation approach described above.

Carrying multiple concepts through a larger portion of the design process can be extended by assigning concepts to multiple sub-teams. Each sub-team can approach the problem either with communication or independently. This is approaching set-based design methods [83, 82].

Overall, concept development requires some flexibility and planning to understand what the goal is and an appropriate scale or scope to achieve the goal. It is important to understand how to compare different concepts, preferably before beginning investigation, so that efforts can be focused on work that will contribute to the evaluation and decision making. Three approaches to concept evaluation include feasibility based on experience and technology availability, absolute comparison

against known requirements and specifications, and relative comparison between different concepts [43]. Relative comparison can leverage a range of objective tools like a decision matrix or Pugh's evaluation matrix. Some subjective evaluation may also be beneficial, incorporating the experience and preferences of the client of the design team. In some cases, a concept may be taken to the prototype stage in order for enough information to be available to make a clear decision. Challenges in understanding the requirements and specifications before a concept of the feature is available can be addressed by iterating to develop an appropriate level of detail, or developing a concept prior to full requirements and specifications [35]. The process and order of activities will vary to match the project.

4.2.7 Understanding Artifacts

Information flows into and out of artifacts at different stages in the workflow and the project. While some updates and use is expected across the whole workflow, it is expected each artifact will play a larger role at different times. For example, the feature backlog will be added to during the review and early planning stages, edited or improved later in the planning stage, and referenced during the execution stage. The artifacts are not meant to be fixed, and should be continuously changed and updated to reflect a snapshot of the current state of the design and the project. However, most artifacts are only able to capture explicit knowledge, and other methods of collaboration and knowledge must support them.

It is suggested that most artifacts be live documents or use a software tool with archivable data. The team must continuously update live documents, and freeze or archive the artifacts at key milestones to save an instance of the design and the project. Source files or documents (some of which may be considered detailed design artifacts) should also be subject to regular archiving, matching key milestones. These source files and documents should also be archived more frequently using a version control system. If possible, archiving activities should be made as automatic as possible. Ideally this can be extended to automatically version the documents as well. Artifact versioning is only necessary when the artifact is archived. The live source document is the real and most up to date

"version". These live, unversioned documents should not be distributed outside of the project team. Even software source or any other type of design artifact should be archived and a version assigned before being released to any external stakeholder. Ideally, the team should integrate document control and authentication practices into the archive workflow that meet APEGA guidelines where appropriate [91]. Finally, all live and archived documents should be backed up regularly regardless of format. Backups should cover all possible content related to the artifacts, and potentially the systems, tools, applications and configurations used for the artifacts as well. Furthermore, backups should be tested to ensure integrity. If possible, backups should should be stored in multiple physical locations. The effort to create backups is minimal compared to the consequence of losing the information and having to recreate it (if this is even possible).

The primary artifacts that should be used in this methodology model are summarized below in Table 4.1. This list was partially inspired by Carryer et. al. [19]. The team should adapt artifacts and add additional ones as needed. For example, there may be a need for additional documents for internal use or to be shared with the client that detail collaboration practices specific to the project. Additional artifacts may support other retrospective activities, like documents to record areas of improvement. Some artifacts may be created from templates, which can be prepared based on previous projects and adapted to the current project.

Table 4.1: Primary methodology artifacts.

Name	Content	Tools or Format	Created	Included in contract?*	Details
Contract	Legal agreement boilerplate, project specific	Document	Late Initiation	Yes	Section 4.1.7
Project Vision and Scope	Main goals of project, lifecycle boundaries, definition of done	Document	Early Initiation	Yes	Section 4.1.1
Roadmap	High level sequence and schedule, key synchronization info	Various, Spreadsheet	Early Initiation	Yes	Section 4.2.5
Budget	Cost estimates and actuals for time and materials	Spreadsheet	Early Initiation	Yes	Section 4.1.1
Feature Backlog	Task list, features, specifications, priority, acceptance criteria	Software or Spreadsheet	Early Initiation	Maybe	Section 4.1.1
Kanban Board(s)	Reflects workflows and has tasks, priorities, policies	Software or Physical	Late Initiation	No	Figure 4.7
Time Tracking Artifact	Team member time on project, categorized	Software or Spreadsheet	Early Initiation	No	Section 3.7.1
Defect Tracking Artifact	Issues discovered or changes requested after review	Software or Spreadsheet	Early Iteration	No	Section 3.7.1
Architecture Documents	Boundary diagrams, block diagrams, flowcharts, etc.	Diagrams, Drawings	Mid Initiation	Maybe	Section 4.1.5
Design, Implement, & Test Artifacts	Detailed design, implementation and test source documents and artifacts with content specific to project, discipline, etc.	Various	Through-out Iteration	No	Task, Feature, Discipline Specific
Internal Process Artifacts (optional)	Procedures, instructions, process descriptions that help a team implement and adapt their methodology	Various	Through-out Project	No	Team and Methodol-ogy Specific

*Only an initial version is included in the contract, expected that artifacts will evolve with the project.

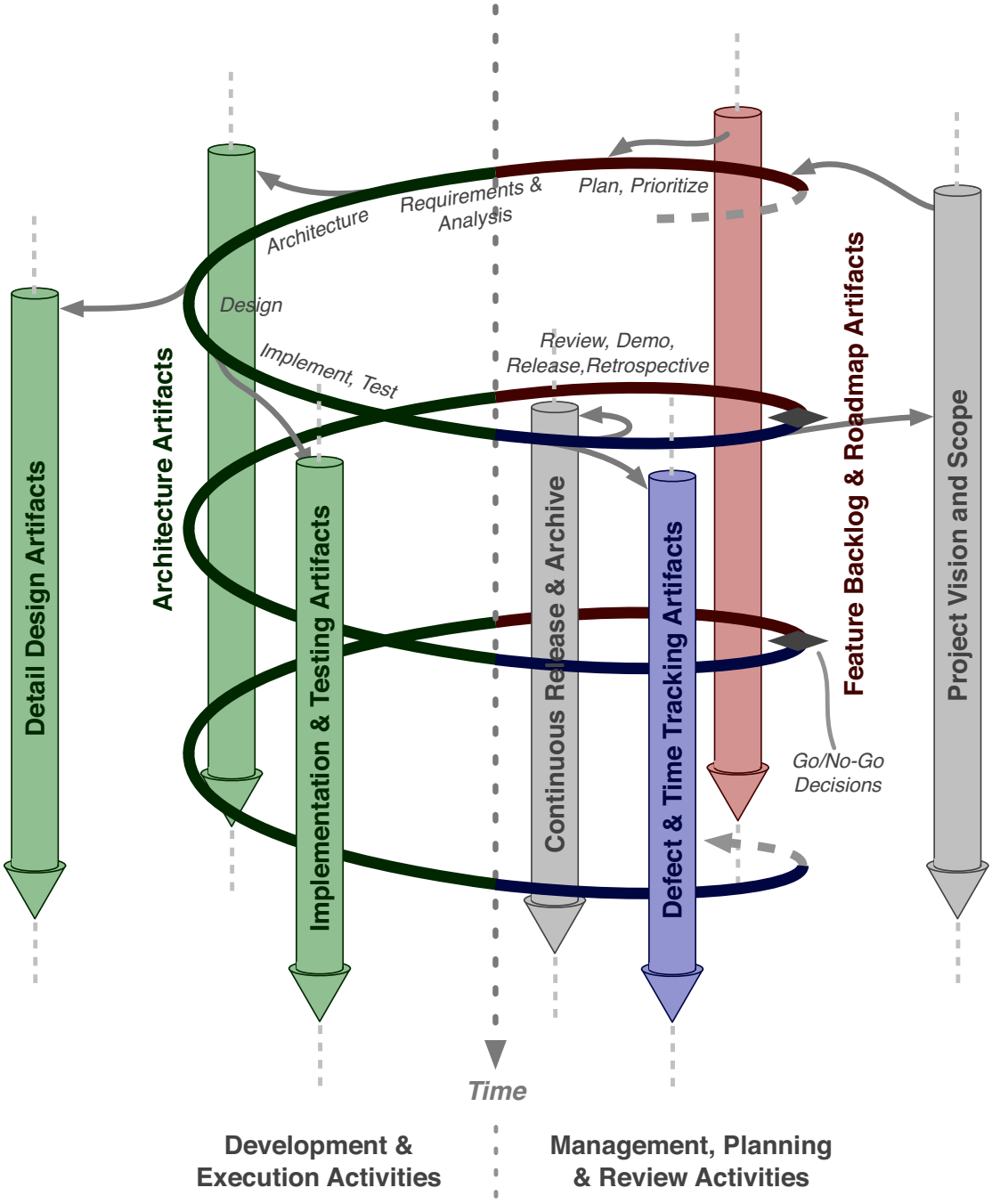


Figure 4.4: Project Iteration as a basic helical model.

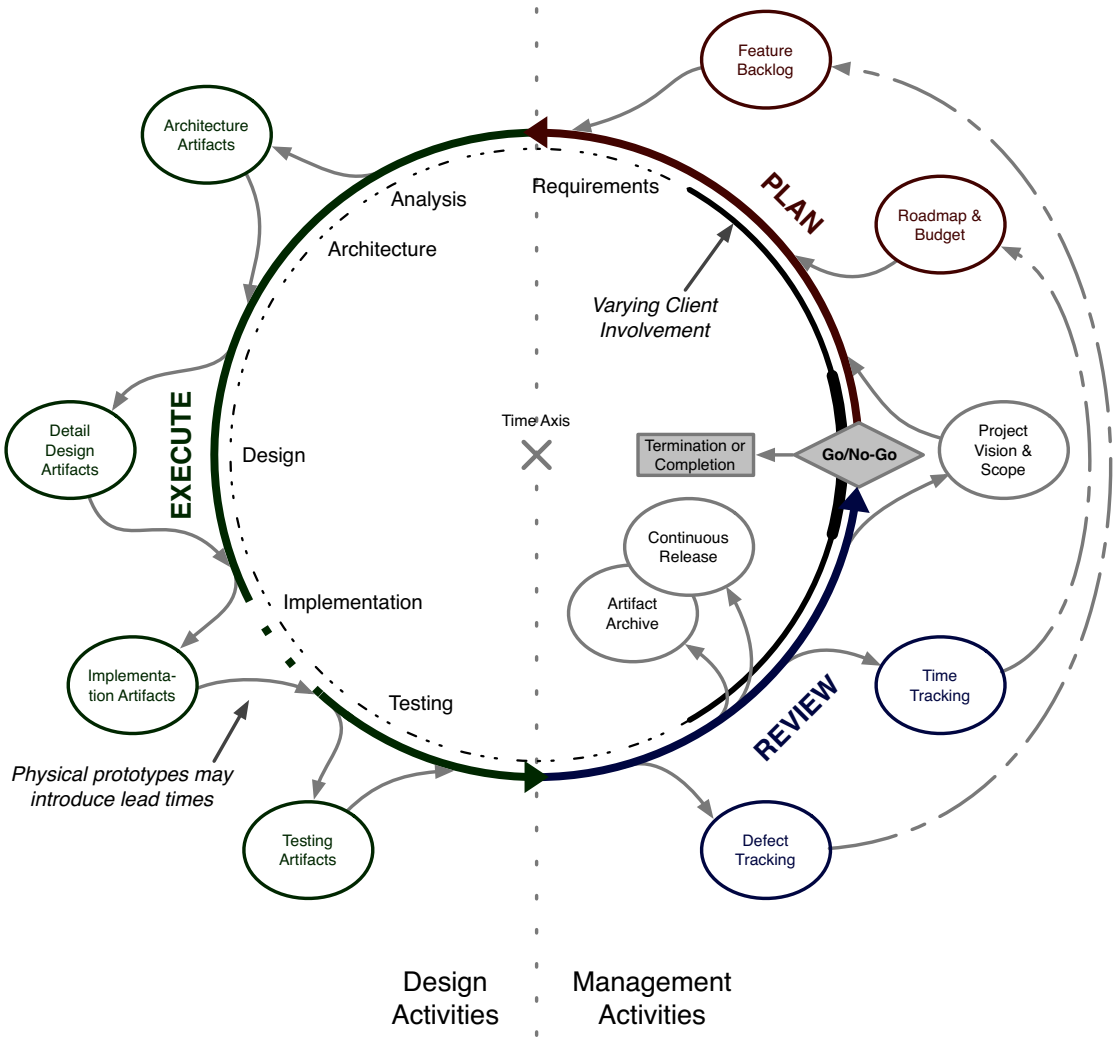


Figure 4.5: Project Iteration iterative workflow and helix in-axis view.

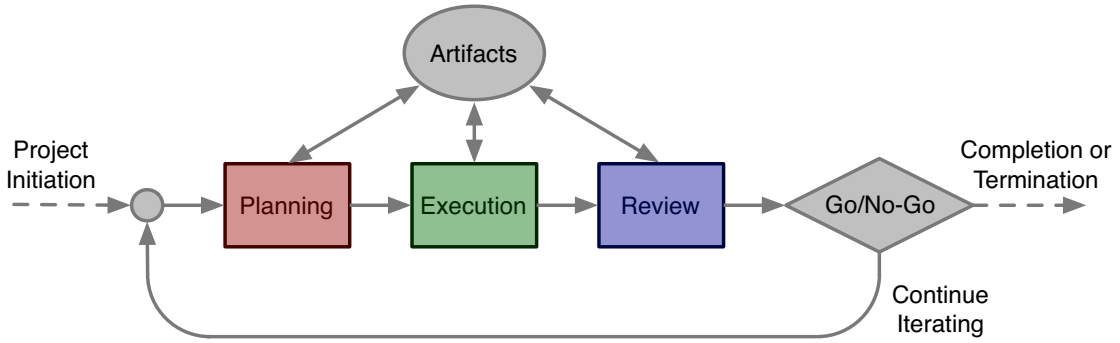


Figure 4.6: Iterative workflow modeled as a feedback loop.

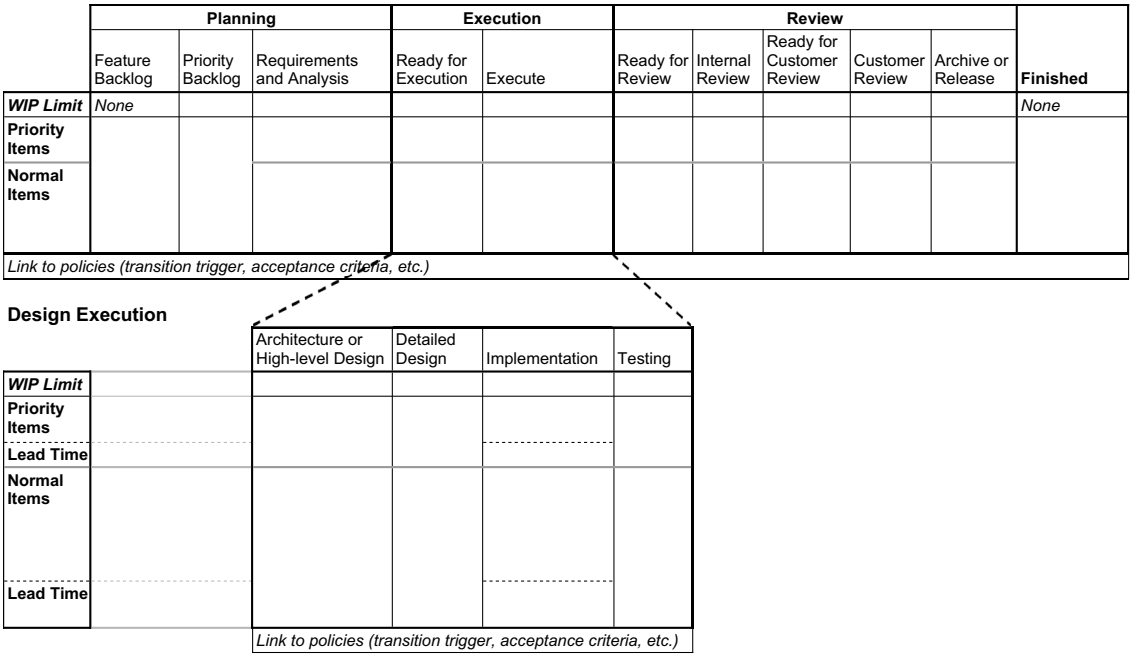


Figure 4.7: Sample kanban showing multiple workflows for Project Iteration phase management.

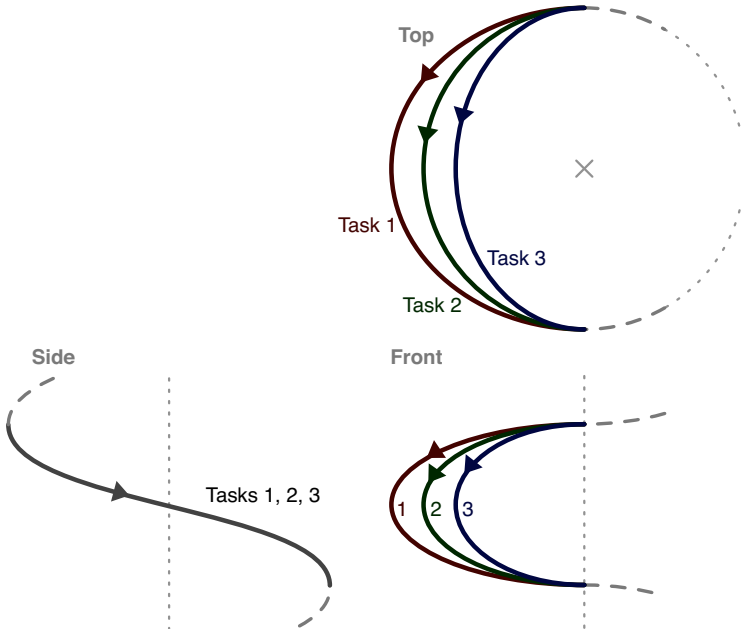


Figure 4.8: Parallel execution of multiple tasks in helical model.

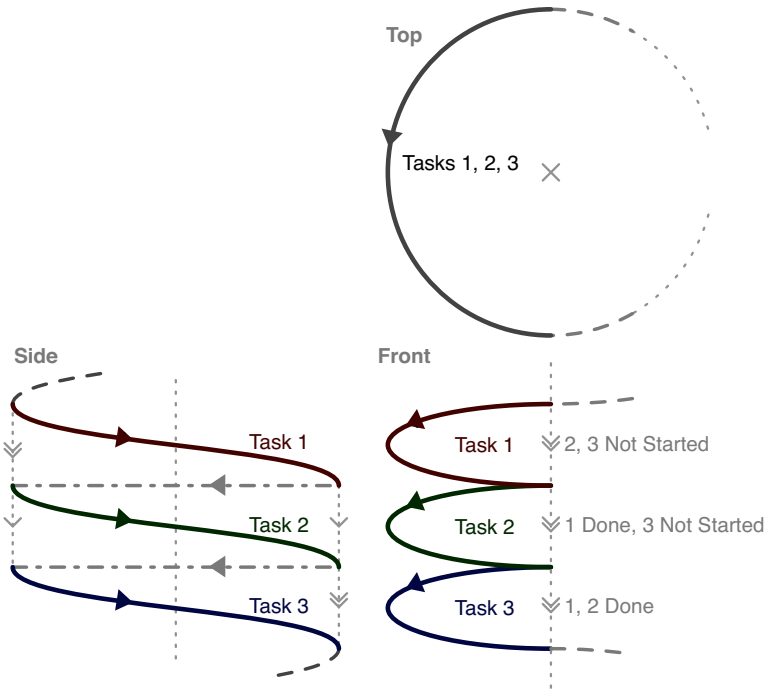


Figure 4.9: Sequential execution of multiple tasks in helical model.

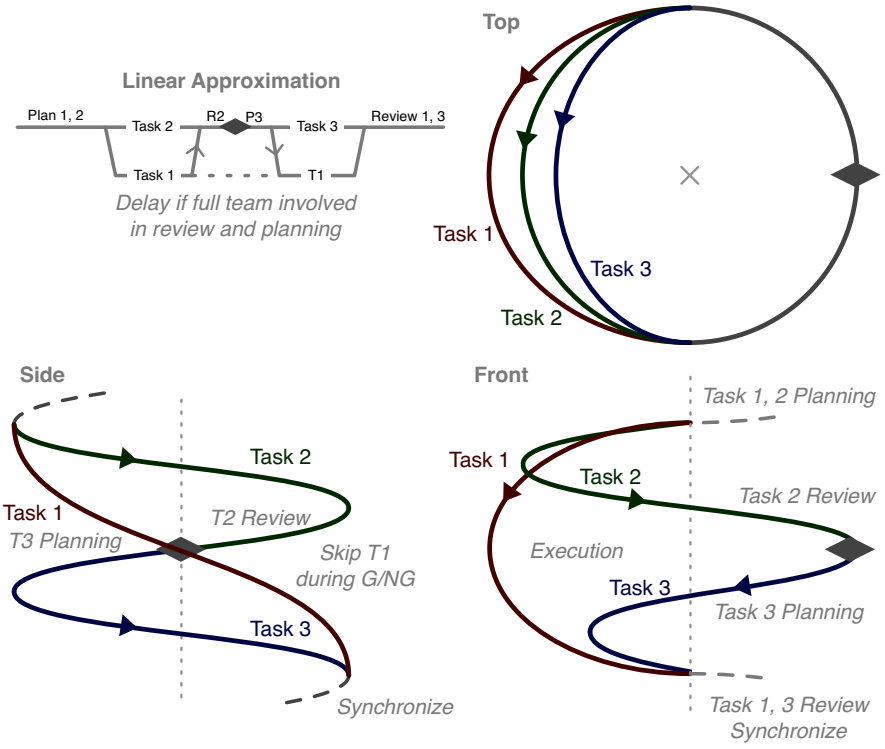


Figure 4.10: Execution of a task across multiple iterations by skipping a review and planning session.

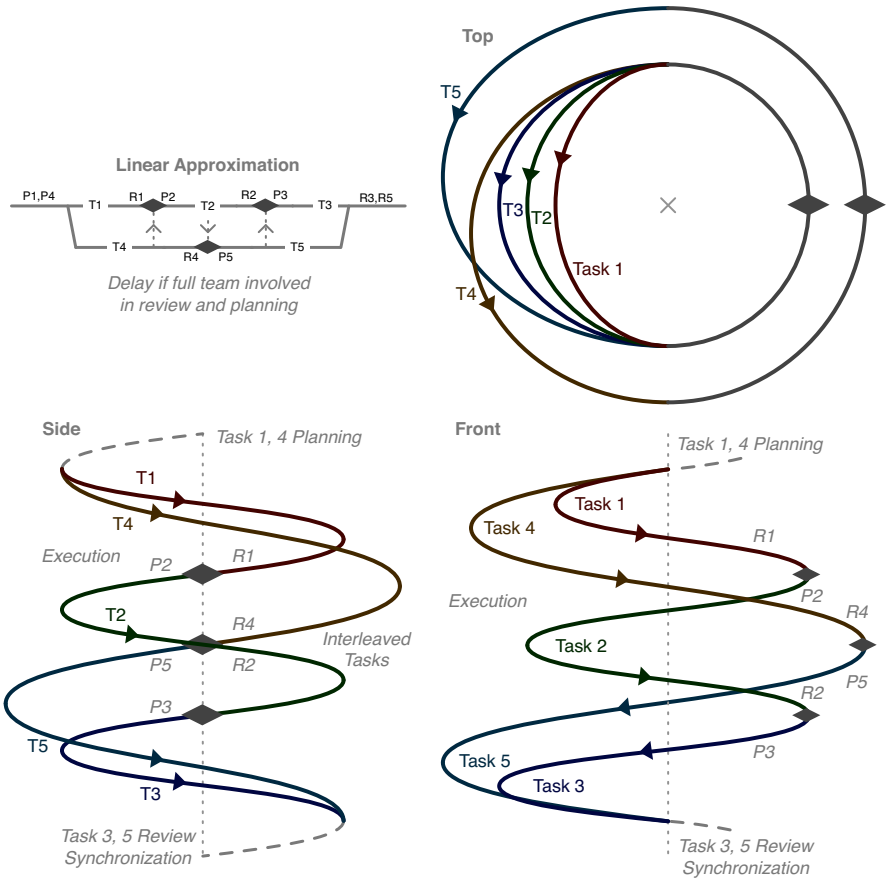


Figure 4.11: Execution of tasks across multiple iterations by interleaving review and planning sessions.

Date	Description	Type						
		External Deadline	High Level Planning	Major Design Milestones	Retrospective	Continuous Release	Prototypes	Client Review
Jan 1	Project Iteration Start							
Jan 8	Review and Go/No-Go with Client							●
Jan 13	First prototype of high risk subsystem A						●	
Jan 14	Simulation of high risk subsystem B						●	
Jan 15	Review and Go/No-Go with Client							●
Jan 15	Retrospective process review meeting with team				●			
Jan 25	First prototype of subsystem C						●	
Jan 29	Simulation of subsystem D						●	
Jan 29	Review and Go/No-Go with Client							●
Feb 5	First prototype of subsystem B						●	
Feb 5	Review and Go/No-Go with Client							●
Feb 12	First prototype of subsystem D						●	
Feb 19	Core system integration demo with A, B, C			●			●	
Feb 19	Review and Go/No-Go with Client							●
Feb19	Retrospective process review meeting with team				●			
Feb 20	Formal scope and strategy review meeting with client		●					●
Mar 5	System integration demo with A, B, C, D			●			●	
Mar 8	Customer demonstration campaign start	●						
Mar 22	Customer demonstration campaign end	●						
Mar 24	Demonstration campaign analysis review			●				●

Figure 4.12: Sample of a roadmap for a project with an incremental design iteration strategy.

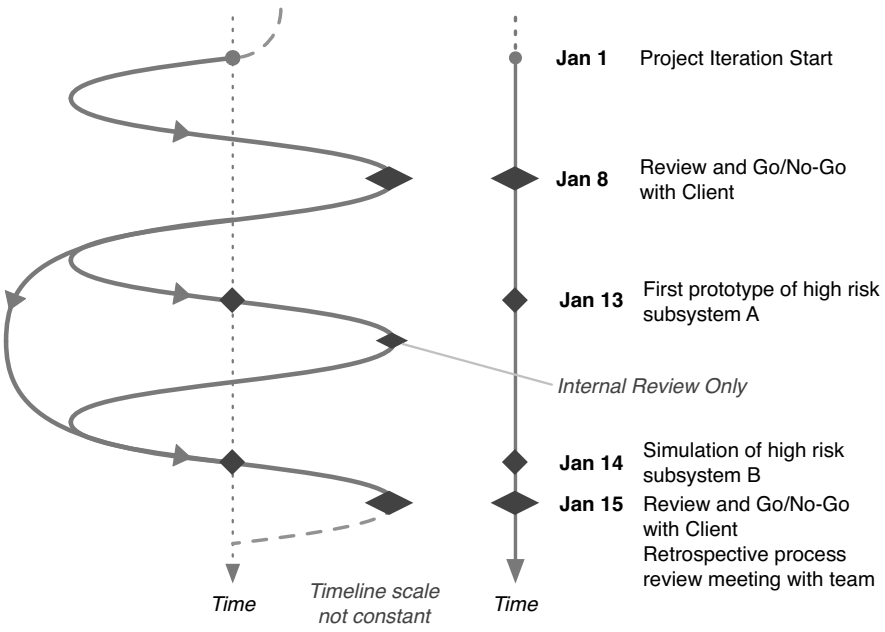


Figure 4.13: Matching the helix timeline with a roadmap.

4.3 Project Conclusion

The project conclusion phase is the end of the current project. The decision to shift to the conclusion phase from the iteration phase is made during the review and go/no-go decision stages of the iteration workflow. There are two reasons for reaching project conclusion, either completion or termination. Completion indicates that the project has been completed up to the definition of done and it meets the acceptance criteria, as approved by the client. Termination occurs when it is deemed that the project is not achieving the value goals of the client or is no longer feasible from a technical or economic standpoint (i.e. it is predicted the value goals will not be achieved in the future). Regular go/no-go decision points ensure that the project stays on track and that the client and the team have ample opportunity to cancel the project if necessary to avoid wasting resources. The project conclusion will have multiple approaches and a focus on different activities depending on the client, the stage of lifecycle of the product, the level of success, and responsibilities of the team and organization.

Project termination may occur at any time throughout the progress of the project at a go/no-go decision point in the iteration cycle. The team should carry out any activities to meet commitments and finish the project cleanly. Such commitments will typically include the delivery of design documentation or the product to date, cleaned up and finished in an appropriate but minimal fashion. There must be some time and resources planned at the outset of the project to cover any such commitments. The specific commitments for the case of project termination should be outlined in the contract if there is an outside client. If possible, the team and organization, and perhaps the client as well, should carry out a post-mortem analysis to learn why the project may have failed. All parties involved have an opportunity to learn from the termination and complete the feedback loop at the project level in an effort to perform better next time. It is noted that not all terminations are true failures. The knowledge and experience developed during the project may be valuable for other projects.

Project completion may include delivery of the design, product or system to the client, followed

by an end of any direct involvement of the project team. This is typical of a consulting type project, that is likely to end abruptly. In this case, the team should arrange the necessary feedback channels and interfaces to understand the performance of the design through the remainder of its lifecycle. This may be achieved by communicating with the client to understand how successful the product or system is both technically and by other measures. This should occur at different timescales matched to other parts of the project lifecycle (for example, through manufacturing, testing, release, maturity, retirement, disposal, or any other combination). Another way to achieve feedback is through technical feedback features built into the design (described as part of agile design principles in Section 3.6. Examples include performance or fault logs for debugging that are collected later or transmitted with a "phone home" feature in connected devices. The team must put in place the infrastructure to handle these features and monitor the data. Lessons learned through these feedback methods can be incorporated into the team's knowledge base for use in the future.

In some cases, project completion will see both the project and the team transition together to a new phase of the product lifecycle. The project conclusion phase can be viewed as a project transition phase at this point. It is still important to create a detailed and complete archive of the project state at this point, even if most of the artifacts will evolve as the project continues. Ideally, the team will move to a short, new project initiation phase and treat the next stage of the lifecycle as a new project. This new initiation will be short because almost all knowledge gathering is already finished. The team primarily needs to focus on understanding the vision of the new project and updating the scope, roadmap, budget estimate, contract, or any other artifacts to reflect the new goals. It is likely the same main iterative structure, team organization and collaborative practices can be used. The team and client can rely on an established level of trust and understanding.

Another possibility is that a completed project is transitioned to a new phase of the product lifecycle handled by a new team. While similar to the delivery on completion option above, it is characterised by greater collaboration between the current team and the new team. Shifting the project to a new team may be necessary if each team has unique skill sets that improve their ability to carry

out work at different stages in the product lifecycle. The new team will need to carry out the next project initiation phase, and the old team should be heavily involved to effectively transfer knowledge to ensure the new team has all relevant information. It may be beneficial to treat the new team as a client or user for some of the documentation and review tasks. Ideally, the new team that a project is passed off to will be identified early in the project so they may be included as stakeholders when appropriate to obtain relevant feedback. Collaborating with the new team early helps when defining the appropriate level of detail or formality required in artifacts that will be passed on. If the two teams can expect additional collaboration after the transition, this will reduce the need for very detailed artifacts capturing explicit knowledge. If feasible, keeping the old team accessible to help the new team clarify their understanding will make the transition easier and require less effort in creating documentation. In fact, the same principle applies to the delivery on completion case above, which can also benefit from continued collaboration (i.e. support).

Regardless of where a project will progress to after the conclusion phase, the team should strive to close the final and largest feedback loop of the project. This involves reviewing the whole process to understand strengths and weaknesses, and outlining how the methodology may be improved in future projects. The use of technical feedback channels for design feedback described above are also applicable to any project that is completed. Completion of any project must also be supported by archiving and delivering documentation or other design artifacts like prototypes. While most of the time this will be a part of the task or feature backlog and completed prior to the conclusion phase, there may be internal use artifacts that need proper treatment. Finally, it is important to make sure processes, tools, or procedures—anything that directly supports the project and the creation or editing of design artifacts—are also archived should the project need to be revisited in the future, which is highly likely. Some examples include compilers, test procedures or configurations, CAD software packages, data dumps from online applications, custom tools or software that were created. Consideration must be made for software tools that rely on a subscription model. It may also be prudent to archive hard drive disk images or the entire development workstation altogether. When setting up and selecting tools, the team needs to think ahead to this archival process. The archive package needs to include at least enough to have all the information to recreate both the design

and a new set of tools if needed.

4.4 Summary

This methodology model implements a lean and agile approach applicable to mechatronics design projects. Practices and elements inspired by Kanban, Scrumban and Scrum are used at lower levels for rapid feedback on a daily and weekly scale. At higher levels, methods are suggested that consider the challenges of physical or capital intensive products, systems and prototypes. The three stage model is illustrated in a simplified form in Figure 4.14. This model reflects the goal of continuous progress feedback and improvement in an iterative fashion, while also including the initiation and conclusion stages which have different requirements, goals and activities.

The first stage is project initiation which involves gathering the information about the project characteristics, goals and requirements, while considering the client, the team, the organization and the project itself. Understanding if the project is exploratory is one key project characteristic to consider, while others include criticality, culture, technical risk, hard requirements and the market and corresponding business model. Activities during this stage support stakeholder collaboration, team-driven knowledge generation and planning, and development of live artifacts to used throughout the project for management and design. By focusing effort on the project initiation stage and increasing the formality and detail in different artifacts, a team can shift the methodology to address projects that have greater optimization and criticality characteristics.

Project iteration focuses on carrying out the circular workflow based on planning, execution, review and go/no-go decisions. Execution activities support design, while planning and review activities and go/no-go decisions integrate management into the process. Typical design activities include requirements, analysis, architecture development, detailed design, implementation and testing. Low-level, rapid iteration is supported by using a Kanban board, while high-level planning leverages a roadmap artifact. Regular reviews, go/no-go decisions and planning activities help ensure the project continues to deliver value and stay focused on goals throughout, while also providing op-

portunity to terminate the project if this is not happening. Using a feature backlog to plan and organize work by feature or subsystems helps manage task size to be achievable in a short iteration cycle as well as distribution to team members. Risk management approaches, an understanding of client value and critical path dependencies help drive task prioritization.

Project conclusion occurs upon completion or termination and may also involve transition to another phase of the product lifecycle. Completion indicates the project has met its goals as measured by the acceptance criteria or definition of done. The team should deliver and archive any design artifacts and as well as archive any internal artifacts like tools or procedures that are needed to revisit the project. Finally, the team must close the largest feedback loop, reviewing the process and the design, and incorporate feedback from the client and the design itself to understand the success and value of the project.

This methodology model demands continuous collaboration and regular communication throughout the project. Documentation can support this, but can certainly not replace it. While collaboration is most important within the project team, it should also be practiced with clients or other stakeholders. The rate of feedback and communication should be highest for those working more closely together, which is the tightest iteration and knowledge transfer loop. Ideally, the team will leverage practices like face-to-face discussions, both formally through regular meetings like daily or on demand stand-ups, as well as informally through constant collaboration to support coupled or dependent work. The team must still respect the need for independent, uninterrupted work sessions to aid concentration and productivity. Distributed teams may need to rely on effective tools to replace being co-located, though occasional meetings in the same physical space are recommended. Tools may include video and audio calls as well as text-based chat. Maintaining open communication channels constantly may create a virtual open office if the team enjoys this work environment. Screen-sharing and video or virtual whiteboards to share sketches and demonstrate work may also help communicate ideas. A number of sample collaboration and communication tools can be found in Table A.2 with the tag ‘Collab’.

In conclusion, the team should be sure to create an environment that supports regular review and

adaptation across multiple time scales, both in the design and in management. It can not be expected that the initial roadmap, project vision, feature backlog, kanbans, or any other artifact, to be correct and work work well throughout the project, especially if there is uncertainty. Nor will the design deliver the greatest value in a single attempt. The team must carry out design and process reviews regularly and use metrics like time and defect tracking to help improve the methodology and practices. This includes improving the tools, documents or other artifacts used by the team to support the process.

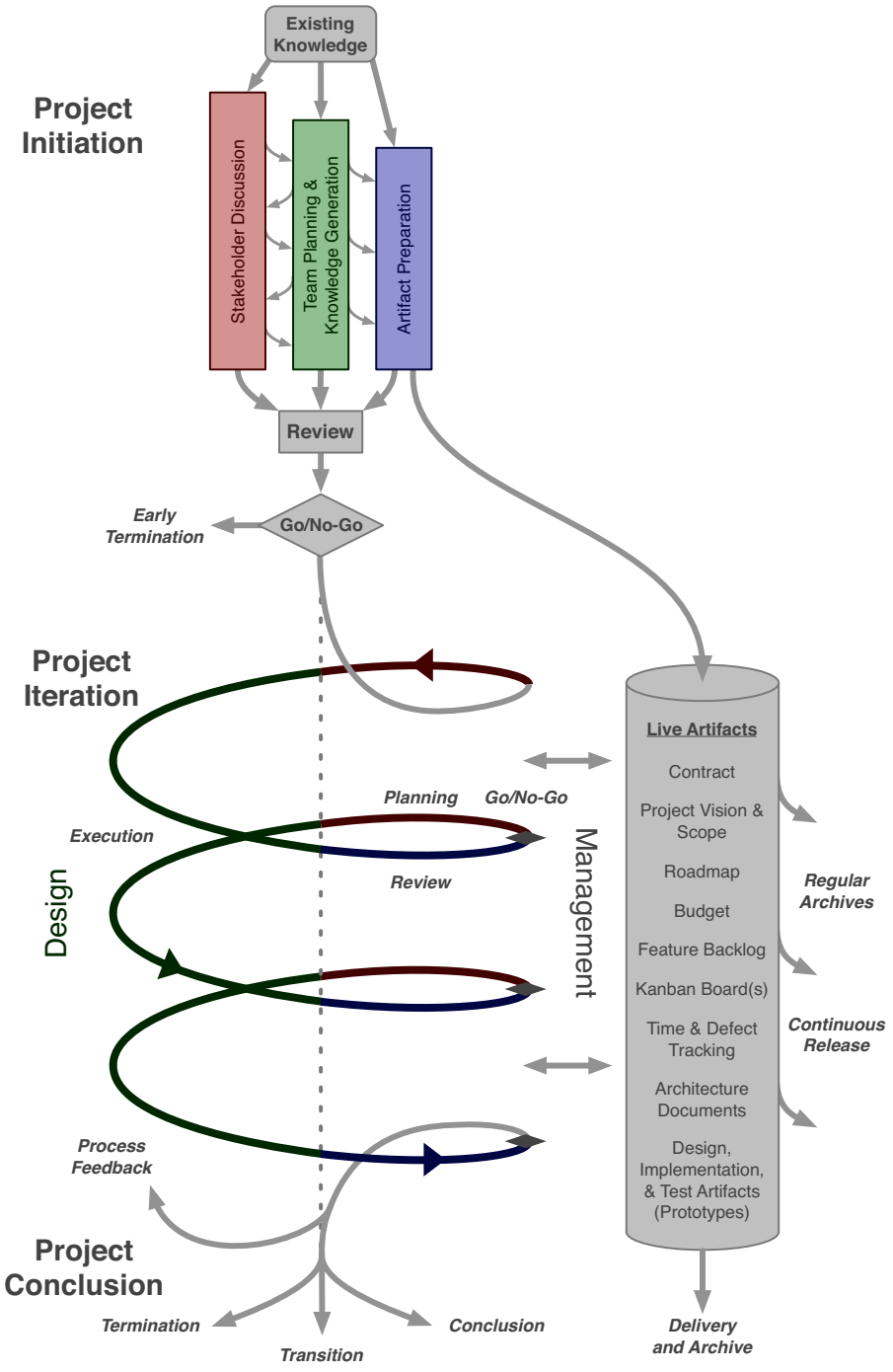


Figure 4.14: Full, simplified three phase helical model of methodology.

Chapter 5

Case Studies

The author has had experience with a variety of different mechatronics projects as part of graduate studies, through the work of Copperstone Technologies (CST), as well as part of undergraduate work in the past. These projects had a range of characteristics, especially with regards to size and the content of different mechatronics disciplines or domains. Some of the products may not be considered full mechatronic systems, but may have been embedded systems or any other combination of mechatronic disciplines. The degree of success of the projects varied, both from a purely technical standpoint as well as value delivery or project management. Together, these projects have inspired this investigation into an alternative and agile design project framework and methodology.

The project team of each case study was similar, and consisted of two to four team members with varying degrees of experience and a range of skills across the disciplines of mechatronics. The size and composition of the team necessitated that almost all activities were carried out collaboratively, and most were concurrent. While some members naturally championed specific types of activities based on previous experience and preference, each member was at least partially involved with all activities. Most team members had previous experience working together, with three having trained as mechanical engineers. The overall culture of the project teams aligned well with agile principles, as the members strived to move quickly, valued prototyping, testing and feedback, and

became easily frustrated with fixed documentation, contracts, estimates or other artifacts that are resistant to change. The teams were driven to innovate, provide value, and work tightly with close collaboration. Challenges included maintaining discipline in a range of areas throughout the projects, and keeping focused on delivering value.

The following case studies describe the technical aspects of the project, outline the methods or approach used in the project, and identify areas of each that were successful or were of concern. The context of the previously described framework and methodology is considered when possible. The goal is to identify information similar to that in a post-mortem review of a project as would be carried out by the team as part of the project conclusion phase. From this, it is reasonable to understand what aspects of the framework and methodology may have increased the success of the projects. It is noted that these projects were undertaken prior to the development of the framework and methodology model.

5.1 Case Study 1: Connected Fleet Monitoring System

A fleet monitoring system connected to a web backend over a cellular network was developed. This product monitors location and temporal usage of vehicles with the goal of improving maintenance processes. Users access the system over a web application. Users can view fleet status information, configure parameters on a per-device basis, and export reports for archive or additional analysis. Data stored in the web backend offers opportunities for future value-added data analysis. An overview of the system is illustrated in Figure 5.1, and a screenshot of the web application is shown in Figure 5.2. The system developed may be categorized as an embedded system instead of a full mechatronics system, as there were no moving mechanical elements in the design.

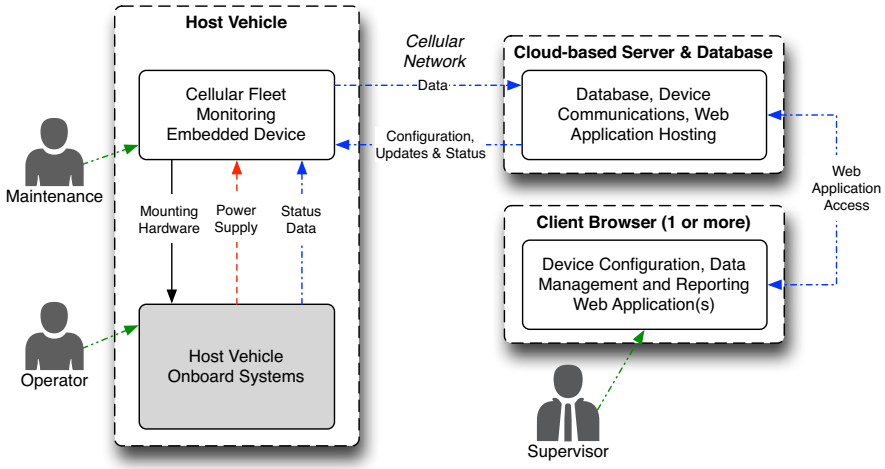


Figure 5.1: Connected fleet monitoring system overview block diagram. (Courtesy of Copperstone Technologies Ltd.)

5.1.1 Project Description

The project involved the early part of the product lifecycle, and was primarily concerned with new product development. The client had previously conducted market research and understood the basic vision and feature set to be included in the product. The project covered lifecycle activities from initial proof-of-concept development up to the delivery of a production prototype and production test run. At the conclusion of the project, the team transferred the design to the client and to the manufacturer in preparation for regular production activities. It was expected that the team would have some future involvement through technical support, maintenance and improvements through bug fixes and feature additions. The business model in use by the client was generally a subscription service, though the team was not aware of the details, including whether the hardware units would be provided free of charge, leased or sold to end users. The market for the product was industrial or commercial end users, and large fleet sizes (and order sizes) were expected. The client was reluctant to share specific market information, but had indicated an expected demand on the order of thousands of units. The client specified several hard requirements in the form of deadlines involving product demonstrations to client company management and potential end users

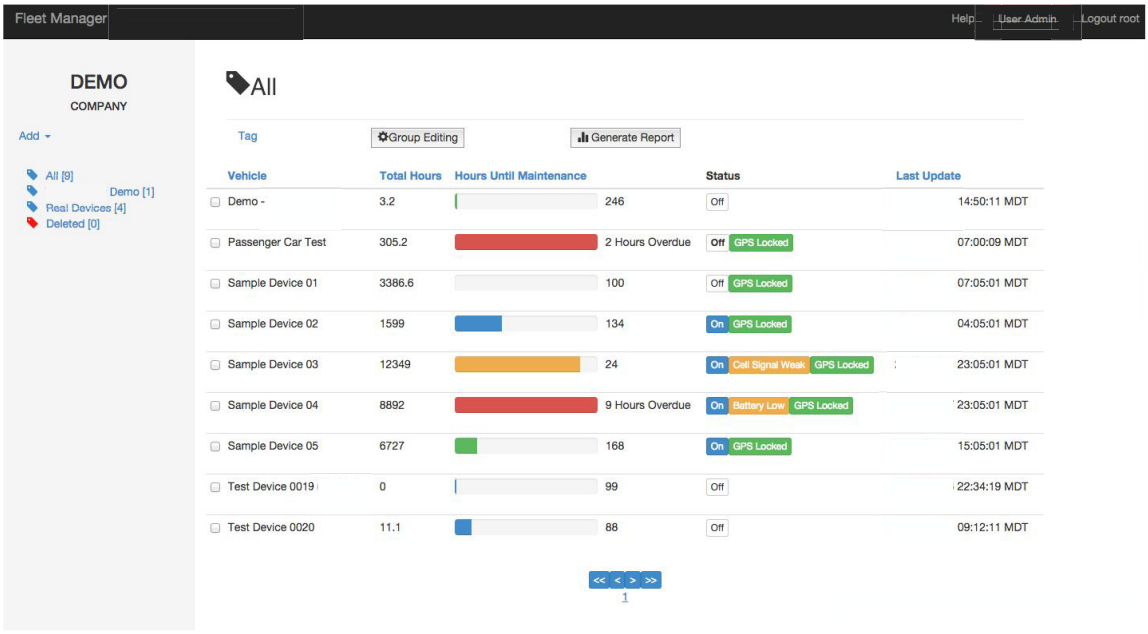


Figure 5.2: Connected fleet monitoring system web application screenshot. (Courtesy of Copperstone Technologies Ltd.)

at industry tradeshows. Other hard requirements included obtaining both FCC certification and UL certification.

The client culture was generally open to a more agile approach to development. The client was interested in receiving regular status updates (short updates weekly or bi-weekly, often by email) and maintaining open communication channels, but had limited time to contribute technically or provide significant amounts of feedback. The client was involved in the business aspects of the project, including contract negotiations with the contract manufacturer and the cellular network operators. Overall, the client was willing to work with the team and adapt to practices that were effective for all parties. The client was billed hourly for non-recurring engineering (NRE) as well as for some capital development costs. The hourly billing was accepted up until the total reached a fixed cost estimate that had been outlined during project initiation. After this, budget overage approvals were negotiated, with the team's organization absorbing some of these costs.

There were several electronics hardware subsystems included in the design. The hardware architec-

ture is shown in Figure 5.3. A microcontroller was central to device operation, and was supported by non-volatile storage to carry data over sleep cycles and power loss events. A set of protected digital inputs allowed the microcontroller to sense vehicle events such as ignition status. Communication with the web backend was achieved with a commercially available OEM cellular modem module that also incorporated a GPS receiver. A major portion of the electronics design was the power supply subsystem. The requirements indicated a wide DC input voltage range (almost tenfold) with the potential for very large transients outside of the normal input range. The power supply needed to provide high pulsing currents to the cellular modem. Furthermore, the overall system needed to have a minimal power draw when asleep to reduce drain on vehicle batteries while also being able to ride through power loss events with minimal data loss. Other requirements included operating in a wide temperature range on a moving vehicle, easy and fast device installation on vehicles, a compact footprint for wider vehicle compatibility, and considerations for two different radio frequency devices. Finally, the design needed to be efficient to manufacture and assemble.

The project had minimal mechanical design elements. The main subsystem was an enclosure and mounting solution that was sufficiently water- and dust-resistant for industrial environments.

The microcontroller in the device ran firmware developed in the C programming language. The main functionality included sensing inputs, measuring time, tracking location, reacting to events, storing event data and measured time, and communicating with the web server to report events, data and receive configuration changes. Other activities included handling sleep and low power cycles and managing over-the-air firmware updates. The overall firmware architecture utilized a super-loop design running on a bare-metal configuration without a real-time operating system. This was supported by a number of software modules, which were mostly divided by functionality. One example is that a library for interfacing with the cellular modem and integrated GPS receiver was needed. Another module handled the communication protocol with the web server. Others handled digital input, non-volatile storage access, and power supply monitoring and control. Challenges in the design included precise cumulative time tracking, handling power failures gracefully, and recovering from data corruption or failures during storage and server communications. The device

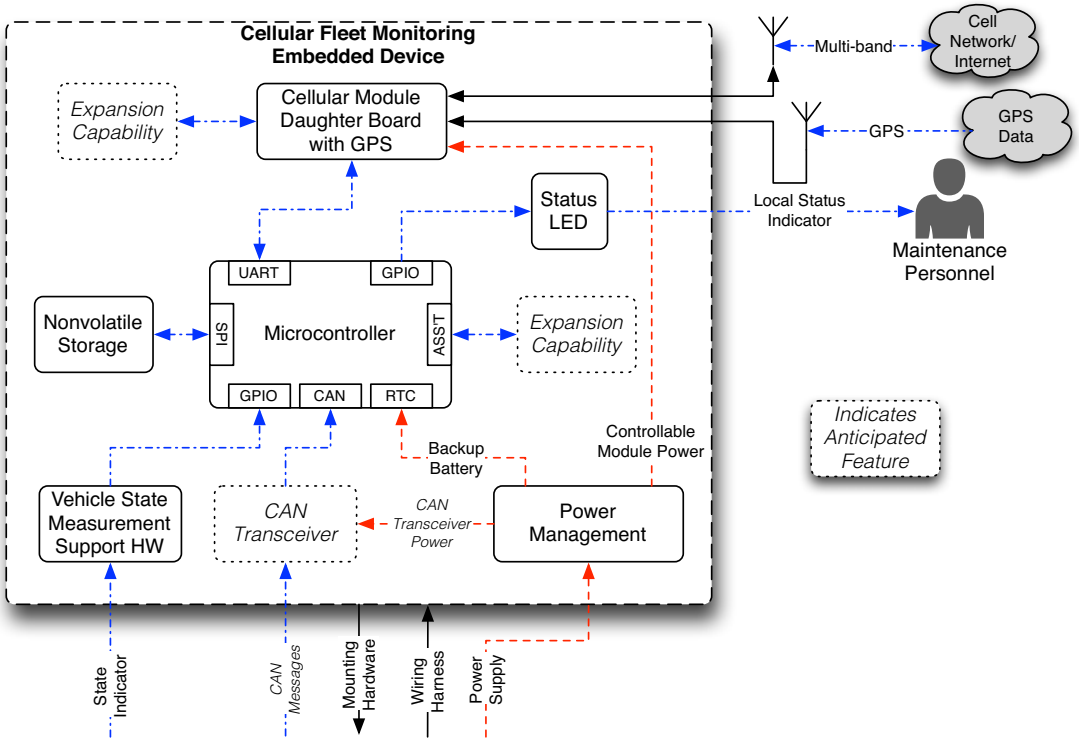


Figure 5.3: Connected fleet monitoring system embedded hardware architecture. (Courtesy of Copperstone Technologies Ltd.)

was designed to be installed in vehicles and operated with no additional physical intervention. This meant that the firmware update system needed to be reliable and fault-tolerant. It is noted that technician access remained possible throughout the lifetime of the device for maintenance, but this would easily become very time-consuming and expensive for a large number of devices.

There were a number of software elements that composed the web server database backend, device communications channels, web application frontend for users, and access to features for administration and configuration. Most of these elements were built with widely available tools, hosting services, frameworks or protocols. Considerations were made for security and maintaining separation of data between different end users. While basic device configuration was accessible to end users, the client maintained control of firmware updates and device provisioning.

The final project tasks included user documentation and installation instructions, as well as the design and documentation of a test jig and procedure for manufacturing. A comprehensive test suite was created and all test data was stored in a database in anticipation of future quality assurance or failure analysis tasks. The design of the test jig and suite included mechanical design, electronics design, firmware development, and software development, and required significant effort.

The project team leveraged existing knowledge and experience numerous places throughout the project. In particular, the client provided several examples of their existing products. This allowed insight into wiring and installation methods that had worked with the client's existing customers, which was then incorporated into the new product. The client was also able to provide a reasonable vision and set of requirements based on existing market research.

5.1.2 Project Methods

Initially, the project team attempted to apply a modified waterfall approach to the design process, but this quickly shifted to several prototype-driven iterations. First, requirements and specifications were gathered. Next, a very basic prototype was developed demonstrating technical feasibility, using the same core microcontroller and cellular modem that were carried through the rest of the project. The server backend was prototyped using a simple software stack. The high level architecture and preliminary firmware source code were carried into the first prototype development. A detailed design phase saw the development of a full prototype, though software and firmware features were minimized. Customer demonstration and feedback after final demonstration indicated improvements were required. At this point, the project team was changed, and a more experienced software developer was recruited. Using feedback from the customer and from testing the first prototype, a new prototype was developed. The two finished prototypes can be compared in Figure 5.4. The architecture remained similar, but some subsystems were redesigned, and the software stack was completely changed. Firmware continued to be built on the initial source code foundation. Finally, the prototype underwent more significant testing and minor revisions, and was deemed ready for production. This led to the development of manufacturing test methods, end-user

documentation, and additional server administration tools. It is noted that the original intent of the waterfall approach broke down quickly with the realization requirements were insufficient early in the design process, and that multiple prototypes would be required. As much of the technical content was relatively new to the project team, multiple prototypes improved opprotunities for learning from feedback. This naturally showed multiple high level iterations in the design.

Figure 5.4: Connected fleet monitoring system prototype A (right) and B (left). (Courtesy of Copperstone Technologies Ltd.)

The project was typically divided by domain, with each team member specializing in discipline specific skills. The team was composed of three mechanical engineers with some experience across other mechatronics disciplines, along with a single software developer with formal computer science training. The software developer focused on the server software. One team member focused primarily on communications protocol and associated test and simulation. Another worked on firmware

while also contributing to hardware design and test, backend architecture, communications protocol and the manufacturing test system. The final team member primarily worked on the hardware design and testing, while contributing to backend development and championing the manufacturing test system.

From a project management standpoint, the project team used a fairly ad-hoc approach. Supporting documents were very minimal, and the process was very informal. There was limited iteration in the process, though the low level project process resembled a pull method with a balance between feature and discipline based task distribution. However, it was not well implemented. Tasks or features were not broken down to small units, and team members would often work several weeks on the same task with very little interaction or feedback. Task selection and prioritization was carried out primarily as informal joint team decisions with minimal analysis. Very basic insight and values from the client were incorporated into the process.

Collaboration was achieved with a number of tools and practices. The team met with the client only intermittently, as the team and client were geographically separated. Phone calls and email were used for updates and queries. The prototype was presented to the client only during formal demonstrations where the goal was to show a finished product. This reduced the frequency of customer feedback. Within the geographically dispersed team, collaboration was achieved with software tools like Google Drive, Google Chat, Google Hangouts, and email (for details, see Table A.2). A limited number of face-to-face meetings were held for higher level planning and review, but these were of limited formality. The "virtual office" tools used allowed for regular (sometimes daily) stand-up style status meetings, though again these were informal and lacking vision. There was clear intention to separate project status update from design discussion during meetings. Design discussions tended to occur on-demand, and typically included only those parties closest to the subsystems involved. While the ad-hoc nature of project management and collaboration worked due to a highly effective team that had experience working together, key decisions could have benefitted from greater formality.

Project retrospectives were carried out very infrequently, typically only at the end of a prototyping

cycle. These retrospectives were not formally denoted, but did allow the team to improve some aspects of workflow and tool use.

Early in the project, the team attempted to formally capture specifications with quantitative metrics in a spreadsheet document through discussion with the client and by team brainstorming. It was discovered that many requirements were challenging to quantify and full of uncertainty and ambiguity. While the core set of quantifiable metrics were useful through the project, the team abandoned this approach and instead used an ad-hoc approach to add or modify tasks, features and specifications as the project progressed. To support this agile-like approach, an online software tool called Pivotal Tracker was used (for details, see Table A.2).

The team carried out basic effort, schedule and budget estimation as part of project initiation. This was included as part of the contract, along with a reasonably detailed scope description including deliverables. The definition of done for the project, however, was vague, which made it difficult to pinpoint exactly when the project was complete.

The team held design reviews of the hardware. This was easier for the team as it was more akin to previous engineering design review experience. Reviews were carried out within the team, and a structured test plan was developed and executed. Many of the quantifiable specifications were related to electrical performance, which allowed the team to clearly understand if the design met requirements. Such clear specifications and an understanding of test and review strategies were not available for the software or firmware portions of the project, which led to poor test and review coverage. This was partially due to limited experience in these fields, which also drove the decision to use a third party to review the hardware design and firmware source code late in the project.

Overall, the project vaguely followed the methodology model described previously in Chapter 4. The team's previous experience with formal design projects had only been with a waterfall model. The early requirements gathering and project scope activities resembled a subset of the project initiation described in this methodology. It was clear this did not fit the project, and the team quickly shifted away, but ended up with a very adhoc approach. The tendency towards more iterative and feedback driven methods, while very informal and unstructured, can be compared

to the project iteration stage of this model. It was clear the team continued to plan, execute and review tasks until the project was complete, even if the criteria for completion were not well defined. The project conclusion did involve a final set of deliverable commitments along with an attempt to support a transition to the next phase of the product lifecycle. It was expected the team would transition to a support and maintenance role with far less involvement.

Early in the project, a set of detailed documents covering scope, requirements and architecture were developed, but the team lacked the discipline needed to maintain and update these artifacts. Detailed design documents were the primary artifacts carried through later in the project. The hardware design was captured using EDA source files, simulation files, and spreadsheets and documents recording test procedures and results. Firmware and software design artifacts were primarily restricted to source code, with a limited number of notes or code comments supporting the code. Some attempts were made to create more formal documentation (for example, to describe the communications protocol), but these were rarely updated to reflect the current state of the project. A wiki was used to track internal techniques or procedures for tasks like configuring development environments, setting up tools or maintaining databases. The team was successful at tracking time per person and per major subsystem using the online tool Toggl, and this tracked time was used for billing the client (for details, see Table A.2). Ideally, this tool would have been set up to allow a more detailed analysis during project retrospectives. For example, time entries could have been tagged and categorized in a standardized way to search and sort entries more effectively. Basic accounting and budgeting activities were completed, as these were needed for billing and estimation (especially early in the project and at the end).

Very little explicit risk management was carried out during the project. The team did naturally focus design and testing efforts on areas of high probability of failure or high severity of failure. Examples include the power supply input protection, timebase accuracy, firmware updates and server communication failures.

At the system level, prototyping and testing helped the project progress. Several prototypes were created throughout the project but overall, the team tended towards a limited number of prototypes.

The earliest prototype leveraged commercially available development boards for the cellular modem module as well as the selected microcontroller, as pictured in Figure 5.5. Along with solderable prototype boards with some support components (like non-volatile storage), this was very useful for early system architecture proof of concept and initial firmware development. The next prototype used a custom printed circuit board which allowed testing and further firmware development. Based on testing and client feedback of the first custom prototype, the next prototype was a significant overhaul with many changes. Of interest was the use of simulation during the design of the power supply subsystem, which was demonstrated to be beneficial when this subsystem worked properly in the prototype without fixes. The team was also able to prototype and carry out very basic simulations and tests on the communications protocol and server backend features that handled this. This was beneficial for both firmware testing and communications protocol validation. The team also tested the systems in vehicle analogs, as well as in real vehicles and operating conditions. Laboratory test setups were used when testing the power supply subsystem and when measuring and improving the performance of the precision time keeping features.

There were several examples of good agile design techniques incorporated in this project. As mentioned above, multiple prototypes of the hardware were created. In the design of these prototypes, an attempt was made to separate and modularize the subsystems. This was especially true of the power supply elements. In the custom PCB prototypes, different sections of the power supply were isolated with jumpers, which allowed completely independent testing with a controlled supply and load. This also improved the reusability of different parts of the power supply. The design included sufficient power supply feedback through analog voltage measurement points and digital power good status indicators linked to the microcontroller, which allowed for the option of automatic tests or real-time monitoring features in the firmware. Extra microcontroller inputs and outputs were made accessible to aid in debugging and development, including serial streaming to a standalone datalogger. Rapid prototyping techniques were not really leveraged beyond the use of simulation for the power supply or commercial development boards for the microcontroller and cellular modem. The requirements, documentation and testing of the electronics hardware was more comprehensive, useful and appropriate when compared to other parts of the project. The

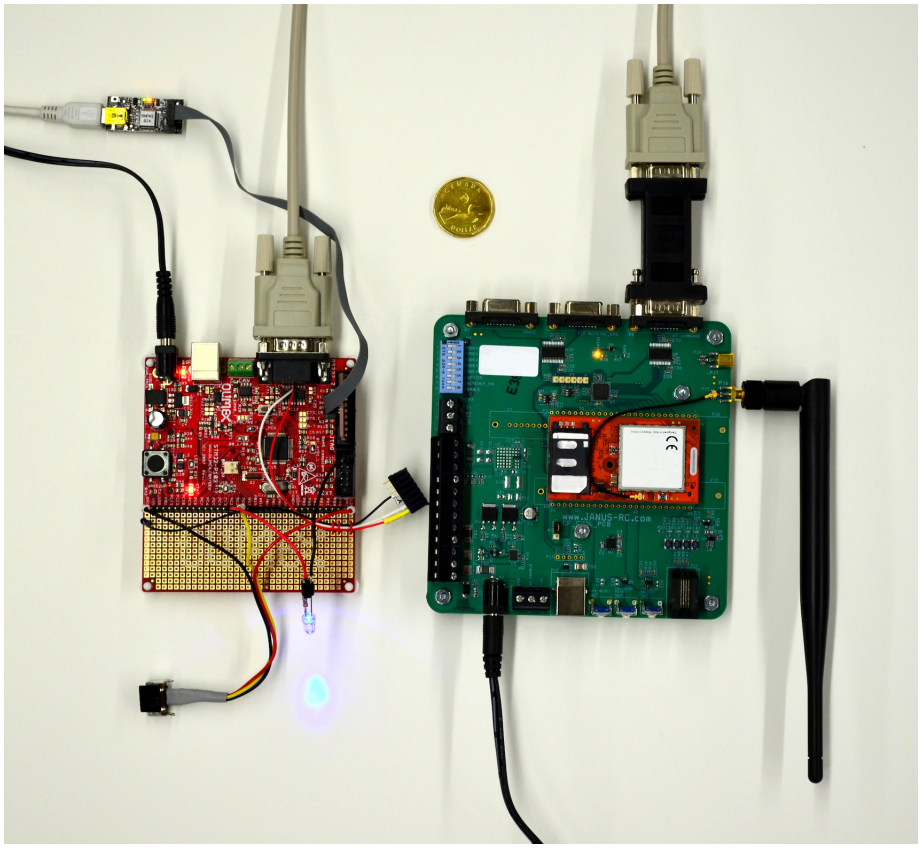


Figure 5.5: Early connected fleet monitoring system prototype using commercial development boards. (Courtesy of Copperstone Technologies Ltd.)

mechanical design portions of the project were simple and did not provide significant opportunity for agile design. The manufacturing test jig did require basic mechanical design for the bed-of-nails and PCB clamping features. This test jig is picture in Figure 5.6. An attempt to make this easy to adjust and assemble was made, but this was not particularly successful. The best example of agile design in the test jig was to leverage as many COTS components as possible, like the enclosure and the commercial development board that provided the foundation of the electronics hardware. During firmware development, an attempt was made to create reusable modules with clear APIs. This was only partially successful, as the underlying superloop design and lack of experience introduced complex dependencies. Furthermore, instead of keeping the design simple to achieve only

the minimum required feature set, a complicated design was often implemented to readily handle a variety of possible future features, of which only a few were used. Debugging features were added to the firmware early, which helped improve the rate of development. Examples included serial stream debugging, JTAG and serial wire debug to allow breakpoints, and a basic set of test firmwares to verify that different hardware features or firmware modules worked properly. This approach is similar to TDD, but was not comprehensive. Later, the development of the over-the-air firmware update feature made it possible for the devices to be improved and features added after deployment, which was critical to rapid project evolution. Besides inline code comments, documentation of the firmware design and implementation was minimal. The limited effort put into program flow and state machine documentation was not maintained or very clear. Further, the comments were not created with discipline or in a manner that would have allowed for automatic code documentation. In contrast, the firmware development environment was documented concisely to allow rapid setup. The availability of prototypes enabled regular firmware testing throughout development, but this testing was not well structured and did not fully cover the firmware. Software development leveraged existing frameworks and tools that had a high level of modularity and debugging capabilities for monitoring and feedback. There was no formal TDD and very limited unit testing, though most of the codebase relied on usability testing, which was carried out regularly but not formally. The communications test and simulation tools did allow protocol testing of both the server and device side elements independently.

5.1.3 Project Successes

The greatest area of success was in the power supply design. The power supply worked well and met specifications in the first revision of the second prototype. This may be attributed to the use of simulation to prototype and verify the design virtually, leveraging vendor application notes, and the care in documenting quantifiable requirements, calculations and design decisions. Other hardware features showed similar success, but the power supply was the most complex and presented the greatest technical risk. The generally modular design of the hardware enabled adding features and

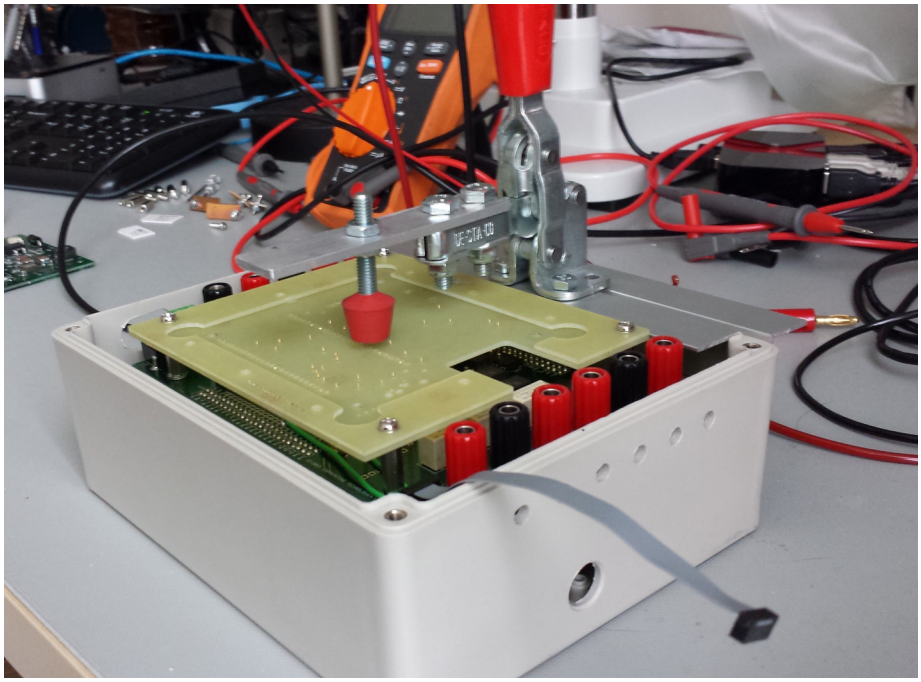


Figure 5.6: Manufacturing test jig for connected fleet monitoring system. (Courtesy of Copperstone Technologies Ltd.)

independent testing. Hardware support for additional features was added early to be enabled by future firmware updates. While this does not reflect the goals of simplicity and only including what was needed to meet requirements, the additional hardware features were minimal and the likelihood of use was deemed to be high to improve the reliability of the system. The effort to include these hardware features in the design was not large when compared to the cost and time to fabricate a new prototype due to a high component count. The use of an existing cellular module and COTS antenna reduced the technical risk and cost of designing and certifying a new module, at the expense of a higher total bill of materials cost. However, the modular nature of the modem decouples this part of the hardware, which means it would be reasonable to design a custom module in the future that would be far less expensive if demand for the product was sufficiently high to offset certification costs. While the number of hardware prototypes was low, the progression of the prototypes and feedback obtained through testing and use was reasonable.

The ability to develop firmware on prototype hardware early allowed for better understanding and basic optimization. Using core components like the microcontroller and cellular modem early in the design and prototype process allowed the team to gain familiarity with features and limitations. Focusing on elements of the firmware that required greater reliability did allow the team to have confidence in the resulting features. While lacking elegance, the accuracy and limitations of time keeping system were acceptable. The firmware update features were also implemented successfully, which allowed rapid development updates to be deployed to demonstration devices, while also providing a path to future improvements once the product had been released to end users.

Finally, by using common frameworks and services for the server and staying well within their limitations, deployment and maintenance of the web backend subsystems was simple.

Overall, iterative prototypes in all areas of the project helped develop the team's understanding, while also improving client buy-in. The prototypes provided useful feedback, even if the frequency was low. The impact of this feedback was clear from the difference in performance and reliability between the initial prototype and final prototype and production-ready design.

Careful design using rules of thumb and standards enabled both certifications to be passed on the first attempt. Certification testing can significantly impact the schedule of a project, as it depends heavily on the availability and turnaround time of an external service provider. Failing the tests results in the need for design changes, new prototypes and additional rounds of testing, all of which are time consuming and expensive. In addition, in-house testing to reduce risk of failure is difficult due to the high cost of building test setups.

5.1.4 Project Challenges and Shortcomings

Numerous areas for improvement were identified following the conclusion of the project. The team did not collaborate with the client enough. Progress updates and feedback on features were intermittent, even though the client was willing to work with the team to achieve a tighter feedback loop. Part of the challenge in providing regular progress updates was that work breakdown was too

coarse, and it was difficult to isolate clear delivery of value on a small enough timescale (for example, on a weekly basis). Improving feature breakdown and understanding clearly the criteria of value of this work would have enhanced this collaboration. In the same way, the team did not deliver value regularly, and often provided results only at the end of large milestones. This significantly increased risk and made meeting key client deadlines challenging. In some cases deadlines were not met appropriately. Original time and budget estimates were exceeded due to insufficient technical and management experience, poor prioritization and lack of focus on items of value. This was reasonable within the context of the contract, but resulted in delays that were detrimental to the client and to the team.

While the team pushed to minimize documentation, the approach was generally too broad and did not optimize the delivery of value. There was very limited formal documentation on the design of the server software or microcontroller firmware. This made changes and reviews more time consuming, but also caused difficulty when quantifying and explaining the operation, behaviour and performance of the system to the client in a simple and understandable manner. Documentation that was created throughout the project was not maintained or updated with discipline, and occasionally not referenced at all. For example, while the initial scope document was reasonable and provided guidance for the project goals, it was not readily used or improved as the project progressed. The exception was some of the quantifiable specifications, especially for the electronics hardware.

The team carried out regular status meetings, but these meetings lacked structure or vision. This practice did help the project progress, but impact would have been greater with clear goals for review and planning activities. While greater formality may not have been necessary, consistency would have been welcome. In addition, reviews would have been far easier and less ambiguous if there had been clear metrics for evaluating completed tasks and features. As part of the planning and review in the project, a feature or task backlog was implemented using the online software tool Pivotal Tracker (for details, see Table A.2). While the intention was reasonable, the implementation was not effective. Even though the tool helped with some estimation and prioritization, the team failed to use the tool properly or with discipline. It was used as a general task list rather than as a

Scrum-style user backlog as the vendor intended. The team did not use a Scrum-style management approach to leverage the software's features. As a result, the team had poor direction and tasks were not always prioritized to deliver the greatest value. This would also have been helped by slightly more structure and collaboration in the planning and review activities. Furthermore, the team attempted to divide work by feature, but as work granularity was too large, delivery often took weeks instead of days. This also led to insufficient effort on testing or review due to the size of the delivered feature.

While both the overall architecture and the electronics hardware design were successful, the firmware was not well implemented. An attempt was made to reuse and improve source code modules developed for early prototypes, but insufficient time was allocated for the appropriate rework that should have been done. Limitations often propagated across prototypes to the final implementation. Instead of improving the existing code, new code was instead adapted to existing shortcomings. As the project progressed, this resulted in a significant amount of interlinked, dependent "spaghetti" code, making the code difficult to maintain, improve, and understand. While some parts of the firmware was well written (especially as the developer gained experience), this was overshadowed by the poorly designed and implemented areas. Furthermore, code reviews were not carried out and testing coverage was sparse.

The team did not incorporate enough feedback from the client as to which features would provide the most value. For example, significant effort was spent on making a variety of device parameters configurable from the end user web application, and later the client requested this feature be disabled to simplify the user experience.

One of the important demonstration milestones for the client was not met successfully due to a small and readily tested technical issue. Even though the potential problem was obvious, the team did not test for the issue, and did not have the infrastructure or process in place to rapidly address the problem during the demonstration. Improved understanding and collaboration with the client prior to this milestone may have helped each party be better prepared.

Finally, there was limited feedback on the technical success of the system after the project conclusion

due to lack of progress through the product lifecycle. Subsequently, effort put into developing a feedback and monitoring infrastructure late in the project (i.e. the manufacturing test suite and QA database) did not deliver the value that was expected.

5.2 Case Study 2: Amphibious Rover Prototype

A prototype amphibious rover was developed to support geotechnical measurements and sample collection in varied terrain [62], as pictured in Figure 5.7. The system utilizes a unique screw-drive propulsion system which was predicted to be more successful in the primary target environment of oilsands tailings. Key parts of the design included an electric drive, a modular payload interface and actuation system, and remote teleoperation with the capacity for autonomy. The gross vehicle weight of the rover is approximately 350 kilograms, of which 100 kilograms is shared between propulsion batteries and payloads. An onboard computer provided control and monitoring functionality to the operator via a radio link with the ground control station.

5.2.1 Project Description

The project represented the earliest phase of the product lifecycle and was very experimental. The goal of the project was to develop a single preliminary prototype to understand the feasibility of the drive system, better understand the operating environment, and provide a minimum viable product for building market and customer buy-in. There were many uncertainties in the project and the understanding of the market and desired technical specifications was limited, with most information gathered through informal discussion with a single potential customer. The project was expected to end after initial prototype testing with the intent of transitioning to the next phase of the lifecycle with the same development team.

The screw-drive propulsion system design has not been previously well characterized. There are few examples of existing products leveraging the propulsion method. Furthermore, the general

Figure 5.7: Prototype amphibious rover during slough testing. (Courtesy of Copperstone Technologies Ltd.)

technology application area of applying robotics to environmental monitoring in industrial settings is relatively new. This indicated that the project would be very exploratory. Preliminary payload weight specifications indicated an expected size and weight envelope that would make system prototypes relatively expensive.

Since the project was strategic and internal to the organization, the client was not external. The project team acted as the client, and fully supported rapid development. The requirement to deliver a demonstrable minimum viable product was understood by all team members. The team interacted regularly with an outside stakeholder and potential client. This interaction was limited in formality, and it is noted that this stakeholder also understood the experimental nature of the project.

The business model of the product was not set at the beginning of the project, pending a greater understanding of technical feasibility and integration with existing industrial processes. The models that were considered included sales of units, rental or lease of units (possibly with operators), or using units internally to provide measurement data as a service. The primary market for the system is for industrial environmental monitoring, with a small potential secondary market of research and experimental work. The primary market has a relatively small number of clients but each client potentially has a significant amount of applicable terrain. Working closely with a single potential client and stakeholder provided access to domain expertise of the operating conditions and the measurements and payloads of interest to the primary market. The team was also able to collaborate with this stakeholder on testing and demonstration to help build client support. The collaboration was important due to the difficulty of obtaining access to real test environments otherwise.

There were several hard requirements identified early in the project. The joint demonstration of the system with stakeholders and potential clients (synchronization of multiple parties) needed to be scheduled in advance. Testing was also dependent on seasonal climate and weather conditions, and had to be completed before temperatures dropped below freezing regularly.[1] The project was funded internally by the organization, which introduced a firm upper budget limit. The goal of keeping costs low (while still meeting schedule requirements) was partially achieved by deferring pay to the development team.

The amphibious rover is a mobile field robotic system. It is a good example of a full mechatronics system, which features and subsystems incorporating mechanical components, electrical elements, electronics, firmware, and software. This includes actuators, sensors, and controllers. A simplified block diagram is presented in Figure 5.8. The overall system design required balancing size, weight and power. This resulted in significant tradeoffs and managing the impacts of one change on the others. For example, increasing power through larger motors resulted in more weight due to heavier motors, larger batteries, and stronger mechanical components.

[1] This resulted in the main project timeline lasting from initiation in early July to final demonstrations in early November.

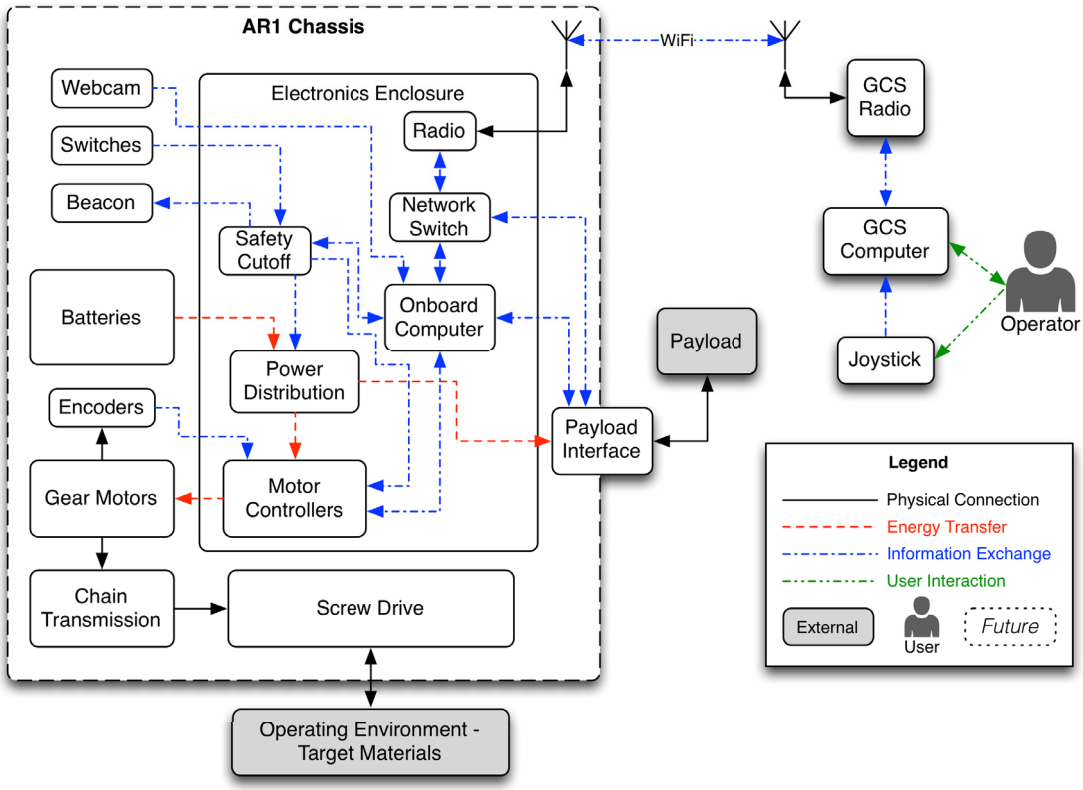
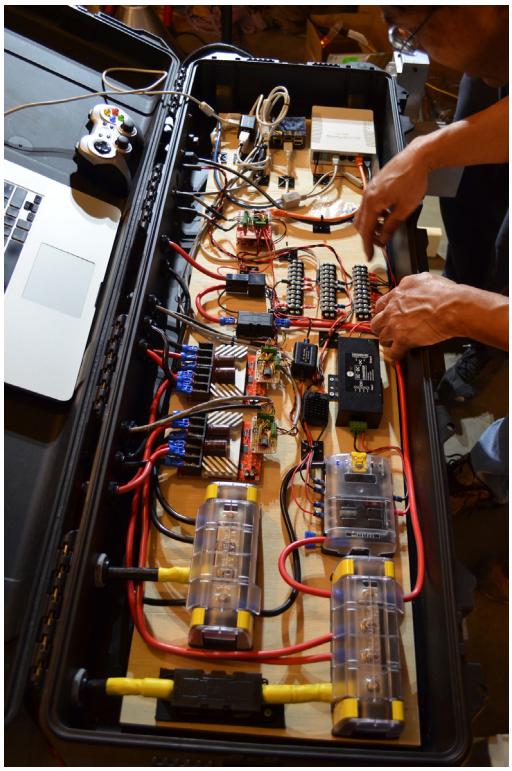


Figure 5.8: Prototype amphibious rover simplified block diagram. (Courtesy of Copperstone Technologies Ltd.)

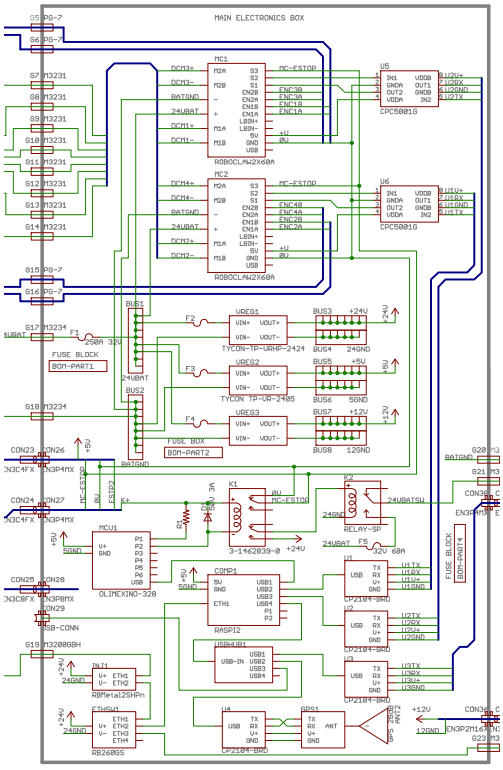
A variety of subsystems and features involved mechanical design. The chassis provided a structural skeleton for the system. Stiffness was important to handle the large moments generated by the drive system. Weight was also a concern to ensure the rover was sufficiently buoyant. The unique screw-drive propulsion system was powered through chains by geared DC electric motors. The screws were designed as watertight drums to provide buoyancy. The poor efficiency of a screw-drive made size, weight and power tradeoffs challenging and increased the importance of the energy density of the overall propulsion system. The extremely high torque requirements made balancing strength and weight difficult. An adjustable and adaptable payload attachment system was developed using t-slot aluminum extrusions. This included an optional vertical actuation system realized with an

additional DC gearmotor and a rack and pinion. A variety of possible payloads were expected, and a basic interface specification included mechanical attachment, electrical supply, and both a serial and ethernet data connection. The system was kept flexible to adapt to the variety of possible payloads. The amphibious behaviour was targeted at a wide range of terrain from hard ground to water, with material in between ranging in water content, cohesion, adhesion and viscosity. Examples of terrain where the system was tested includes grass, mud, snow, slough, open water, and water with thin ice. The main target terrain was various oilsands mine tailings materials including mature fine tailings (MFT) and centrifuge output. This range of environments increased the importance of ingress protection (IP) during operation. The selected motors had an appropriate IP rating, and as did the COTS mounted bearings.

Electrical energy was the primary power transfer medium in the system. Power management, including distribution and protection, represented an important part of the design. The propulsion system had a maximum power draw of approximately 6 kW at the battery voltage, while other parts of the system required a regulated supply of 12 V or 5 V. Inexpensive lead acid batteries were configured to supply a nominal 24 V to the system. The low voltage provided easier integration, lower cost, and lower hazard at the expense of high currents and lower efficiency. The batteries were removable for charging and used high capacity connectors for easy removal and replacement, as well as to provide a physical power disconnect. Each motor used a similar connector to make adjustments and testing easier. Between the batteries and all loads, a high current fuse was used to prevent significant energy release. Each subsystem load was also protected by an appropriately sized fuse. Removable connectors were used throughout the system for easy isolation, testing and configurability. Commercial motor controllers were specified that had excellent feedback features for monitoring or closed-loop control, including velocity and current measurements. The design also leveraged COTS voltage regulators for the system voltage rails. A COTS environmental enclosure was used to protect the electrical and electronics components, while IP-rated components, connectors and cable glands were used externally. A view of the inside of this enclosure is shown in Figure 5.9a. The wiring diagram and schematic was developed during design (Figure 5.9b), while the components were laid out in the case and wiring harnesses sized during prototype fabrication.



(a) View inside enclosure after assembly. (Courtesy of Copperstone Technologies Ltd.)



(b) Subset of main wiring schematic corresponding to inside of enclosure. (Courtesy of Copperstone Technologies Ltd.)

Figure 5.9: Rover electronics enclosure.

The two main electronics subsystems were the safety cut-off system and the onboard computer and communications. The safety cut-off system used a dual-redundant design to disable the motor controllers and cut power to the payload using a signal from two emergency shut-off switches. In addition to directly driving the cut-off signals, the switches were also monitored by a microcontroller that could drive the cut-off signals. This controller provided status feedback to the onboard computer as well as directly to the users through an amber beacon. It is noted that there was no controllable physical disconnection between the input power from the batteries and the propulsion or electronics subsystems. This is due to the very large current requirements that would have necessitated heavy, bulky and expensive components to accomplish. Instead the system relied on the tested COTS motor controllers to disable motive power. The primary control of the system was carried out using a small readily available general purpose single board computer (Raspberry Pi). Opto-isolated serial-to-usb adapters allowed communication between the computer and the motor controllers, the payload and the safety cut-off microcontroller. A GPS receiver was also connected for position measurements while webcams improved operator awareness. A local wired ethernet network connected the computer to a high-power WiFi radio for communication with the ground control station. The onboard network allows for extensibility and for interfacing to advanced payloads (both between the payload and onboard computer, as well as between the payload and the ground control station).

There was little firmware development for this prototype. Most functionality was handled by the onboard computer. The safety cut-off microcontroller did require a small amount of simple but critical firmware development. It is noted that the redundancy in this system mitigated the risk of errors in the firmware. The cut-off system was reviewed and tested to ensure it functioned correctly, as it was used to override the onboard computer, which was far more complicated and did not undergo any significant review as this project was in early prototype stages and most of the design was expected to change.

The onboard computer used a Linux operating system to provide most of the system functionality. The software was generally extendable to almost any hardware, with a standard Linux distribution

and no special features used (USB and ethernet connections only, along with power). The system did not implement a realtime kernel, redundancy, or any other special features for this prototype. Control and feedback in the system were built on the Robot Operating System (ROS) framework [70] to allow for easy development, adjustment, and the ability to leverage many built-in features. A mix of community-provided and custom software was used. ROS enabled significant data logging capability for analysis and review, including onboard video capture matched with motor controller and status feedback streams. The ROS framework also extended well to the ground control station for operator monitoring and control. The software stack and the system design for the rover were complex.

The ground control station (GCS) consisted of a basic notebook computer connected to a high power WiFi radio for communication with the rover. A consumer gamepad was used for basic remote control. ROS was used as part of the GCS software stack as well, to leverage the features available on the rover. As this was a prototype system, limited effort was placed on the design of the human-machine interface or the GCS. No graphical user interface was developed, and system bring-up was completed manually. This reduced usability to expert operators only, and introduced some delay when the system was started. An emphasis on monitoring and data collection saw the development of system monitors and onboard video feeds, which proved effective in aiding the expert operator. There was no ruggedization in this prototype GCS due to the use of a consumer-grade notebook and gamepad. This led to challenges during testing in harsh environments due to dirt and cold.

The team was able to leverage some existing knowledge and experience to offset the uncertainty in the propulsion system design and the operating environment. While there was no quantitative data available on the characteristics of the oilsands tailings material, the team did have some domain knowledge on the expected operating environment and a limited amount of direct experience operating a robotic system in a similar environment. The potential client provided insight on desired payloads and operations logistics. The team also had significant previous experience with many of the underlying subsystem components, including ROS, the embedded microcontroller and firmware,

electronics hardware components, and the onboard computer.

5.2.2 Project Methods

From the beginning of the project, the team understood the need to move quickly to deliver the first minimum viable product prototype and demonstrate the system to potential clients in order to build stakeholder support. The team was also aware of the high technical risk, and that feedback from the first prototype was important to evaluate the technical feasibility of the design. Numerous tradeoffs were made in the first prototype to focus on key aspects of the design. These goals included producing a system that provided opportunity to carefully test the propulsion system design while meeting minimum functionality for demonstration for stakeholders. The team expected that the best design feedback would be from a real prototype due to the great uncertainties in the specifications and characteristics of the operating environment. Based on previous experience with other robotics systems (both unmanned ground vehicles and unmanned aerial vehicles), there was an expectation of unintended consequences as a result of system integration [52, 61]. To address this, integration and test activities were prioritized and given an ample time budget. It was clear early that this project would represent an early part of the lifecycle and deliver a single major prototype iteration that would be the first of several. As a result, most features and elements of this prototype were designed and implemented with little or no design iteration. Tasks and features were in part broken into phases due to the cost and difficulty of building a prototype, resulting in a balance of incremental and progressive design iteration. The team did spend more time reviewing and iterating during the design, concept and architecture development of the mechanical subsystem components, as these were expected to be more expensive and difficult to change later (even only considering the lead time to modify or manufacture new components). Other components prioritized due to potential lead time included specification of motors and controllers, while items available locally or more quickly (for example, through inexpensive overnight shipping) were pushed to later in the schedule. There was also iteration present during testing. The team was able to carry out multiple rounds of testing and leverage the client technology demonstration as another test

opportunity. These testing tasks were included later in the project and provided excellent feedback on the design.

The management strategy for this project was fairly ad hoc with significant resemblance to agile or lean methods. Some planning was carried out at the beginning of the project to understand some of the goals and characteristics of the project, and outline a strategy to meet hard requirements in budget and schedule. The team worked together in an ah hoc manner with limited formality or structure, but still managed to follow an approach resembling the pull paradigm. A basic plan and informal prioritized task lists were used, with regular updates and collaboration that resulted in fast feedback. While the approach was not formally a Kanban-style pull method (the team was unaware of this methodology at the time), it was very similar in style and intent. Decisions were made collaboratively, supported by informal prioritization and analysis. This helped to rein back activities that were not delivering immediate value. Regular, on-demand review, planning and prioritization activities were carried out to review completed tasks and then assign new tasks based on the current state of the project. Collaborative co-design efforts were also an important practice, especially early during architecture or concept development, with group brainstorming sessions and design discussions.

The project team had some specialization and overlap of experience. It consisted of three mechanical engineers with experience in different areas. One team member had a basic level of domain knowledge about the oilsands tailings materials. The team was intermittently joined by an experienced fourth member, also a mechanical engineer, with additional domain knowledge during development, fabrication and especially testing. During design and implementation, the team tended to work by feature or subsystem, but also divided tasks by discipline. Mechanical development was shared, but one team member focused on electronics and architecture, another on electronics and electrical design, and a third on software development, based on past experience. While individuals focused on separate tasks, the team continued to collaborate on planning, design and review. The team worked together closely during prototype fabrication and testing, during which many tasks required more than one individual to complete safely.

The tight collaboration achieved by the team was supported by regular updates and a variety of tools. Limited on-demand interaction with the potential client was necessary for coordinating the field demonstrations and as part of some basic development activities to integrate a client payload subsystem. This was accomplished primarily with email and phone communication. The project team was distributed, and used tools like cloud-based file sharing, text chat, video chat and email to manage the project. There were limited face-to-face meetings early in the project during which the team completed high level planning and review tasks, but these were not very formal. The team carried out regular, sometimes daily, design discussions and status meetings. These updates were similar to a daily standup suggested by Scrum and Kanban methods, but with less structure and vision. The discussions certainly helped keep the team abreast with current project status and helped the project progress, but may be been even more effective with clear goals outlining review and planning activities. During these ad hoc discussions, there was clear emphasis on either co-design discussion, informal but joint review of completed work, or planning activities. While the primary focus was on completing design tasks, the team remained distributed. When the design activities shifted to implementation and testing (i.e.prototype fabrication and integration), the team co-located for periods of time as required.

The team attempted to formally capture specifications with quantitative metrics early in the project with discussion and team brainstorming, but quickly discovered many requirements and features were challenging to quantify and prioritize and were often uncertain and ambiguous. This was quickly abandoned in favour of matching specifications with tasks where possible and adjusting them informally as the project progressed and more information was known. Several key specifications were outlined early, especially regarding payload requirements and the overall system operation. Another outcome of trying to fully specify the system was that a number of important unknowns were identified that drove task prioritization, especially with regards to the uncertainty in propulsion system design. Since the project was a very open-ended with significant focus on research and feasibility assessment, the lack of specifications was an intrinsic aspect of the work.

The team completed a reasonably detailed budget estimate for both capital costs and time costs

early in the project. This budget was used as a guide to meet the hard budget limit as well as a baseline to understand estimation accuracy by comparing against actual expenditures and time spent which were both tracked. A high level schedule focused on milestones was prepared to help the team complete prototype implementation and testing prior to the time-dependent client demonstration campaign. The scope of the project was not formalized, but early in the project and regularly thereafter, the team had informal discussions to ensure all members understood the vision and goals of the project, and to adjust the direction as necessary.

Mechanical subsystems and design work was reviewed with a medium level of formality. This included calculations, CAD models, manufacturing drawings, and bill of materials. Review of these items was prioritized due to the cost of components and that a number of parts had to be fabricated by an external machine shop. It was important to provide clear communication to the supplier to ensure the parts were manufactured as intended. The electrical design, specifically the overall wiring diagram and the components selected, was reviewed informally and collaboratively to identify any issues in the design prior to purchasing components. Problems caught during reviews were subsequently corrected prior to procurement and prototype implementation. No reviews of the software were carried out due to the tight schedule, the ease of change later, and the likelihood of change to adjust system functionality. No formal retrospective activities were carried out during the project, though effectiveness of the process was considered during the project conclusion phase. Small adjustments were made to the process and practices through off-hand observations and discussion within the team. For example, the importance of time tracking was reiterated as the team failed to maintain records with discipline.

Overall, the team followed a method similar to that described by this work in Chapters 3 and 4. While the method was fairly ad hoc, there was a clear intent of following some agile and lean principles during the project. The team and organization cultures were compatible with the adaptive and feedback-oriented strategies of agile methods. There was a clear project initiation phase with limited characteristics analysis, preliminary architecture discussion, and high level budget and schedule estimation. This transitioned to a iterative and pull-style phase of development with tight

collaboration amongst the team. The pull-style of task management was effective for the team, though no time-boxed iteration was implemented beyond identifying several key project milestones. The team was able to synchronize with other stakeholders for larger milestone activities without supporting practices or artifacts due to the simplicity and small timescale of the project. The team tended towards a phased approach in design due to prototyping cost and lead time limitations. The team did attempt to deliver usable and reviewable work at task completion within the prototyping constraints, but more effort could have been placed on the earlier delivery of working prototypes or results. While often informal and ad hoc, the team took time to discuss the approach and make a plan at different tiers in the project, carry out the work (sometimes collaboratively) and carry out some basic review and approval activities. There were limited metrics outlined for review on most activities, though there were exceptions. For example, mechanical design drawings were reviewed carefully against a standard prior to release to an external supplier. Prototype performance testing and client technical demonstrations provided important design feedback later in the project. In the overall product lifecycle, this was important and provided the highest level of feedback (on a project scale) about overall design performance that helped the team decide the direction and priorities of the project in the next project iteration. The project conclusion was represented by a transition to a new cycle of prototype development by the same team. The main activities were to carry out post-mortem retrospection and design review to determine design limitations and development priorities for the next lifecycle stage. The team also put a small amount of effort into archiving artifacts and recording relevant information that would be beneficial in the next stage. While artifacts and documentation were minimal throughout the project, it was deemed sufficient given the high level of team continuity. It is expected the next phase will begin with a short project initiation phase followed by an iterative stage with a better understanding of appropriate methodology practices.

There was very little explicit risk management carried out during the design. The team did spend effort ensuring the safety cut-off subsystem design worked properly due to the high energy content of the system. The team also attempted to test the system prior to demonstration with external stakeholders to ensure the demonstration would be successful. Regular safe work practices and risk

mitigation were used during fabrication and testing activities.

A variety of artifacts and documentation was created to capture knowledge during the design process. As mentioned previously, there was no formal project vision or scope developed, and the initial attempt at a formal requirements document was limited and not kept up to date. A basic and informal schedule was used to help synchronize with external stakeholders, and was simple to avoid detracting from understanding. The budget estimate was prepared in greater detail, which included both capital cost estimates as well as time estimates broken down by subsystem and by phase. No architecture documents were prepared, though the team shared an excellent understanding of the system architecture through discussion aided by informal sketches. A variety of informal notes and sketches supported brainstorming, design discussion and concept development. Issues and defects were captured in a similar manner. This informal documentation was often recorded in laboratory notebooks and was archived through photographs or in shared documents to ensure they were accessible throughout the project. Furthermore, team discussions over email and chat were automatically recorded and archived in a searchable format. Time tracking was done carefully and with reasonable detail using the online service Toggl, while the budget was monitored and updated regularly as well using the same initial estimate artifact (a shared spreadsheet in Google Drive) (for details, see Table A.2). There was varying levels of formality in documenting detailed design activities. Calculations were carried out using spreadsheets for easy modification and review. Items requiring outsourced manufacturing or component ordering were prepared with greater formality. This included CAD models and associated manufacturing drawings, bill of materials, orders and quotes. A wireframe representation of the CAD model of the system is shown Figure 5.10, while a manufacturing drawing sample is presented in Figure 5.11. A reasonable level of detail was present in the main wiring diagram (Figure 5.9b), with supplemental sketches used for some simple prototype circuits. An example of a supplemental sketch is depicted in Figure 5.12. Firmware and software was not documented beyond minimal comments in the source code. While the firmware was simple enough for this to suffice, the complex software stack would have benefitted from greater design and implementation documentation. During project conclusion post-mortem discussions, a range of notes were taken based on team member observations to ensure ideas and insights

would be carried over to the next project. A reasonable attempt was made to absolutely minimize artifacts and documentation while still making sure the team understood the project. The informal documentation was acceptable for internal use, and where external release was required, it was not difficult to prepare a more formal version of the same information. The team was also able to maintain appropriate levels of knowledge transfer and archiving of information by sharing all artifacts with online collaboration tools like Google Drive and BitBucket (for details, see Table A.2).

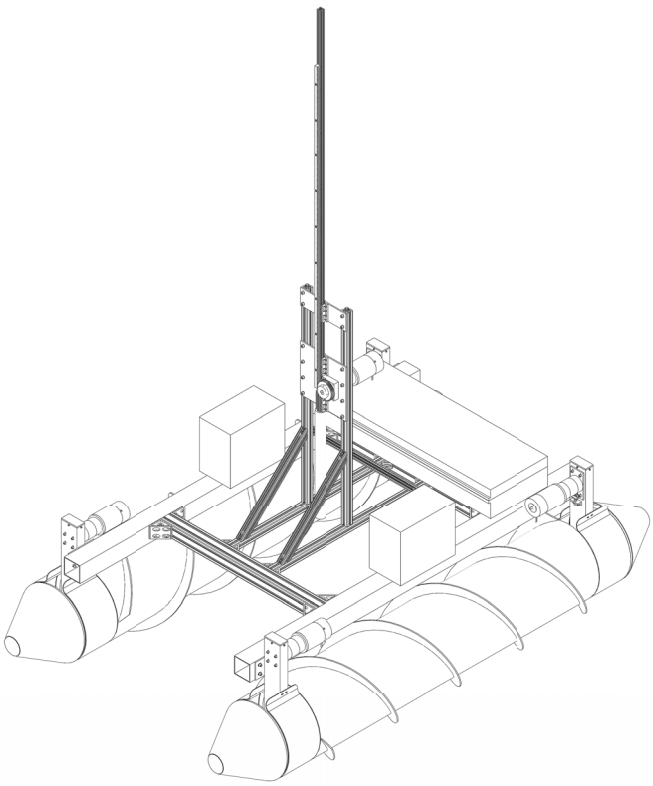


Figure 5.10: Wireframe representation of rover CAD model. (Courtesy of Copperstone Technologies Ltd.)

For this project, most intermediate prototyping methods (like modeling and simulation) were not expected to deliver enough value for the effort given the timelines and goals of the project. It was

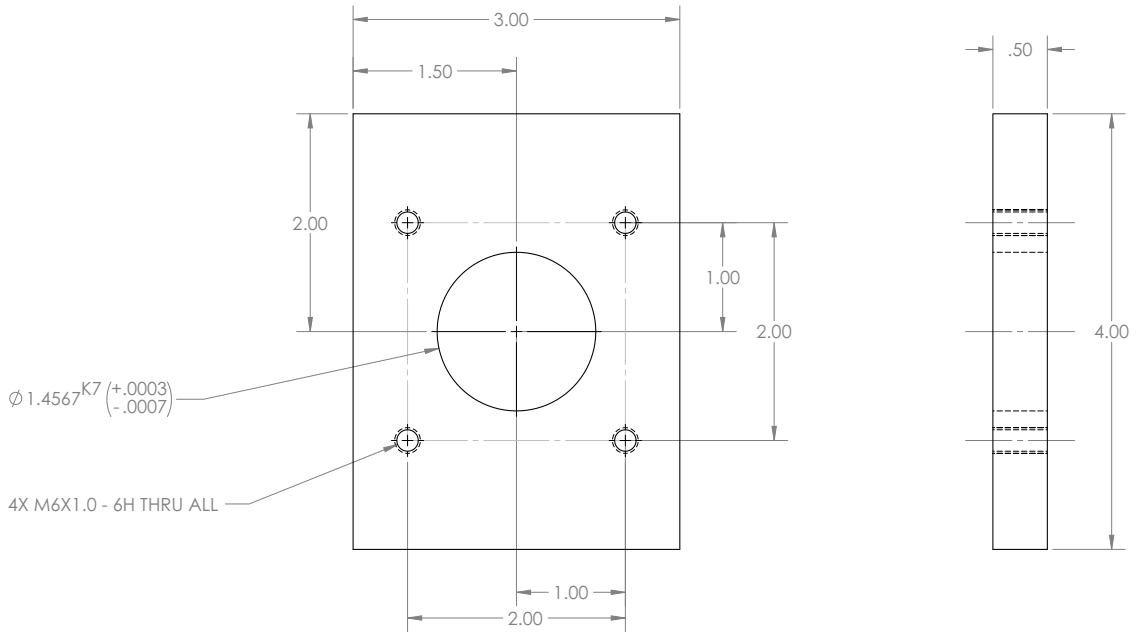


Figure 5.11: Subset of a rover motor bearing housing manufacturing drawing. Demonstrates the simplicity of part design. (Courtesy of Copperstone Technologies Ltd.)

more appropriate to work on the full prototype. An exception is that CAD was used heavily for mechanical design. Subsystem elements were often tested independently before being integrated into the prototype. For example, initial prototyping and testing was carried out with the motors and motor controllers during software development to reduce risks associated with testing software in the full prototype due to the high energy in the system. The full prototype was built with several techniques, and typically targeted a field fit and adjustability. By relying on good tradecraft, the need for absolute detail in design efforts was reduced. This resulted in faster and easier full integration given the low level of complexity and optimization. Examples include making the motor position adjustable to both align the bearing and shaft and to remove chain slack. The chassis leveraged aluminum extrusions to improve frame and subsystem mounting adjustability and assembly speed. An effort was made to create reasonable interfaces between different physical modules, for example in the form of common bolt patterns, fasteners and connector pinouts. This helped to reduce the number of unique parts needed, making spares and replacements more accessible. Where

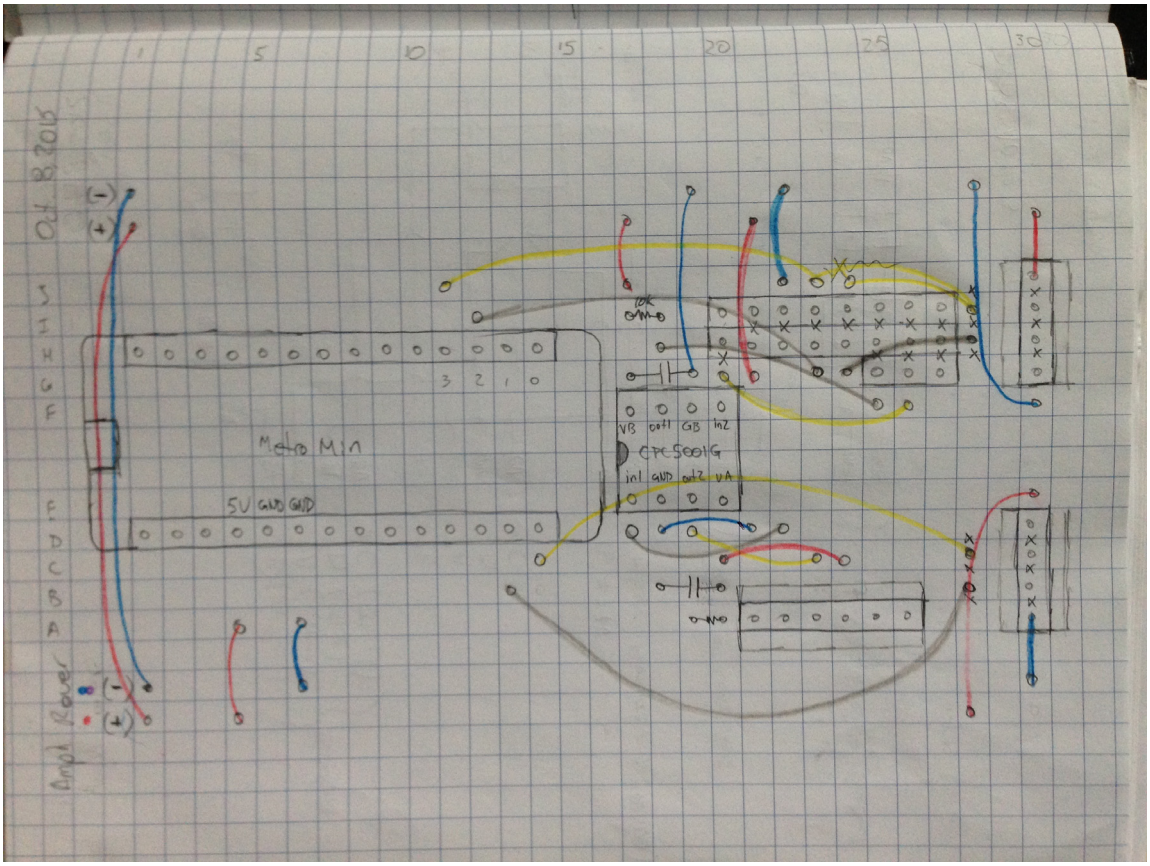


Figure 5.12: Supplemental circuit layout sketch for fabricating a solderable breadboard. (Courtesy of Copperstone Technologies Ltd.)

possible, easy access to components was maintained. The use of readily available COTS components reduced manufacturing cost and lead times. Custom precision components were avoided or designed to be as simple as possible. For example, mounted bearings were used most places, and a single custom bearing mount was designed to be simple and fast to machine. Electronics subsystem prototypes leveraged commercial development boards and breakout boards, supported by solderable breadboards for simple circuits and to integrate connectors. Connectors were solderable or used inexpensive crimpers to avoid purchasing costly assembly equipment. Where components were enclosed, screw terminals, terminal blocks and busbars were used to make wiring adjustable and fast. Though the reliability and longevity of some of these solutions was limited, it was acceptable for

the limited operation of the prototype in this project. The software stacks used in the project were flexible enough to be readily developed on desktop machines and the onboard computer was inexpensive enough to purchase multiple units for development and test configurations.

The team demonstrated agile design techniques like simplicity, modularity, and feedback by design when developing the amphibious rover. Rapid prototyping was the goal of the project overall, but not implemented in an iterative manner. Documentation was limited, and may not have been quite sufficient in some areas, particularly software. This was augmented by close team collaboration and continuity. Mechanical and electrical design artifacts were appropriate for the fabrication and procurement required. The team attempted to maintain disciplined development, especially in a few key areas like time tracking, but this remained challenging.

From a mechanical standpoint, the team tried to keep the design simple and minimize manufacturing while also keeping components adjustable and modular. Examples include incorporating aluminum extrusions and modular drive system components. A reliance on external machining services was avoided for most components. A basic payload interface was specified, but designed to remain flexible to accommodate a wide variety of different payloads. Electrical components were made modular and interchangeable where possible. For example, the opto-isolated serial to usb converters were identical for multiple connections. The selection of USB and ethernet made interfacing easy. Even though these represent complex systems, they are readily available and modular and did not need to be reimplemented by the team. Where possible, connections like screw terminals were left accessible for testing. Connectors were used in many places to increase modularity, adjustability, replaceability and the capacity for isolating subsystems or modules. A key design goal was to enable monitoring and feedback of motor controller performance, safety system status and control status to both the operator in real time and through data logging for later analysis. Much of the overall design was kept simple to make development and debug easy, at the expense of ruggedness and reliability. This was acceptable for the prototype. For example, a controller area network network (CAN) implementation would be more robust than the USB and serial connections used, but would have required significant development and integration efforts by comparison. Firmware was kept as simple

as possible, and achieved the goals of reliability and basic system status feedback well. The ROS framework enabled significant feedback and monitoring through built-in features and by extension with custom software. This was critical to gather large volumes of performance data needed to evaluate and improve the design. It is a very modular framework, allowing easy customization and extension. The team used many existing modules and features. For example, a driver for the motor controllers was available through the community. The logging and debug features were excellent for feedback and review. For example, the live video feed and corresponding video log were beneficial for operator awareness and during post-mission analysis, respectively. A sample screenshot of the operator payload monitoring video stream is presented in Figure 5.13.

Figure 5.13: Amphibious rover operator payload monitoring video screenshot. (Courtesy of Copperstone Technologies Ltd.)

5.2.3 Project Successes

The overall project was successful in delivering the demonstration prototype as envisioned. The design was developed, implemented and tested rapidly, with project goals achieved within a short time period given the complexity and cost of the system. This was attributed to a team that collaborated effectively and was dedicated to project success and rapid progress. It was also helped by an effective prototyping approach to get to a minimum viable product suitable for demonstration and testing purposes. This involved tradeoffs like sacrificing runtime (which was not needed for basic

demonstration purposes) to reduce cost (by using inexpensive batteries with a lower energy density). The team also balanced detailed design with relying on tradecraft for prototype fabrication. Where the mechanical design was sufficiently definite, part manufacturing was outsourced while the simplicity and flexibility of the design also enabled fast modifications and changes to address integrations found during integration and assembly. For example, the enclosures for the chain drives in the propulsion system interfered with the chains, which was addressed by quickly modifying the enclosure on site with no major impact on performance. The use of modular COTS components in the system helped enable modularity, speed development, and reduce the use of costly custom components. Sealed and mounted bearing units removed a significant amount of precision machining, while combination gearbox and motor units made alignment and mounting less critical. Commercially available motor controllers, onboard computer and network equipment accelerated integration, while development boards, breakout boards and solderable breadboards allowed for rapid electronics prototyping. Assembly and maintenance of the prototype was improved through a focus on adjustability and modularity in the design. The team drew on previous experiences of prototyping and integrating mechatronic systems when developing the design. Careful isolation and grounding configuration was incorporated to eliminate ground loops and components were mounted firmly to handle vibration based on lessons learned during UAV system development.

The design features and prototyping approach allowed the team to react to test feedback rapidly. When testing showed that greater chassis rigidity and more drive torque was needed, the chassis was modified and the motors were swapped with a very short turnaround of about two weeks. This was made possible by a simple chassis design that was easy to adapt and the selection of a modular motor product line with multiple performance options. In another case, a failed motor controller prompted the team to move to an alternative model that had similar interfaces. The feedback and data collection that supported testing provided valuable information to improve the design, both in the present project and for future phases of the lifecycle. Current measurements available from the motor controller supplemented observations with which the team concluded a drive torque increase was necessary. Overall, the team was successful in implementing an iterative testing program, culminating in a customer demonstration to gather stakeholder feedback and operate in a real field

environment.

The prioritization of critical parts of the design proved effective at mitigating risk. The effort spend on the relatively simple safety cut-off and status monitoring system was warranted, as this was used to break off unintended motion during testing due to a software defect.

The observations and analysis completed immediately following testing and demonstrations ensured quantifiable performance data as well as human factors, use cases and operating procedures were captured in sufficient detail to provide feedback and baseline information for design improvements. Issues were addressed either immediately or archived for use in the next project stage of the lifecycle.

While documentation was minimized, the team did have more than sufficient records to meet basic audit requirements. Specifically, the time, budget, and description of work completed was used to obtain a research and development tax incentive.

Another key area of success was in the feedback obtained through budgeting activities. The cost and time estimates prepared were compared against recorded expenditures and tracked time. This allowed the team to understand the accuracy and limitations of their estimations with the goal of improving estimation ability in future projects. Time tracking was divided into major activity categories (administration, budgeting, design, procurement, manufacturing, testing, and uncategorized). Estimates were made only for three categories (design, manufacturing and testing), with one estimate only covering a subset of expected work (no field testing included). Through analysis, it was determined that the actual time spent on all activities was 74 percent more than estimated, while actual time spent on budgeted activities was 34 percent more than estimated. The cost estimate also failed to include field testing activities, resulting in a 45 percent total budget overage and a 24 percent overage without including field testing costs. The total cost was still within the budget limit hard requirement set initially. As a result, the team was able to change estimation practices to include all activities and to improve estimations in categories that were less accurate.

5.2.4 Project Challenges and Shortcomings

While the project was generally successful, there were a number of areas where the team could have been more effective in both design and management. Design models and reviews could have been more comprehensive to increase the chances of identifying simple mistakes. In one case, the chains interfered with the inside surface of the enclosure. This was due in part to the fact that the chain was not included in the CAD model. While the design was reasonably agile and adapted to fabrication techniques available to the team, some items were were difficult to adjust due to poor implementation. For example, the motor and bearing position adjustment system used to align the shaft and tension the chain was not strong enough and time consuming to adjust. Even though the feature was important for correct operation, it required improvements. There were a variety of similar design enhancements identified, but this was expected due to prototype development characteristics of the project.

The limited effort spend on developing an operator interface meant that expert knowledge was required to control the system. This knowledge was not documented, making operation impossible even for other team members. The complex interface also resulted in significant setup times during testing.

The project team did not consider all use cases during either the budget estimation or the design process. In particular, testing activities and associated system transportation procedures were not analyzed. As a result, there were significant budget overruns due to field testing. The transportation of the rover was also difficult a lack of lifting points and an awkward battery mounting method.

A more directed testing approach could have been implemented with clearer goals and defined experiments. This may have provided a greater quantity of and more relevant performance data. The project may have also benefitted from pursuing less expensive and more frequent prototypes to reduce technical risk and uncertainty in the design earlier, and possibly with less investment. Unfortunately, the hard schedule requirements made this difficult.

While operation of the system lacked documentation, so too did the architecture and design of the software subsystems. Improving documentation in these areas would have reduced a reliance on team continuity for future lifecycle stages to be successful. The complexity of the software stacks used in the project made comprehensive testing or review difficult to implement, and a lack of discipline in software development manifested in poor use of version control and insufficient documentation. There were a variety of other areas missing key pieces of documentation as well.

The team may have been able to reduce uncertainty in requirements and technical risk by spending more effort gathering information about the expected operating conditions. This is also related to understanding in greater detail the goal and scope of the field demonstrations. While there was ambiguity in the demonstration plan early in the project, the team should have identified the need to address these areas of high uncertainty and taken appropriate action. Failing to do so reduced the technical success of the field demonstration. The field demonstration had already been pushed into late Fall, and as a result the ground was partially frozen at the test site. This reduced the success of the demonstration due to lower rover mobility by operating outside of the original technical specifications (frozen ground and thixotropic mud). The target materials' properties also showed the limitations of a differential steering method, which was effective in softer materials tested earlier. The anti-fouling coatings tested were also ineffective against one target material, a mud treated with a flocculating polymer. Regardless, the team was able to adapt to some changes in the demonstration goals supplied by external stakeholders, and the demonstration provided excellent value in performance feedback. In a related matter, the team put limited effort into verifying the market conditions early in the project, relying instead on qualitative and anecdotal market information from a limited number of sources. A clearer understanding here would have resulted in a more effective client demonstration.

Chapter 6

Conclusion

An agile framework and methodology model applicable to mechatronics design projects has been presented to address the shortcomings in existing approaches to project management and the design process. Previous experience with the waterfall approach to design taught during undergraduate education had proven frustrating when trying to deliver a successful solution effectively. The experiences, domains of interest and attitudes of the author's organization triggered a desire to shift to a method that valued agility to achieve innovative design. A review of existing methodologies indicated variation by discipline, with minimal, agile approaches popular in software projects and those with physical products relying on more formal and phase-oriented methods. It was also noted that there is no quantitatively correct way to complete all design projects, as success depends on the characteristics of the project and the preferences of the team. This has led to a vast number of different methodologies for design projects ranging widely in style and rigour, of which only a small subset were surveyed.

6.1 Review of Proposed Framework and Methodology Model

The framework and methodology model presented are applicable primarily to small cross-disciplinary teams working on exploratory projects that have high uncertainty and risk. While describing these systems as mechatronic reflects the different disciplines of interest, projects may encompass only a subset of these disciplines. These projects will typically be early stage prototypes or low volume industrial and commercial systems, where optimization for mass production is not a priority. Additionally, the project budgets will be relatively small, reflecting the size of the team.

By Wynn [104], the framework and methodology model can be generally classified as activity-based and solution-oriented, with some characteristics each of abstract and procedural models incorporating both a design-focus and project-focus. The frameworks for studying methodologies of Eisenbart [32] and Qureshi [72] place significant focus on the phase- or stage-based (i.e. progressive iteration) nature of most methodologies used across different disciplines. In contrast, this methodology is primarily feature-driven with incremental iteration, much more like agile software methodologies. As a result, it is expected that the design state or phase may be very different across the set of features or subsystems of the design.

6.1.1 Framework

A framework was presented to outline what aspects of a project a design team should consider when applying a methodology, especially when the team values agility. The framework also outlines the goals and patterns for design activities and deliverables (like prototypes and artifacts) that are applicable across all disciplines relevant to mechatronics.

It is important to understand how the design project fits into the overall lifecycle of the product to help identify overall goals, inputs and outputs. Projects in the earlier stages of the product lifecycle will typically be more exploratory and suited to an agile approach. Different business models for the product will drive design decisions, with models exhibiting greater change capacity able to benefit more from an agile methodology.

Iteration is present both in the design process (progressive or incremental strategies) and management (frequency of planning and review or feedback cycles). In striving for fast feedback, a pull paradigm is recommended at low iteration tiers, supplemented with appropriate synchronization practices that leverage time-boxing at higher tiers. Iteration in design and prototype cycles is limited by the "shearing layer" characteristics of different disciplines and subsystems, though this can be addressed in part through prototyping techniques, agile design principles, and high level planning.

It is recommended that a mechatronics design team prefer generalization over specialization, and divide work by feature or subsystem instead of by discipline. Excellent collaboration helps a team to achieve knowledge transfer and learning, along with the continued development of skills through experience, education, and professional development.

Evaluating project characteristics is important to determine if an agile approach is appropriate, and to adapt the methodology to the specific project. Projects that are exploratory in nature with a higher level of risk and uncertainty benefit from agility more than those addressing an optimization problem with clear requirements where more planning is possible. Risk, especially technical risk, can be reduced if the team has experience with the project domain, or if there are examples in the market of solutions demonstrating feasibility. Greater criticality due to regulation or consequence of failure increases the need for formality, generally reducing the potential for agility and speed in a project. The culture of the team, the organization and the client need to be compatible with an agile approach for it to be successful. For example, a client may need to accept an alternative to a fixed bid contract, which should be avoided. The characteristics and risks of the target market and problem domain (which may include the industry or market culture) can impact the acceptability of an agile approach. A high cost of prototyping has a significant effect on the change and iteration capacity of a design project, and the team may need to adopt alternative prototyping techniques to maintain agility.

Seven key design principles are outlined by the framework: simplicity, modularity, feedback by design, customer engagement, feedback through prototyping, keeping documentation relevant, and

disciplined development. Simplicity aids in communicating and maintaining the design effectively. A team should avoid anticipatory design and reduce effort spent on features that do not add immediate value. Modularity is achieved through cohesive feature or subsystem groups, decoupling modules, leveraging standard interfaces and planning for the reuse some modules in other projects. Modularity can apply to both the design and to the process (for example, by reusing artifact templates). Integrating feedback mechanisms like sensing and logging into the design can help evaluate performance and analyze failures. Customer feedback can be obtained and integrated early by leveraging a minimum viable product. Technical risk can be reduced by prototyping early and frequently to get first-order feedback on performance and limitations of the design. A wide range of prototyping techniques can be used across different disciplines, and system integration should be considered early. It is suggested that Test-Driven Development be adopted, especially in software, though perhaps in other disciplines as well. Effective prototyping will have clear goals and methods of evaluating the prototype against these goals to help direct later development efforts. Artifacts and documentation need to be relevant and minimized to reduce effort spent creating and maintaining them. Making them easy to update through appropriate selection of media and tools will reduce the chances that they become stale and outdated. Artifacts should be dynamic or live, with regular archiving to capture the state of the design and provide a history. Artifacts need to be supported by heavy collaboration to augment their strictly explicit knowledge. The project team must remain disciplined in implementing agile design principles and methodology practices like reviews and planning in order to keep the minimal process and artifacts working and up to date.

Documentation and prototyping both support the need for feedback and knowledge transfer. This need is also supported by process feedback. It is important to know if the methodology and practices are helping the team perform. If not, the process should be adjusted to work for the team based on the core goal of delivering value. This is aided by regular retrospective activities backed by objective feedback through metrics like time tracking, budget tracking and defect tracking.

Risk management is an integral part of the development process to ensure success of the project and

product. There may be a need of formal tools based on qualitative evaluation to properly identify and mitigate high risks.

6.1.2 Methodology Model

The methodology describes a model and associated process for how a team should carry out a design project. The model integrates both design process and project management, and suggests activities and artifacts to aid the team completing the project. The methodology model presented has three stages: project initiation, project iteration and project conclusion. Effort should be balanced between initiation and iteration depending on the characteristics of the project (a more exploratory and less formal project will shift towards iteration). The iteration stage is visualized as a helix to indicate forward progress rather than repetition or redoing work as is often associated with iteration in other methods.

The goal of the project initiation stage is to build a foundation of knowledge and understanding of the project so a team can decide if it can be completed successfully. This stage requires close collaboration with the client, and the methods and expectations of collaboration throughout the project should be defined. This stage also provides the opportunity to create a set of baseline artifacts that will evolve with the project. Initial activities include gathering existing knowledge about the project along with information on project characteristics and life cycle. An initial set of technical requirements should be prepared, with particular attention paid to hard requirements like deadlines or budget caps. A project vision and scope can be defined using the goals, inputs, outputs, deliverables and definition of done. Other activities include organizing the team and assigning responsibilities like stakeholder points of contact. Developing a preliminary high level architecture can help identify areas of risk, uncertainty, or lack of competency. Finally, the team should review artifacts with the client and ensure a collective project understanding, create a mutually beneficial contract that fits with agile development, and evaluate the project together to make a go/no-go decision.

The project iteration stage is where the design is realized. Tasks and features move through a workflow managed with a kanban board, which improves feedback through visualization. The kanban board should be customized to the team, the project and the workflows associated with different features and tasks expected in the project. The general workflow model is composed of planning, execution, review and go/no-go phases. Planning activities include feature breakdown, estimation and prioritization, along with defining requirements and acceptance criteria. Execution covers requirements, analysis, high-level architecture or concept design, detailed design, implementation and testing. Testing closes the feedback loop by providing information needed to evaluate if task acceptance criteria have been met. Review of work completed may or may not include the client, and will vary in formality. A key aspect of review is tracking defects or rejections, while retrospectives help a team improve their process. The go/no-go decision compares progress against the project vision and scope to make a decision of whether to continue, complete or terminate the project. Varying levels of client collaboration may be required throughout the workflow, but it is especially important during review, go/no-go decisions and planning. Techniques for handling tasks that occur in parallel, sequentially or overlap are described with respect to the pull paradigm and review requirements. Different iteration rate tiers are discussed for reviews, prototypes, releases, retrospectives, major milestones, high level planning, and the full project, along with strategies for handling tasks or features with different iteration capacity. The methodology encourages fast iteration while balancing the overhead of collaboration, review, planning and go/no-go activities. A roadmap is suggested as a tool for high level planning, visualizing schedule requirements and synchronizing with stakeholders using time-box-driven or date-driven collaboration. Concept development is identified as an important part of the design process. This methodology suggests concept development should be applied appropriately across a range of tiers in the design, from basic tradeoff decisions within a single feature to the full development of multiple solutions as in set-based design. A minimum set of live, archivable artifacts is suggested to include a contract, a vision and scope document, a feature backlog, a roadmap, a kanban board, a budget, time and defect tracking artifacts, architecture design documents and detailed design, implementation and test source documents and artifacts. The artifacts should be adapted to the team and the project

to provide the greatest value.

The project conclusion stage is reached when a project is deemed complete by the team and the client, or the decision has been made to terminate the project. During conclusion, the team should deliver any commitments decided upon during initiation, which may vary depending on whether the project was completed or terminated. The team should prepare for the role they will play in the project as it transitions to a new phase in the product lifecycle. Preparations should also be made to begin gathering feedback on technical success and performance as a design moves through the product lifecycle. Finally, the project should be evaluated for success and failure retrospectively (regardless of completion or termination) to help the team improve in future projects.

For this methodology to be successful, the importance of continuous collaboration and an environment that supports change and adaptation can not be understated.

6.1.3 Case Studies

Two case studies were presented describing mechatronics projects that had been completed in the past. The methods used in those projects were compared to the framework and methodology described in this work, and key areas of success or shortcomings were identified. It was clear that the ad hoc approaches taken by the team in both projects had steered away from a waterfall method towards greater agility. There were several examples of principles and techniques suggested in this work improving the outcomes of the projects (for example, modular design, pull-based task distribution, and process metric tracking). There were also challenges identified that may have been addressed effectively if the teams had had access to this methodology. Examples suggested insufficient collaboration, iteration, and feedback. The observations made regarding both projects suggests there is some validity to the methodology presented within the target domain, though a more rigorous approach must be taken to properly evaluate the framework and methodology.

6.2 Future Work

The framework and methodology proposed in this work has been compared to existing projects, but not implemented in a new project. Parts of the model could be improved and several artifacts could be developed in greater detail to aid adoption. Several other areas of interest beyond the scope of this thesis were identified, including investigating design project education, exploring the relationship between traditional engineering methods and agile methods, and further exploring set-based design and systems engineering in the context of mechatronics.

The most important extension of this work will be to test and validate the methodology and framework through implementation in a real mechatronics project with appropriate characteristics. It will be important to include methods of capturing feedback and identify metrics with which to evaluate the effectiveness of the methodology prior to beginning the project. While these activities should be part of all design projects (e.g. retrospective activities suggested by this methodology), they are typically subject to the needs of the specific team. Instead, the method should be "instrumented" in a way that will decouple analysis from a specific team's process improvement goals and activities, if this is possible. A possible strategy for validating the methodology within an academic setting is described in Appendix B. Should a team be interested in adopting the proposed methodology, it is suggested that a project with appropriate characteristics be selected. Ideally, potential failure of the test project should be of little consequence to the organization. A balance should be struck between a project readily carried out with the team's existing approach and a project with high uncertainty that is well suited to an agile approach. This will help with comparison to the team's current methods. If feasible, a transition to a new method may be smoother by using the strategy of Kanban [6] and Scrumban [73], with a phased adoption of elements and practices.

Two key areas of improvement in the methodology model were identified. Integration of explicit risk assessment activities into the planning stage of the workflow would improve prioritization, while reviewing risk mitigation effectiveness and tracking cumulative project risks in the review stage would provide greater insight into project progress, process effectiveness, and the delivery of value.

The helical model of the methodology iteration is currently qualitative. By improving the detail of the model and incorporating team resources, roadmap timelines and tracked time, a quantitative version may be developed that represents project progress accurately. Different metrics could be mapped to helix characteristics like slope, diameter and thickness. Multiple paths may represent different task or resource allocations. A resulting live model may improve progress visualization for the team, especially if linked to other artifacts.

The implementation of artifacts was only briefly described, and several would benefit from detailed examples that are tightly linked to the methodology. Alternative methods of implementing the roadmap artifact could be explored. Possible examples include a matrix-type structure or a Gantt chart type document. In the case of a Gantt chart, this work recommends against using one as a schedule control tool. Instead, if a Gantt view can be integrated with time tracking and task allocation, it may provide an excellent visual feedback tool on project progress to supplement the kanban board. Furthermore, roadmap information could be integrated into a kanban tool to display work completed on tasks against estimated in progress bars, or progress of the task since start compared to a task deadline.

A set of architecture document styles and an explanation for specific use cases could be developed. Further, by actively and automatically linking architecture documents through different levels of detail along feature or subsystem breakdown paths, the design may be easier to navigate. This linking could be extended to detail design artifacts or to requirements and specifications to help implement traceability as well. Eisenbart's IFM [34] incorporates some of these goals, and should be considered as an integrated method for functional modeling and architecture artifacts. While some effort has been made to develop software support for IFM from requirements to early simulation [28], the suggested tools may not be generally suitable small organizations due to cost.

Another artifact that deserves greater attention is the feature backlog. A more advanced, flexible and comprehensive feature backlog pattern should be developed suitable for the range of tasks and subsystems present in a mechatronic system. While many tools already include a variety of useful and important information to link to a task or feature, other features that should be considered are

a method of traceability, risk assessment, multi-discipline information, and active links to a kanban, roadmap artifact, architecture artifacts and detail design artifacts. Again, it may be possible to adapt IFM to help with this task. Handling requirements may also benefit from integrating a range in attributes of interest to achieve minimal, target and outstanding results [101, 100, 102, 103] or the best and worst case specifications [42].

The possibility of adapting Test-Driven Development to disciplines outside of software was only briefly described. It would be beneficial to investigate existing methodologies or develop a new methodology for cross-disciplinary TDD and demonstrate its application in a real project.

Set-based design and systems engineering are typically targeted at large projects and organizations. An investigation into their applicability to small, agile teams and mechatronics projects is warranted.

Traditional engineering design project approaches and agile development methods have similar goals, but often have much different cultures or attitudes in the organizations and teams practicing each. One particular area of interest, identified by Jackson [49], is to understand whether the assignment of responsibility and potentially the liability for a design to an individual (i.e. a responsible APEGA member) inhibits innovation through an aversion to risk.

6.3 Summary

A design project methodology has been proposed that draws on agile principles and is suited to prototype stage mechatronic systems development being carried out by small teams. The proposed methodology was developed through the description of a framework and methodology model. A variety of techniques and practices applicable to design and management approaches in agile hardware design projects were gathered and incorporated. A preliminary discussion of validity was based on successes and challenges identified in two previous projects.

A primary trait of this methodology is the use of a feature-oriented, incremental approach in mecha-

tronics design, reflected in both design and management activities, while still maintaining sufficient flexibility to shift towards a phase-oriented design progression when appropriate. While similar to agile software methodologies, this is unique from most design methods in other disciplines, where the focus is on phases or stages. In conjunction with the phase-oriented view, it is suggested that cross-disciplinary development requires collaboration between specialist members [72] comprising large teams and organizations. Here also the proposed methodology is unique in preferring a team of generalists, which is favourable due to the domain of small teams. The feature-oriented approach is supported by techniques for rapid feedback and knowledge transfer. Three orders of feedback and knowledge transfer were identified, with prototypes providing the most direct. Prototyping methods were described that can improve agility and the capacity for change between and within domains relevant to mechatronic systems. Agile design and effective prototyping is complemented by considering the "shearing layers" in system's features and domains, as well as tiered concept development. Incremental iteration and continuously evolving artifacts suggest greater asynchronicity in the phases or stages between different features or subsystems in the project. The feature-driven approach also led to an alternative tiered interpretation of concept development. The methodology model exhibited a novel visualization using a helix to reflect continued forward progress through iteration. Key principles for agile design, team organization, project characteristics and minimal, live documentation artifacts were established. Together, the framework and methodology provide basis for implementation and extension by others.

The proposed framework and methodology model integrated agile principles, embraced the iterative nature of design, and focused on delivering value. They were designed to address the variation in projects which tend to have a significant range in characteristics, and to be flexible to accommodate different workflows or task types that are expected across the diverse disciplines that contribute to mechatronics systems. A focus was placed on feedback through visualization, prototypes and collaboration, in addition to the goal of streamlining documentation to ensure relevancy, value and currency with minimal effort. The challenges with physical products were addressed with alternative approaches to rapid prototyping that would still provide direct feedback. Several areas of future study were outlined, including the next step of implementing and formally validating the proposed

approach in a real mechatronics design project. Design process and project management are broad, continuously evolving disciplines, and methodologies can always be improved. The design project methodology proposed in this work is no exception, where continued learning and experience will lead to improvements in the future.

References

[1] *Systems engineering fundamentals*. Fort Belvoir, Virginia: Defense Aquisition University Press, January 2001.

[2] *NASA Systems Engineering Handbook*. Number NASA/SP-2007-6105. National Aeronautics and Space Administration, 1 edition, December 2007.

[3] Agile Business Consortium. Agile business consortium home page. *https://www.agilebusiness.org/*, 2016. Accessed November 18, 2016.

[4] M. Alemanni, F. Destefanis, and E. Vezzetti. Model-based definition design in the product lifecycle management scenario. *International Journal of Advanced Manufacturing Technology*, 52(1-4):1 – 14, 2011.

[5] Besiana Alite and Nikolay Spasibenko. Project suitability for agile methodologies. Master's thesis, Umeå University, 2008.

[6] David J. Anderson and Andy Carmichael. *Essential Kanban Condensed*. Seattle, Washington: Lean Kanban University Press, July 2016. Online: ISBN 978-0-9845214-2-5.

[7] Tom Arbogast, Craig Larman, and Bas Vodde. Agile contracts primer. *http://www.agilecontracts.com/*, 2012. Version 5. Accessed Feb 8, 2017.

[8] Morris Asimow. *Introduction to design*. Englewood Cliffs, N.J.: Prentice-Hall, 1962.

[9] Backblaze Inc. Application of scrum methods to hardware development. White Paper, Online, July 2015.

[10] Bird & Bird LLP. Contracting for agile software development projects. https://www.twobirds.com/~/media/pdfs/brochures/contracting-for-agile-software-development-projects.pdf?la=en, March 2016. Accessed Feb 8, 2017.

[11] Lucienne T.M. Blessing and Amaresh Chakrabarti. *DRM, a Design Research Methodology*. Springer London, 2009.

[12] Engineers Canada Accreditation Board. 2015 accreditation criteria and procedures. Online, 2015.

[13] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, May 1988.

[14] Derek Bonderczuk, Patrick Currier, and Matthew Nelson. Application of principles from the scrum agile method to a prototype vehicle control development cycle. *Proceedings of the ASME 2014 International Mechanical Engineering Congress and Exposition*, 14(IMECE2014-40018):V014T08A007, 5 pages, November 2014.

[15] Stewart Brand. *How buildings learn: what happens after they're built*. New York: Penguin Books, 1995.

[16] Frederick P. Brooks. *The mythical man-month: essays on software engineering*. Reading, Mass: Addison-Wesley Pub. Co, anniversary edition, 1995.

[17] Jacob Buur and Mogens Myrup Andreasen. Design models in mechatronic product development. *Design studies*, 10(3):155–162, 1989.

[18] Cambridge EDC. Cambridge advanced modeller. *https://www-edc.eng.cam.ac.uk/cam/*, 2014. Accessed November 18, 2016.

[19] J. Edward Carryer, R. Matthew Ohline, and Thomas William Kenny. *Introduction to mechatronic design*. Upper Saddle River: Prentice Hall, 2011.

[20] Guey-Shin Chang, Horng-Linn Perng, and Jer-Nan Juang. A review of systems engineering standards and processes. *Journal of Biomechanics Engineering*, 1(1):71–85, 2008.

[21] Robert N. Charette. This car runs on code. *http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code*, February 2009. Accessed February 5, 2017.

[22] Alistair Cockburn. *Crystal clear: a human-powered methodology for small teams*. Boston: Addison-Wesley, 2005.

[23] Copperstone Technologies Ltd. Copperstone tech home. *http://copperstonetech.com/*, 2015. Accessed November 18, 2016.

[24] Ward Cunningham. Manifesto for agile software development. *http://agilemanifesto.org/*, February 2001. Accessed August 11, 2016.

[25] Clarence W. De Silva. *Mechatronics : a foundation course*. Boca Raton: CRC Press, 2010.

[26] Steve Denning. Wikispeed: How a 100 mpg car was developed in 3 months. *http://www.forbes.com/sites/stevedenning/2012/05/10/wikispeed-how-a-100-mpg-car-was-developed-in-3-months/#3499b4a03f3e*, May 2012. Accessed August 11, 2016.

[27] George Ellwood Dieter and Linda C. Schmidt. *Engineering design*. Boston: McGraw-Hill Higher Education, 4th ed. edition, 2009.

[28] Fabio Dohr, Boris Eisenbart, Christian Huwig, Lucienne Blessing, Michael Vielhaber, et al. Software support for the consistent transition from requirements to functional modeling to system simulation. In *sings of the 10th NordDesign conference*, 2014.

[29] Clive L. Dym and Patrick Little. *Engineering design: a project-based introduction*. John Wiley & Sons, Hoboken, N.J., 3rd edition, 2009.

[30] Wolfgang Ernst Eder. Theory of technical systems - educational tool for engineering. *Universal Journal of Educational Research*, 4(6):1395 – 1405, 2016.

[31] Boris Eisenbart, Lucienne Blessing, Kilian Gericke, et al. Functional modelling perspectives across disciplines: a literature review. In *DS 70: Proceedings of DESIGN 2012, the 12th International Design Conference, Dubrovnik, Croatia*, 2012.

[32] Boris Eisenbart, Kilian Gericke, Luciënne Blessing, et al. A framework for comparing design modelling approaches across disciplines. In *DS 68-2: Proceedings of the 18th International Conference on Engineering Design (ICED 11), Impacting Society through Engineering Design, Vol. 2: Design Theory and Research Methodology, Lyngby/Copenhagen, Denmark, 15.-19.08. 2011*, 2011.

[33] Boris Eisenbart, Kilian Gericke, Luci enne Blessing, et al. A shared basis for functional modelling. In *DS 71: Proceedings of NordDesign 2012, the 9th NordDesign conference, Aarlborg University, Denmark. 22-24.08. 2012*, 2012.

[34] Boris Eisenbart, Ahmed Qureshi, Kilian Gericke, and Luciënne Blessing. Integrating different functional modeling perspectives. In *ICoRD'13*, pages 85–97. Springer India, 2013.

[35] Atila Ertas and Jesse C. Jones. *The engineering design process.* New York: John Wiley & Sons, 2nd todo check edition, 1996.

[36] Ralph A. Evans. Engineering design handbook: development guide for reliability part two design for reliability. *http://www.dtic.mil/dtic/tr/fulltext/u2/a027370.pdf*, (AMCP-706-196), 1976. Publisher: US Army Materiel Command.

[37] B. Fitzgerald, K. J. Stol, R. O'Sullivan, and D. O'Brien. Scaling agile methods to regulated environments: An industry case study. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 863–872, May 2013.

[38] Brian Foote and Joseph Yoder. Big ball of mud. `http://www.laputan.org/mud/`, June 1999.

[39] Kilian Gericke, Ahmed Jawad Qureshi, and Lucienne Blessing. Analyzing transdisciplinary design processes in industry: An overview. In *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V005T06A031–V005T06A031. American Society of Mechanical Engineers, 2013.

[40] James W Grenning. *Test-driven development for embedded C*. Pragmatic Bookshelf, 2011.

[41] Abdelaziz Guerrouat and Harald Richter. A component-based specification approach for embedded systems using fdts. In *Proceedings of the 2005 Conference on Specification and Verification of Component-based Systems*, SAVCBS '05, New York, NY, USA, 2005. ACM.

[42] Reinhard Haberfellner and Olivier de Weck. 10.1.3 agile systems engineering versus agile systems engineering. *INCOSE International Symposium*, 15(1):1449–1465, 2005.

[43] Yousef Haik and Tamer M. Shahin. *Engineering design process*. Stamford, CT: Cengage Learning, 2nd edition, 2011.

[44] James A. Highsmith. *Agile software development ecosystems*. Boston: Addison-Wesley, 2002.

[45] Percy H. Hill. *The Science of Engineering Design*. New York: Holt, Rinehart and Winston, 1970.

[46] Project Management Institute. What is project management? *https://www.pmi.org/about/learn-about-pmi/what-is-project-management*, 2016. Accessed September 9, 2016.

[47] iRobot Corporation. irobot home robots website. *http://www.irobot.com/*, 2017. Accessed February 8, 2017.

[48] ISO/IEC. Information technology—Open System Interconnection—Basic Reference Model: The Basic Model. *International Standard*, (ISO/IEC 7498-1:1994(E)), November 1994.

[49] Chad Jackson. Agile development for mechatronics products? *http://www.lifecycleinsights.com/engineering-management/agile-analogues/*, July 2011. Accessed August 9, 2016.

[50] Tomas Kellner. The faa cleared the first 3d printed part to fly in a commercial jet engine from ge. *http://www.gereports.com/post/116402870270/the-faa-cleared-the-first-3d-printed-part-to-fly/*, April 2015. Accessed February 9, 2017.

[51] Corey Ladas. Scrum-ban. *http://leansoftwareengineering.com/ksse/scrum-ban/*, July 2008. Accessed November 7, 2016.

[52] M. G. Lipsett, J. D. Yuen, N. A. Olmedo, and S. C. Dwyer. Condition monitoring of remote industrial installations using robotic systems. In *Proc. World Congress on Engineering Asset Management (WCEAM) Cincinatti*, October 2011.

[53] Larry Maccherone. Top 10 questions when using agile on hardware projects. *http://maccherone.com/larry/2010/02/23/top-10-questions-when-using-agile-on-hardware-projects/*, February 2010. Accessed August 9, 2016.

[54] Debbie Madden. I'm agile but my contract isn't: How to align contracts with agile software development teams. *http://blog.stridenyc.com/blog/im-agile-but-my-contract-isnt-how-to-align-contracts-with-agile-software-development-teams/*, October 2014. Accessed February 8, 2017.

[55] Davide Maritan. *Practical manual of quality function deployment*. Springer, 2015.

[56] Bertrand Meyer. Applying 'design by contract'. *Computer*, 25(10):40–51, Oct 1992.

[57] Geoffrey Moore. *Inside the tornado: marketing strategies from Silicon Valley's cutting edge*. New York: HarperBusiness, 1995.

[58] Thor Myklebust and SINTEF. Safescrum - sintef. *http://www.sintef.no/SafeScrum#/*, 2017. Accessed March 28, 2017.

[59] Bent Myllerup. Why agile does matter in an embedded development environment. *https://www.scrumalliance.org/community/articles/2011/march/*

why-agile-does-matter-in-an-embedded-development-e, March 2011. Accessed August 11, 2016.

[60] Tammy Noergaard. *Embedded systems architecture : a comprehensive guide for engineers and programmers*. Oxford: Newnes, 2nd edition, 2013.

[61] N. A. Olmedo and M. G. Lipsett. Design and field experimentation of a robotic system for tailings characterization. *J. Unmanned Vehicle Systems*, March 2016.

[62] Nicolas A. Olmedo, Stephen Dwyer, Jamie Yuen, and Michael Lipsett. Amphibious robot for environmental monitoring of oil sands tailings. In *Proceedings of IOSTC 2016*, December 2016.

[63] Oxford University Press. Definition of quality in english. *https://en.oxforddictionaries.com/definition/quality*, 2016. Accessed September 7, 2016.

[64] Oxford University Press. Definition of risk in english. *https://en.oxforddictionaries.com/definition/risk*, 2016. Accessed September 7, 2016.

[65] Oxford University Press. Definition of scope in english. *https://en.oxforddictionaries.com/definition/scope*, 2016. Accessed September 7, 2016.

[66] Oxford University Press. Definition of uncertain in english. *https://en.oxforddictionaries.com/definition/uncertain*, 2016. Accessed September 7, 2016.

[67] G. Pahl, W. Beitz, J. Feldhusen, and K. H. Grote. *Engineering design: a systematic approach*. London: Springer, 3rd edition, 2007.

[68] P.K. Palani Rajan, Michael Van Wie, Matthew I. Campbell, Kristin L. Wood, and Kevin N. Otto. An empirical foundation for product flexibility. *Design Studies*, 26(4):405 – 438, 2005.

[69] James K. Peckol. *Embedded systems: a contemporary design tool*. Hoboken, N.J.: John Wiley & Sons, 2008.

[70] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[71] Ahmed Jawad Qureshi, Kilian Gericke, and Lucienne Blessing. Design process commonalities in trans-disciplinary design. In *International Conference on Engineering Design*, 2013.

[72] Ahmed Jawad Qureshi, Kilian Gericke, and Lucienne Blessing. Stages in product lifecycle: Trans-disciplinary design context. *Procedia CIRP*, 21:224–229, 2014.

[73] Ajay Reddy. *The Scrumban [r]evolution: getting the most out of Agile, Scrum, and lean Kanban*. New York: Addison-Wesley, 2016.

[74] Þórdís Reynisdóttir. Scrum in mechanical product development. Master's thesis, Chalmers University of Technology, 2013.

[75] Ben R. Rich and Leo Janos. *Skunk Works: a personal memoir of my years at Lockheed*. Boston: Little, Brown, 1st edition, 1994.

[76] Eric Ries. *The lean startup: how today's entrepreneurs use continuous innovation to create radically successful businesses*. New York: Crown Business, 1st edition, 2011.

[77] Johanna Rothman. Helping hardware be agile, part 1. *http://www.jrothman.com/mpd/agile/2015/12/helping-hardware-be-agile-part-1/*, December 2015. Accessed August 12, 2016.

[78] Johanna Rothman. Helping hardware be agile, part 2. *http://www.jrothman.com/mpd/agile/2015/12/helping-hardware-be-agile-part-2/*, December 2015. Accessed August 12, 2016.

[79] Johanna Rothman. Helping hardware be agile, part 3. *http://www.jrothman.com/mpd/agile/2015/12/helping-hardware-be-agile-part-3/*, December 2015. Accessed August 12, 2016.

[80] Michael John Safoutin. *A methodology for empirical measurement of iteration in engineering design processes*. PhD thesis, University of Washington, 2003.

[81] K. J. Schlager. Systems engineering-key to modern development. *IRE Transactions on Engineering Management*, EM-3(3):64–66, July 1956.

[82] Anja Schulze. Developing products with set-based design: How to set up an idea portfolio and a team organization to establish design feasibility. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 30(3):235–249, Aug 2016.

[83] David J. Singer, Norbert Doerry, and Michael E. Buckley. What is set-based design? *Naval Engineers Journal*, 121(4):31–43, 2009.

[84] Zhe Song, Ziyuan Zhang, Xiaolin Xu, and Chunlin Liu. An agent-based model to study the market dynamics of perpetual and subscription licensing. *Journal of the Operational Research Society*, 66(5):845–857, 2015.

[85] Space Exploration Technologies Corp. Spacex launches 3d-printed part to space, creates printed engine chamber. *http://www.spacex.com/news/2014/07/31/spacex-launches-3d-printed-part-space-creates-printed-engine-chamber-crewed*, July 2014. Accessed February 9, 2017.

[86] Peter Stevens. 10 contract forms for your next agile project. Presentation Slides, `https://www.scrumalliance.org/resource_download/1119`, October 2009. Accessed February 8, 2017.

[87] Samuel Suss. *Coordination in complex product development*. PhD thesis, McGill University, 2011.

[88] The Association of Professional Engineers and Geoscientists of Alberta. Guideline for management of risk in professional practice v1.0. Online, September 2006.

[89] The Association of Professional Engineers and Geoscientists of Alberta. Guideline for professional responsibilities in developing software v1.0. Online, February 2006.

[90] The Association of Professional Engineers and Geoscientists of Alberta. Guideline for professional practice management plans v1.4. Online, February 2013.

[91] The Association of Professional Engineers and Geoscientists of Alberta. Practice standard for authenticating professional documents v3.1. Online, January 2013.

[92] The Association of Professional Engineers and Geoscientists of Alberta. About apega. *https://www.apega.ca/about-apega/*, 2016. Accessed November 17, 2016.

[93] Jean Thilmany. Can mechanical engineers adopt agile product development? *http://www.lifecycleinsights.com/engineering-profession/can-mechanical-engineers-adopt-agile-product-development/*, May 2015. Accessed August 9, 2016.

[94] U. S. Department of Defense. Electronic reliability design handbook. *http://www.everyspec.com/MIL-HDBK/MIL-HDBK-0300-0499/MIL-HDBK-338B_15041/*, (MIL-HDBK-338B), October 1998.

[95] USB Implementers Forum, Inc. Universal serial bus. *http://www.usb.org/home*, 2016. Accessed November 17, 2016.

[96] Gerald M. Weinberg. *Quality software management (Vol. 1): systems thinking*. Dorset House Publishing Co., Inc., 1992.

[97] Dr. Edward White. You have a jedi sword. *http://embedded.fm/episodes/2013/7/31/you-have-a-jedi-sword*, July 2013. Online: Podcast. Accessed September 6, 2016.

[98] Elecia White. *Making embedded systems*. Beijing: O'Reilly, 2011.

[99] Rebecca Wirfs-Brock. Agile architecture myths #4 because you are agile you can change your system fast! *http://wirfs-brock.com/blog/2011/08/26/agile-architecture-myths-4-because-you-are-agile-you-can-change-your-system-fast/*, August 2011. Accessed August 22, 2016.

[100] Rebecca Wirfs-Brock. Agile landing zones. *http://wirfs-brock.com/blog/2011/07/28/agile-landing-zones/*, July 2011. Accessed August 22, 2016.

[101] Rebecca Wirfs-Brock. Introducing landing zones. *http://wirfs-brock.com/blog/2011/07/20/introducing-landing-zones/*, July 2011. Accessed August 12, 2016.

[102] Rebecca Wirfs-Brock. Landing zone targets: Precision, specificity, and wiggle room. *http://wirfs-brock.com/blog/2011/08/05/landing-zone-targets-precision-specificity-and-wiggle-room/*, August 2011. Accessed August 22, 2016.

[103] Rebecca Wirfs-Brock. Who defines (or redefines) landing zone criteria? *http://wirfs-brock.com/blog/2011/08/16/who-defines-or-redefines-landing-zone-criteria/*, August 2011. Accessed August 22, 2016.

[104] David Charles Wynn. *Model-based approaches to support process improvement in complex product development*. PhD thesis, University of Cambridge, 2007.

[105] Yaskawa Electric Corporation. Yaskawa technical history. *http://www.yaskawa.co.jp/en/technology/history*, 2016. Accessed September 6, 2016.

[106] Wang Ying. Dji sees jump in revenue. *http://www.chinadaily.com.cn/business/tech/2016-12/13/content_27649387.htm*, December 2016. Accessed February 5, 2017.

Appendix A

Vendor, Software Tool, and Framework/Ecosystem Lists

A sample list of vendors relevant to mechatronics prototyping is presented in Table A.1. A key for vendor tags can be found in Table A.4.

A sample list of software tools for project management and design activities is presented in Table A.2. A key for software types and tags can be found in Tables A.5 and A.6, respectively.

A sample list of frameworks and ecosystems that can be leveraged for different subsystems in mechatronics projects is presented in Table A.3. A key for framework and ecosystem tags can be found in Table A.7.

These tables have been adapted from a Copperstone Technologies Ltd. document with permission. Copperstone Technologies Ltd. does not necessarily endorse the organizations or products in these tables, nor reject any not present in these tables.

Table A.1: Sample List of Vendors

Vendor Name	Website	Location	Tags
Adafruit	https://www.adafruit.com/	New York	Elec, Tools
Sparkfun	https://www.sparkfun.com/	Colorado	Elec, Tools
Elmwood Electronics	https://elmwoodelectronics.ca/	Ontario	Elec
Solarbotics	https://solarbotics.com/	Calgary	Elec
RobotShop	http://www.robotshop.com/ca/	Quebec	Robotics, Elec
SeeedStudio	https://www.seeedstudio.com/	China	Elec, PCB, PCBA
Parallax	https://www.parallax.com/	California	Elec, Robotics
Olimex	https://www.olimex.com/	Bulgaria	Elec
Pololu	https://www.pololu.com/	Nevada	Elec, Robotics, Mech, Tools
ITEAD	https://www.itead.cc/	China	Elec
Great Hobbies	https://www.greathobbies.com/	Edmonton	Robotics, Mech, Tools
Digikey	http://www.digikey.ca/	Minnesota	Elec
Mouser	http://ca.mouser.com/	Texas	Elec
Newark/Element14	http://canada.newark.com/	Indiana	Elec
Octopart	https://octopart.com/	-	Elec, Meta
McMaster-Carr	https://www.mcmaster.com/	USA	Mech, Tools, Raw
Metal Supermarkets	https://www.metalsupermarkets.com/	Edm/Calgary	Raw
Gregg's Distributors	https://www.greggdistributors.ca/	Edm/Calgary	Tools, Supply
AP Circuits	https://www.apcircuits.com/	Calgary	PCB
OSHPark	https://oshpark.com/	Oregon	PCB
OSH Stencils	https://www.oshstencils.com/	Utah	Stencil
PCBShopper	http://pcbshopper.com/	-	PCB, Meta
PCB:NG	http://www.pcb.ng/	New York	PCB, PCBA
MacroFab	https://macrofab.com/	Texas	PCB, PCBA
proto labs	https://www.protolabs.com/	USA	CNC, 3DPrint
Xometry	https://www.xometry.com/	Maryland	CNC, 3DPrint
Shapeways	https://www.shapeways.com/	New York	3DPrint
Tindie	https://www.tindie.com/	-	Elec
80/20 Inc.	https://www.8020.net/	Indiana	Mech

Table A.2: Sample List of Software Tools

Software Tool	Website	Type	Tags
Gmail	https://www.google.com/gmail/	Browser, App	Collab
Google Drive	https://www.google.com/drive/	Browser, Desktop, App	Collab, Doc
Google Hangouts	https://hangouts.google.com/	Browser, App	Collab
G Suite	https://gsuite.google.com/	Browser, Desktop, App	Plan, Collab, Doc
Slack	https://slack.com/	Browser, Desktop, App	Collab
Hipchat	https://www.hipchat.com/	Browser, Desktop, App	Collab
Skype	https://www.skype.com/en/	Desktop, App	Collab
Dropbox	https://www.dropbox.com/	Browser, Desktop, App	Collab, VCS
Toggl	https://toggl.com/	Browser, App, Desktop	Track
SolidWorks	http://www.solidworks.com/	Win	ME, Sim
Onshape	https://www.onshape.com/	Browser, App	ME, Collab
GrabCAD	https://grabcad.com/	Browser, App	ME, Collab
Eagle	http://www.autodesk.com/products/eagle/overview	Mac, Win, Nix	EE
KiCad EDA	http://kicad-pcb.org/	Mac, Win, Nix	EE
Altium Designer	http://www.altium.com/altium-designer/overview	Win	EE
GitHub	https://github.com/	Browser, Desktop, App	VCS, Collab, Track
BitBucket	https://bitbucket.org/product	Browser, Desktop, App	VCS, Collab, Track
Pivotal Tracker	https://www.pivotaltracker.com/	Browser, App	Plan, Collab, Track
JIRA	https://www.atlassian.com/software/jira	Browser, App	Plan, Collab, Track
Trello	https://trello.com/	Browser, App	Plan, Collab, Track
Taiga	https://taiga.io/	Browser	Plan, Collab, Track
Sublime Text	https://www.sublimetext.com/	Mac, Win, Nix	SW
Atom	https://atom.io/	Mac, Win, Nix	SW
iTerm	https://www.iterm2.com/	Mac	SW
Homebrew	https://brew.sh/	Mac	SW
Virtualbox	https://www.virtualbox.org/	Mac, Win, Nix	SW
Vagrant	https://www.vagrantup.com/	Mac, Win, Nix	SW
Git	https://git-scm.com/	Mac, Win, Nix	VCS
Subversion	https://subversion.apache.org/	Mac, Win, Nix	VCS
OmniGraffle	https://www.omnigroup.com/omnigraffle	Mac	Doc
OmniPlan	https://www.omnigroup.com/omniplan	Mac	Plan
LTspice	http://www.linear.com/designtools/software/#LTspice	Mac, Win	Sim, EE
Falstad Circuit Sim	http://www.falstad.com/circuit/	Browser	Sim, EE
Ngspice	http://ngspice.sourceforge.net/	Mac, Win, Nix	Sim, EE
Swift Calcs	https://www.swiftcalcs.com/	Browser	Sim
SimScale	https://www.simscale.com/	Browser	Sim, ME
OpenFOAM	https://openfoam.org/	Mac, Win, Nix	Sim, ME
ANSYS	http://www.ansys.com/	Win, Nix	Sim, ME, EE, FW
LISA	http://www.lisafea.com/	Win	Sim, ME

Table A.3: Sample List of Frameworks and Ecosystems

Framework/Ecosystem	Website	Tags
Arduino	https://www.arduino.cc/	FW, EE
mbed	https://developer.mbed.org/	FW, EE
Raspberry Pi	https://www.raspberrypi.org/	FW, SW, EE
Django	https://www.djangoproject.com/	SW
Unity (TDD)	http://www.throwtheswitch.org/unity/	FW, Sim
ChibiOS/RT	http://chibios.org/dokuwiki/doku.php	FW
FreeRTOS	http://www.freertos.org/	FW
wxWidgets	http://www.wxwidgets.org/	SW
Unity (3D Engine)	https://unity3d.com/	SW
Qt	https://www.qt.io/	SW
Google App Engine	https://cloud.google.com/appengine/	SW
Heroku	https://www.heroku.com/	SW
Particle	https://www.particle.io/	FW, SW, EE

Table A.4: Vendor Tags

Tag	Description
Elec	Any sort of electronics parts, assemblies, prototyping or breakouts. Not differentiating parts and assemblies (i.e. Digikey and Sparkfun)
Robotics	Robot specific components, parts, etc, including commercial systems
Mech	Any sort of mechanical components or subsystems. Not differentiating parts and assemblies.
PCB	PCB manufacturing
PCBA	PCB assembly
Stencil	Solder paste stencils
Meta	Search or comparison service
CNC	CNC machining services
3DPrint	3D printing services
Tools	Tools and/or consumables
Raw	Raw materials (metals, plastics, etc.)
Supply	Industrial supply

Table A.5: Software Types

Tag	Description
Browser	Browser based
Mac	Mac desktop app
Win	Windows desktop app
Nix	Linux or similar desktop app
iOS	iPhone/iPad app
Android	Android app
Desktop	All desktop apps
App	All mobile apps
Tag	Description

Table A.6: Software Tags

Tag	Description
Collab	Collaboration tools (project management)
Plan	Planning tools (project management)
Track	Tracking tools (project management)
Doc	Documentation tools
VCS	Version control system
SW	Software development tools
FW	Firmware development tools
ME	Mechanical design tools
EE	Electrical/electronics design tools
Sim	Simulation tools

Table A.7: Framework and Ecosystem Tags

Tag	Description
SW	Software development tools
FW	Firmware development tools
ME	Mechanical design tools
EE	Electrical/electronics design tools
Sim	Simulation tools

Appendix B

Suggested Methodology Verification and Validation Strategy

B.1 Scope

The next step for the proposed framework and methodology model is to verify and validate in the context of real projects. This should be carried out as a two stage empirical study, the first evaluating the application of the methodology to a project, and the second to evaluate the success of the methodology in improving design project outcomes. Both quantitative and qualitative data should be collected and analyzed. Based on feedback from the initial evaluation or verification stage, improvements can be incorporated into the methodology, data collection techniques can be fine tuned, and the introduction approach can be fully developed. The second stage should ideally be a comparative validation. It is suggested that the unique characteristics of an academic environment be leveraged for the study. The quantity and type of data collection may prove challenging, and careful study design is necessary due to the definite subjective and human factor components to many aspects of the methodology and the evaluation process.

B.2 Initial Application Study

In the first stage, the methodology should be verified by applying it to a project and evaluating this initial application. Key areas of evaluation include the completeness of the methodology description, the approach to introducing the methodology, and which aspects of the methodology can be measured effectively with what techniques. Implementation improvements can be identified and the methodology revised appropriately.

An appropriate context for this study would be a small team of graduate students working together on designing and prototyping an experimental setup for use in a laboratory environment. Ideally, the project will incorporate cross-disciplinary mechatronic functionality with sensing, control and actuation. The project timeline should last several months to allow sufficient evaluation time and completion of a useful project with appropriate complexity, without detracting from other commitments.

It is suggested the team be composed of several graduate students. Ideally, the primary researcher will be involved, who may then translate the experience to the comparative study. The researcher should focus on documenting and implementing the methodology during the project. It may be useful to have members both with and without the design training that will be used as a control case in the comparative study. It is suggested that the acting project client (another graduate student) not be explicitly included in the design team, though this individual should understand the purpose of the project, be prepared to collaborate regularly and may be from the same research team or an external research team.

Prior to and throughout the project, the primary researcher should prepare and test artifact templates and data collection tools that will be used again during the comparative study stage. Other members of the team should be aware of the verification nature of the study and work to provide feedback and insights. Sample data should be analyzed after the project (or perhaps throughout) to understand if the data collection and analysis techniques are effective.

B.3 Comparative Success Study

The second stage comparative study should be used to validate the methodology using measurable criteria to determine if it makes an impact and improves outcomes in design projects. Based on the methodology description, the study should measure some or all of the following:

- Ability to deliver value: is the client happy and feeling like value is being delivered, both throughout the project and after the project conclusion?
- Impact of visualization and feedback: are techniques helping the team and the client?
- Ability to deliver or iterate prototypes more quickly: is the team able to leverage rapid prototypes that provide value for feedback for both the team and the client? Do they have a better understanding that can improve decision making and design?
    - Quantity, speed, time, cost, type, portion of design, and other characteristics
    - Number of technical uncertainties addressed by prototyping
- Effectiveness of documentation: is less time spent on documentation, is the documentation proving valuable, and is the documentation easier to manage?
    - Balance of time spent on initial creation and on revisions or improvements
    - Quantity of documentation
- Impact of alternative project management artifacts: are the alternatives working?
- Ability to adapt to change with agility: can the team manage uncertainty and changes at different times during the project?
    - Manipulated through intentional ambiguity in or changes to requirements during the project
- Ability to support a feature-driven, incremental approach: is the team able to maintain a feature-oriented progression or do they fall back to a phase-oriented progression?
    - Task definition, selection, and completion, and long term planning approach
    - Time and effort balance between planning, review and technical execution
    - Iteration characteristics at different levels, quantity and duration

The data collection tools and coverage should be able to provide the data necessary for addressing these topics. Care must be taken in determining appropriate measurement criteria that can be compared against a performance standard and the control methodology. Considerations must be made as to the data collection approach in the control group that allows valid, direct comparisons between the two methodologies.

It is suggested that undergraduate or graduate design courses that are project-based will provide an excellent framework for this study. A good example is undergraduate capstone design courses. Generally, projects in these courses are carried out in a reasonable timeframe (one or two semesters) by small teams (3 to 6 members). A limitation is the strict schedule and (possibly) budget requirements, which needs to be explored. Another limitation is the lack of cross-disciplinary design courses, through projects presented may incorporate some aspects of different disciplines. For example, electrical engineering capstone projects may incorporate electronics, firmware and software elements. Dividing teams within or between classes (i.e. different semesters for a 1 semester course) will allow for a control group that is taught the current methodology and a study group that is taught the new methodology. It may be necessary to explore optional courses instead, as there may be ethical concerns in providing different methodology instruction to different students in a required course. The approach to introducing the methodology to teams must be considered. The course curriculum and organization may need to be adapted to focus on methodology instruction early, and to present to the control and study groups appropriately. One or more researchers may be embedded into the course as teaching assistants to actively aid students in applying the methodology or using tools.

An appropriate project or set of projects will have to be prepared. Typically, capstone design projects are unique, but this may introduce too much variation for the study to be effective, and a single project may be more appropriate. Alternatively, the projects' characteristics can be carefully recorded and analyzed to account for potential variation.

Another area of interest is to understand the performance of the team working in the project. The proposed framework and methodology are built on the assumption of a high performance team that

collaborates effectively. It may take some time for a new team to transition to this performance phase. This is likely to impact the effectiveness both the control and new methodology, and should be considered during data collection and analysis.

Data collection can be accomplished in multiple ways, with some recommendations below. If possible, many of these data collection techniques may be included as part of team retrospective activities. Data should be collected from both team members and clients.

Artifacts Teams may be required to complete certain deliverables based on clear, standardized artifact templates that are designed to provide both feedback to the team and usable data to the researcher. By capturing archive snapshots of these artifacts throughout the project, quantitative time series data can be collected. For example, regular and detailed time tracking can show the time spent on documentation efforts. Key process metrics to record include time, defects or changes, and budget. Records of review and meeting acceptance criteria may also be required. It is noted the methodology encourages artifact customization, but this may need to be limited to provide an efficient method of data collection. Final design documents and prototypes may also provide useful information about utility and the success of the design solution. Alternative or equivalent data collection may need to be considered for the control case.

Surveys Regular surveys should be completed to understand the impressions, feelings and attitudes of both team members and clients. Examples of survey coverage include:

- Perceived delivery of value
- Perceived effectiveness or effort of methodology or certain elements or tools
- Perceived team performance and collaboration levels
- Perceived levels of stress
- Current external commitments or impacts

Survey question design requires care, and appropriate scale selection can make results more quantifiable. Ideally results of survey can be used by team to improve their own process as well, though the impact of this on the methodology should be considered (i.e. will the surveys need to be made part of the methodology?).

Interviews Interviews can provide additional qualitative insight. Short, pointed questions similar to surveys will reduce time commitment, but allow for additional explanation or expansion on the part of both the researcher and the interviewee. They will also help address inconsistencies and verify survey data, while providing the ability to explore new directions or insights.

Observations Researchers embedded as teaching assistants or separately may observe team behaviour during in-class or in-lab work. While extensive observations will prove logistically challenging, even a partial dataset may provide useful insight.

Participants in the study should be evaluated pre- and post-project. Before the project begins, an understanding of characteristics like previous experience, culture or personality tendencies, preferences for analytical or experimental methods, and attitudes towards the methodology and project should be captured. Following the project, a final exit survey and interview should be conducted.

B.4 Alternative or Supplemental Success Study

As an alternative to or supplemental to an academic course-based study, the methodology could be introduced to and evaluated with a small development team working in industry, similar to the organization of the author or as a unit in a large organization. This may be helpful in determining suitability in a commercial environment, where the motivations of participants will be different than an academic environment.

Another potential study domain is to collaborate with extracurricular design project teams in the

university community. Again, these teams will have different constraints and motivations than either within courses or within industry.

B.5 Other Considerations

Due to the broad coverage of the methodology, it may be necessary to study only a subset through phased introduction of elements across several study iterations. The post-project feedback and evaluation may be extended beyond the timeline of the project to students that have become employed with design-oriented positions in industry to understand how the specific methodology education may have had an impact. The role of the original researcher that proposed the methodology in any evaluation studies should be considered to avoid introduction of any bias. Finally, prior to any additional research efforts on this methodology, the work "DRM, a Design Research Methodology", by Blessing and Chakrabarti, should be consulted [11]. This book provides excellent guidelines for research methods in engineering and industrial design, and influenced the validation study strategy presented here, which is merely a starting point.