University of Alberta

How to integrate object-oriented methodology with QFD-style matrices

By

Yunbo Zhou 

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

Department of Electrical and Computer Engineering

Edmonton, Alberta

Fall 2004

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-95890-6
Our file Notre référence
ISBN: 0-612-95890-6

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table of Contents

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Tables

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Figures

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.0 Introduction

Software engineering theory and practice are essential for understanding the construction of good software and for evaluating the risk and opportunities that software present in our everyday lives. In fact, most organizations might have difficulties in implementing many software projects. They always face problems in the development of software for example, over-budget, overdue, and low quality. All the problems affect the development of the software project without doubt.

RUP, as a software design process, has been used in a wide variety of projects and organizations. It unifies the entire software development team and optimizes the productivity of team members by bringing them the experience of industry leaders and the lessons learned from thousands of projects. It provides detailed and practical guidance through all phases of the software development life cycle.

Therefore, this method can be used to produce a predictable schedule, budget, and high-quality software by adopting the industry-standard Unified Modeling Language (UML) and other industry best practices.

The UML is the standard language for specifying, visualizing, constructing, and documenting all the artifacts of a software system. UML can be used with all processes throughout the development life cycle and across different implementation technologies to deal with its static structures and dynamic behaviors.

However, these two methods mentioned above still have some limitations when applied in software development domain even though they have been the general-purpose standard techniques. So, in this thesis, we try to employ QFD (Quality Function Deployment) and other effective methods to enhance RUP and UML so that we can implement high quality software and avoid failures of software project.

In this thesis, we mainly apply our new techniques in the initial phase of software development, i.e., customer requirement analysis.

The rest of this thesis is organized as follows: Section 2 introduces Unified Modeling Language (UML) and Rational Unified Process (RUP). Section 3 explains the techniques and methods that will be adopted in the thesis. Section 4 illustrates the reasons to apply QFD in software and its benefit. Section 5 describes the new structure method. Section 6 points out the limitations of UML and RUP, and illustrates how to solve the problems in UML and RUP. Section 7 implements a new process and description. Section 8 uses these two methods mentioned above in a case study (an elevator system) to illustrate the approach. Section 9 concludes the thesis.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.0 UML AND RUP

Some customers may complain that their software does not meet their requirements, even though software designers have tried their best and spent much time in completing the project. On the other hand, the designers may think that they have understood the requirements of the customers, and the problem is that the customers didn't express their needs very clearly and they always change their mind. To sum up, all the problems are derived from different standpoints between the designers and customers.

So what on earth is the problem? It is because most of the design is not customer oriented but program oriented so that it causes the gulf between the customers and designers. However, there is one language that can solve this problem, i.e., UML (Unified Modeling Language).

2.1UML(Unified Modeling Language)

The Unified Modeling Language (UML) is a standard language for writing software blueprints. Three prominent object-oriented programming professionals, Gray Booch, Ivar Jacobsen, and James Rumbaugh are the principal authors of UML.

Back in the late 80s, there were many different methodologies. And each methodology had its own notations. The problem was that if different people were using different notations, somewhere along the line somebody had to do a translation. A lot of times, one symbol meant one thing in one notation, and something totally different in another notation. (Terry 1999)

In 1991, everybody started coming out with books. Grady Booch came out with his first edition. Ivar Jacobson came out with his, and Jim Rumbaugh came out with his OMT methodology. Each book had its strengths as well as its weaknesses. OMT was really strong in analysis, but weaker in design. The Booch methodology was stronger in design and weaker in analysis. And Ivar Jacobson's Objectory was really good with user experience, which neither Booch nor OMT really took into consideration back then.

Then, in 1993, a funny thing happened. Grady came out with the second edition of his first book; it still had the good design, but some of the good analysis stuff from OMT had started creeping into his methodology. And actors and use cases from Ivar were in there as well. And Jim was writing a series of articles for the Journal of Object Oriented Programming that people referred to as OMT 2, which still had the good analysis work, plus some of Grady's good designs, and all of a sudden actors and use cases were added into OMT 2. That was the beginning of the informal unification of methodology. And it came as something of a relief, because it really had been a method war.

In 1995, Rumbaugh and then Jacobsen joined booch at Rational, and started developing an enhanced integrated version of their earlier work. It started being called the "Unified Method". It was submitted to the OMG. Within the OMG, a working party was formed to define a standard OOA&D modeling approach. The scope was

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

restricted to a set of models and modeling tools (including their semantics and syntax). The first drafts were produced in 1996 and a submission on model semantics & syntax put to the OMG in September 1997(Graham 1991).

UML (Unified Modeling Language) is a modeling language using text and graphical notations for documenting specification, analysis, design and implementation of an OOSD (Object-Oriented System Development) Process. One early indication that UML is well suited for this task is that many of the concepts in UML previously existed in object-oriented programming. One part of object-oriented programming is object-oriented analysis. Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain (Rumbaugh 1999). An object is something you can do things to (Rumbaugh 1999). A class is a set of objects that share a common structure and a common behavior (Rumbaugh 1999). The significance of object-oriented analysis is to build objects that directly represent things in the physical world. Other techniques try to capture information and translate that information into some kind of container that is well suited for constructing software but may not exist outside of the software. Object-oriented analysis equates a real entity to a corresponding software entity as closely as possible. UML can be used with all processes throughout the development life cycle and across different implementation technologies to deal with its static structures and dynamic behaviors. (Amatya 1999)
Success has far exceeded this goal because:

- Political:UML developed by a consortium led by three leaders in OOA/OOD; wide acceptance among software professionals; respect and confidence by the majority of software industry.
- Marketing: Submitting UML specification to a standardization process within Object Management Group (OMG). OMG has over 900 member organizations; UML is perceived as an open and widely supported standard.
- Technical: Concentrated on a standard modeling language, not a standard modeling method; provides common notation.

2.1.1 Modeling elements and tools

UML establishes a collection of graphical symbols as well as semantics to support and define these symbols. This collection can be broken down into three kinds of building blocks: things, relationships, and diagrams. Things are the abstractions that are first-class citizens in a model; relationships bind these things together; diagrams group interesting collections of things. There are nine different kinds of diagrams in UML: class, object, use case, sequence, collaboration, state chart, activity, component, and deployment (Booch 1999).

2.1.1.1Modelling Elements

UML provides various basic elements for building models; basic elements may be grouped into composite elements. Relational elements deal with various kinds of relationships between the model elements. Other elements are there to describe object states and interactions. Annotations are provided for clarifying the meaning.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(Rumbaugh 1999)

- Basic elements
- Composite elements
- Relationships between elements
- States and Interactions
- Annotations

Modeling Mechanisms: Specifications are saved in files to document responsibilities and capabilities of the model elements. Adornments make symbols mean specific things.

- Adornments
- Specifications
- Modeling Rules

Notation Extending Mechanisms: In order to cover every possible situation UML provides notation-extending mechanisms. One such mechanism is stereotyping which specializes in the general notation to specific application areas. Standard stereotypes and icons are provided, though domain specific stereotypes and icons may be introduced as and when required. Notation extension uses tagged values to add more information. Constraints are used to show restrictions that apply. (Berner 2000)

- Stereotypes
- Tagged Values
- Constraints

2.1.1.2Modeling tools

Class Diagram

UML class diagrams are the mainstays of object-oriented analysis and design. UML class diagrams show the classes of the system, their interrelationships (including inheritance, aggregation, and association), and the operations and attributes of the classes. Class diagrams are used for a wide variety of purposes, including both conceptual/domain modeling and detailed design modeling (Booch 1999).

Package Diagram

Package diagrams provide a mechanism for dividing and grouping model elements (e.g., classes, use cases). In UML, a package is represented as a folder:

- In effect, a package provides a namespace such that two different elements in two different packages can have the same name.
- Packages may be nested within other packages.
- Dependencies between two packages reflect dependencies between any two classes in the packages. For example, if a class in Package A uses the services of a class in Package B, Package A is dependent on Package B. An important design consideration is the minimization of dependencies

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

between packages

Use Case Diagram

A use case is a typical way a stakeholder might want to use the system. It is high-level function the system must support. This name reflects a historical bias towards thinking of analysis as asking: What functions must this software support? I.e. how will people want to use it? Some stakeholders don't directly interact with the system; one that does is called an " actor".

A use-case diagram shows actors (external objects) that interact with the system and the system functions that they 'interact with'. It is easy to ask various stakeholders what the system should do and document their responses on a use-case diagram.

Object interaction diagrams

Object interaction diagrams (OIDs) model the behavior of use cases by describing the way groups of objects interact to complete the task. There are two types of OID (Fontoura):

- Collaboration diagrams can be used to show how objects in a system interact over multiple use cases. Collaboration diagrams are helpful during the exploratory phases of a development process (i.e., trying to search for objects and their relationships). Since there is no explicit representation of time in Collaboration Diagrams, the messages are numbered to denote the sending order.
- A Sequence diagram is typically used to show object interactions in a single use case and it is easier to see the order in which activities occur. The emphasis of sequence diagrams is on the order of message invocation. The vertical axis of a sequence diagram represents time whereas the horizontal axis represents objects.

Activity Diagrams

Activity diagrams show behavior with control structure. Activity diagrams can be used to show behaviors over many use cases, model business workflow, or describe complicated methods.

- Activities in a diagram may or may not correspond to methods.
- Specific notation found in this type of diagram includes guards, which are logical expressions that evaluate to true or false.
- A synchronization bar indicates that the outbound trigger occurs only after all inbound triggers have occurred.
- Swimlanes (using a swimming pool analogy) allow you to vertically partition an activity diagram so that the activities in each lane represent the responsibilities of a particular class or department.

2.1.2 Analysis, Design and Implementation

Analysis: Requirements

UML support for Analysis to be discussed include:

- Use cases and actors are used in use-case diagrams to visualize, capture and describe functional requirements at the requirements analysis phase

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of the software development life cycle. Use case diagrams provide means to communicate with domain experts to refine the conceptual model of the system being designed. They provide "snapshots" of different aspects of the system required.

- Domain analysis using static domain class diagrams, as well as state transition diagrams, sequence diagrams and activity diagrams and parallel processes capture static and dynamic behaviors of the system and its constraints.

Design: Package, Class, and Relationships

UML support for design to be discussed include (Monarchi 1992):

- Use of package diagrams for architectural design and reuse.
- Detailed design using class diagrams, class stereotyping, class packaging and documenting.
- Class structure design using attributes, operations, and inheritance.
- Relationships: Class diagram association, aggregation, multiplicity, package diagram relationships.

Design (Dynamic): Sequence, Collaboration, State, and Activity.

Dynamic modeling of object interactions using:

- Sequence Diagrams: Show object interactions by time sequence.
- Collaboration Diagrams: Show object interactions by context.
- State Diagrams: Show object states and events that cause transitions between them.
- Activity Diagrams: Show flow of activities due to operations and object interactions.

Implementation: Components, Codes, and Deployment

UML notation for implementation to be illustrated will include:

- Component Diagrams: Show dependencies among various types of codes: source, binary, executable, interfaces, linking, execution, etc.
- Deployment Diagrams: Show components distribution among processor nodes, repositories, networks, and communications between them.
- Java and UML: Existing tools for domain modeling, Specification modeling and code generation will be looked into.

2.2 RUP (Rational Unified Process)

Software process is the aggregation of stage, technology, practice and related products used in the software development and maintenance. Effective software process can promote the efficient software development, improve the software quality and reduce the cost and risk.

RUP is one unified process developed by Rational Company with continuous conceptual and practice development. RUP emphasizes that software development is an iterative model and separates the process of software development into four phases

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(Inception, Elaboration, Construction, and Transition) and nine distinct core workflows. Each phase ends with one major milestone; evaluation will be work in the end of phase to make sure if the goal of the phase is finished. If yes, it will forward to the next phase (Pollice).

According to the traditional waterfall model, Software development can be divided into the following steps: Business Requirement Analysis, System analysis, system design, implementation, test, deployment, supporting, and change management. The traditional waterfall model assumes that the previous processes have been completed before the new one starts. This model has lots of problems although it seems a reasonable and high, efficient solution. The problem is that the method ignores the implicated software development process is affected by lots of reasons. So you will always face difficulties in each phase. It means if you adopt that model, it is very possible to waste the time, money and energy requiring redo it.

Therefore, RUP adopt interactive model to implement software design. Given the time it takes to develop large sophisticated software systems it not possible to define the problem and build the solution in a single step. Requirements will often change throughout a projects development, due to architectural constraints, the customer's needs or a greater understanding of the original problem.

Iteration allows greater understanding of a project through successive refinements and addresses a projects highest risk items at every stage of its lifecycle. Ideally each iteration ends up with an executable release – this helps reduce a projects risk profile, allows greater customer feedback and help developers stay focused. Figure 2.1 describes the steps to implement the process.

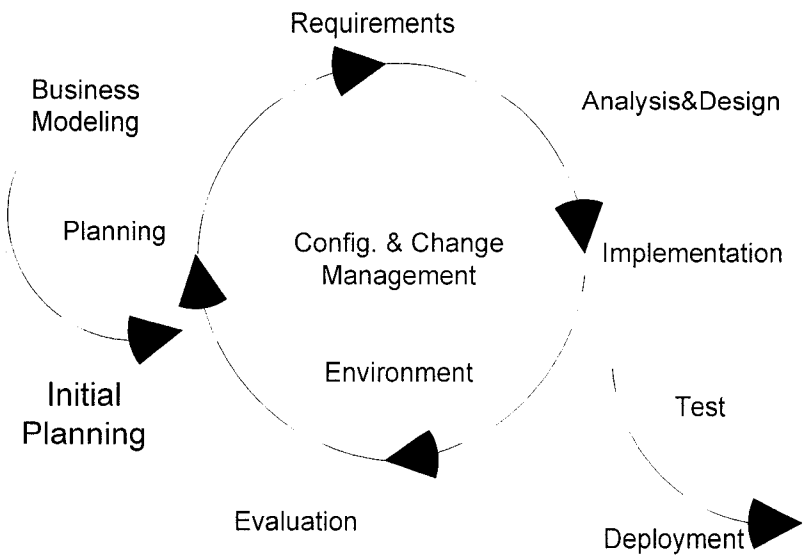


Figure 2.1 RUP Module

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In addition, RUP is driven by use cases. A Use case is an important concept in the RUP methodology and used to separate complicated and large systems into small units (use cases) in the system analysis and system design phases, and then develop each use case. In fact, Use Cases and Scenarios are examples of artifacts prescribed by the process and have been found to be very effective at both capturing functional requirements and providing coherent threads throughout the development and deployment of the system. In the process of business requirement analysis of RUP, customers describe use cases; in the system design, designers analyze the use case; in the development process, programmers implement the use case; in the test process, tester tests the use case.

Finally, RUP emphasis software development focuses on the architectural design. Architectural design is one important part of the system design. In the architectural design, an architect must design the whole framework and design the public module for example, auditing, log, exception handling, and security. Also, an architect must provide a solution to system extensibility, security maintainability, scalability, reusability and performance.
Component Based Architecture creates a system that is easily extensible, promotes software reuse and intuitively understandable. A component often relates to a class or sets of classes object in Object Oriented Programming.

RUP also defines 4 modules, Use Case Model, Analysis Model, Design Model and Implementation Model. Use Case model consists of Use Case Diagram and Use Case document. In fact, the Use case module is the basis of the other 3 modules; the Analysis model is result of the system analysis, also known as the conceptual model. The Analysis model includes class diagrams, sequence diagrams and activity diagrams. The Design module is the result of architecture design and system design. A Programmer can do the coding after implementing the design module. The Design module mainly consists of class diagrams, sequence diagrams and state chart diagrams(Pollice).

It seems that the analysis and design modules have some in common, but they do have some difference. The Analysis module doesn't focus on the solution to the problem but on the boundary of the problem, it doesn't involve in the technique and the platform. In the contrast, the design module should put forward the entire solution to problem. Certainly, the design module is based on the analysis module and each class in the analysis module can map into the design module directly, but these kinds of mapping are not one to one.

The last module is the implementation module. Implementation consists of component diagrams. Programmers can create skeleton source code from this module.

It implements the software lifecycle in iterations. Software designs are more and more complicated; in addition, people think software should process high quality and stability. These requirements are more pressure to software designers.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

They make the designers develop software quickly and at the same time to ensure the quality of the software. Therefore, it is unbelievable that there is not one definitive and repeated process in use in software development. It also is impossible to direct each phase of the development process. Therefore, some modeling languages like UML are very necessary to software development. Moreover, abstracting your programming from its code and representing it using graphical building blocks is an effective way to get an overall picture of a solution. It can also allow less technically competent individuals who may have a better understanding of the problem to have a greater input. UML simplifies the process of software design and provides a blueprint for the system design. Since the diagrams show both general and detailed information, they demonstrate that UML is capable of displaying various kinds of information. Flexible, easy to comprehend, and easy to build are traits that make UML diagrams a superior choice for business process modeling (Booch 1998).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.0 Some useful techniques and methods

In the section, we will introduce some new methods and techniques. Most of these methods will be used in the following sections.

3.1 QFD (Quality Deployment Function)

Quality Function Deployment (QFD) originated in Japan in 1966 and came to North America in the 1980s. Quality Function Deployment is a design tool of Total Quality Management (TQM) that was originally used to bring new products to market faster (Figure 3.1). QFD is not a quality tool itself, but rather a visual planning tool that helps to improve quality. When used to focus on the customer's needs early in the design, the team responsible for the development and the introduction of the product finds that fewer changes are required during the development and after introduction of the product. A bonus to this is that the product is of higher quality in terms of being the right product. When QFD is correctly utilized it creates a closed loop that lowers costs, and increases quality, timeliness, productivity, profitability, and market share. (Lamia PP 15213-3890)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

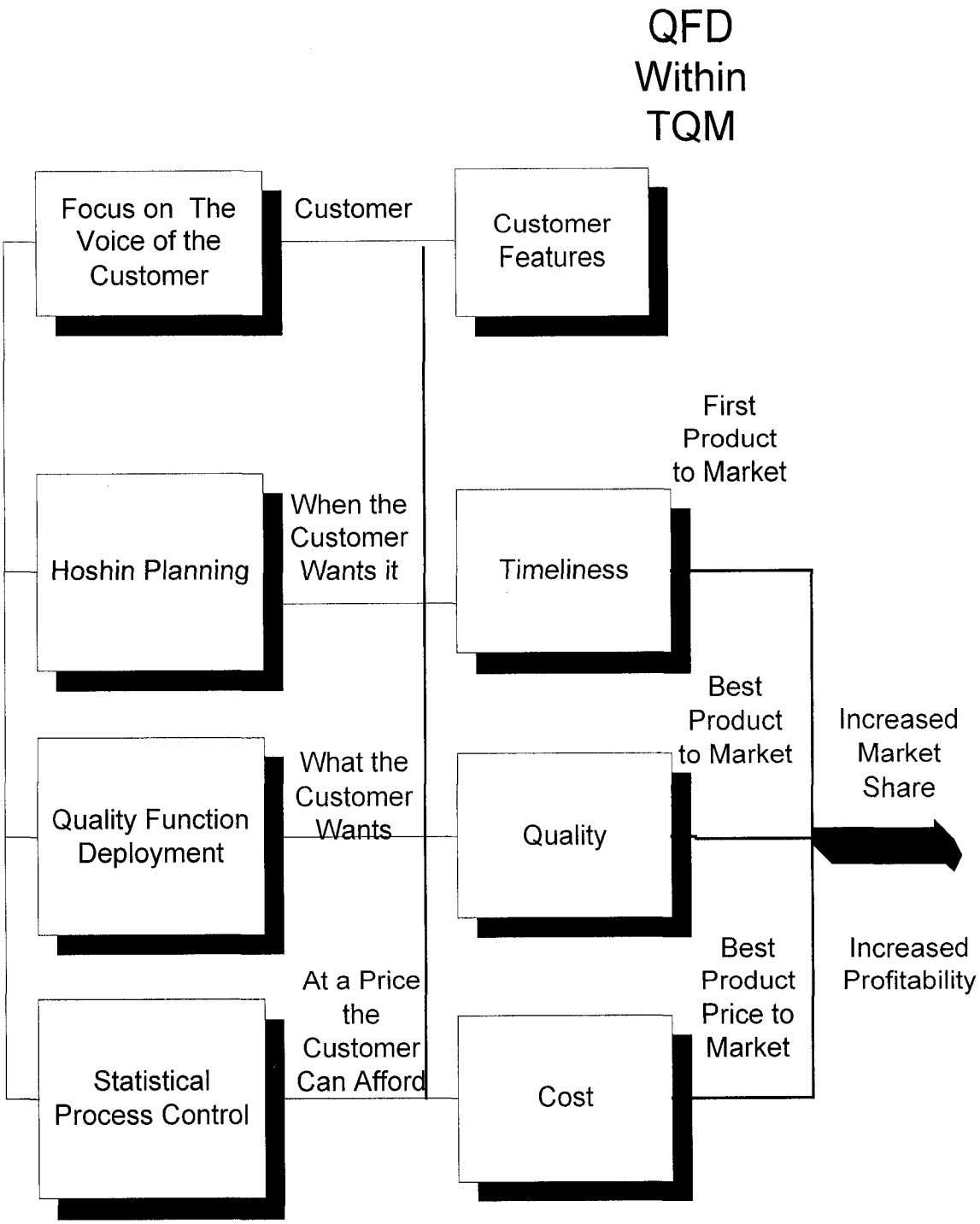


Figure 3.1 QFD within TQM

Quality Function Deployment (QFD) is used to understand the needs of the customer and then assistant in translating them into a set of design and manufacturing

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

requirements. The most common use of QFD is to define product requirements. By using QFD a business is motivated to focus on its customers, and to translate customer requirements into internal product specifications. With good initial requirements, the customer obtains a higher quality product in a shorter time. QFD plays an important role in Software Engineering because it gives a systematic and quantifiable approach to determining what is of value to the customers.

According to Akao (1990), the definition of QFD reflects two purposes, and together these two purposes create 'Quality function deployment in the broad sense'.

- Quality deployment: focus on the product, deployment of customer needs and requirements together with other important areas of the product, e.g. technology,
- Quality function deployment in the narrow sense: focus on the processes, deployment of quality activities in the functional organization.

QFD's primary goal is the overcoming of three major problems:

- Disregard for the voice of the customer
- Loss of information
- And different individuals and functions working to different requirements

QFD is used to make the transition from reactive to preventative manufacturing quality control. By clearly defining the objectives needed to achieve customer-defined quality, QFD helps to build quality into the product. This minimizes the impact of variability as the product is being developed. The costs are higher in the beginning of product development, but are greatly reduced after the product is released. Traditional approaches to development spend less at the beginning of development, but costs can be high after release as the product that has to be fixed or improved due to poor quality (Figure 3.2). When used properly QFD helps companies design more competitive products, in less time at lower cost and with higher quality.

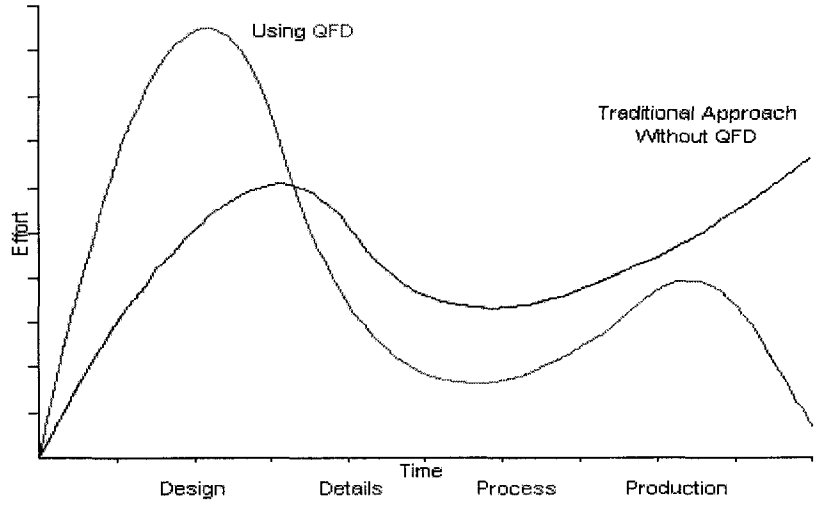


Figure 3.2: The Competitive Advantage

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

By using Quality Function Deployment a knowledge base is built that can be used in similar projects. We can see from the QFD documentation why decisions were made. The initial time spent on the QFD process may have been long, but work on a similar project can be shortened by reviewing past documentation. The QFD documentation also provides traceability from the product back to the customers needs.

The QFD documentation acts as glue for all the various product development stages. It ties design and process stages together and the documentation can be checked at any stage to see if the initial requirements are being met. This helps to control Murphy's Law by keeping things from going wrong as a product makes its way through a complex series of design and production activities. (Boushi)

For software development, Quality Function Deployment can greatly reduce the cost of a project by helping to insure that the correct initial requirements are used. QFD helps the organization to get the correct requirements before development starts and this reduces redesign, re-coding, and other related costs later in the project.

3.1.1 Voice of the Customer

Defining the voice of the customer is the most important and the most time-consuming step in Quality Function Deployment. Without a clear understanding of the voice of the customer, QFD can become a futile exercise (Eureka 1994).

The initial step in defining the House of Quality is to determine the Customer Requirements, their relative weightings, and to also determine Customer Evaluations of competitors. It allows a development team to understand what the Customers perceive as their most important needs and where they think competitors have strengths and weaknesses in regards to these needs. This is essentially defining the "what" of the system.

The voice of the customer is characterized by customer requirements defined by interviews, brainstorming, feedback mechanisms, and market research. Not all customer requirements are verbalized or easily voice of the customer can be diverse. It can be a consumer, a supplier, or even multiple customer voices within an organization. A key point of QFD is that the customers using their own terms to define quality. Traditionally software and systems have been defined in terms that mean little to the users or customers. With QFD the software and systems must be defined in terms that are meaningful to the customers.

Three essential types of information can be derived from looking at the User Voice in the House of Quality. The first of these is a list of Customer Requirements that is the first step in the process. The defined requirements can then be ranked according to customer's perception of their importance. Any scale can be used for this rating as long as it used consistently. Typically a rating scale of 0 to 5 (from No Importance to Very Important) is used in QFD processes. This allows the development team to concentrate on the requirements that the customers value the most, thereby increasing the value of the system to those who will be using it. As with any form of subjective

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

evaluation, it is important to verify that the results reflect what is actually needed by the customers. Large samples help to weed out discrepancies but there may be common views that would result in an inaccurate picture of what is really important. Customers may perceive that particular issues are important but on using a system or thinking further realize that their initial views were not correct. It will always be difficult to get a definitive answer as to what is important but QFD allows the evaluation of customer perceptions and gives structure to decision making(Boushi).

The final result from determining the User Voice is an evaluation of competitors' strengths and weaknesses for each of the defined requirements. Gathering the information for this can be difficult, and for internal projects possibly irrelevant, but if done it allows an evaluation of the proposed system against competitors strengths and weaknesses. If a requirement is rated as important and competitors are weak in that area there may be an opportunity to gain advantage by stressing solutions to that requirements. Conversely areas where competitors are strong for important requirements may require additional effort so as to not be perceived as having a weakness.

3.1.1.1 Analyzing the Customer Voice

The outputs listed above are the usual products of the initial phase of developing a House of Quality. A wide variety of additional information can be included or derived during this stage. The House of Quality is very flexible and can be extended in any way that the users of it see fit.

Affinity Analysis can be done on the requirements to see which requirements are related. This can help in determining structural and functional boundaries for a system. Target quality goals for the next release of the system can be included and compared to current ratings and competitors' ratings. This in turn helps determine improvement factors for particular requirements; how much work needs to be done to reach the quality goals? The overall importance of each requirement can then be determined by multiplying the importance to the customer with an improvement factor. This helps in determining which requirements should be concentrated on; an important requirement with a middling improvement factor may be of more importance than a low importance requirement with a high improvement rating. Sales Points, which are an indication of areas that the company feels are important (whether or not they are rated as important by the customers), can also be included to give another way of evaluating the importance of each requirement (Kulik 1998).

3.1.1.2 Evaluating the customer voice

The matrix approach used by the House of Quality can also be used for information other than the normal requirements and features. User types can be analyzed against requirements, business needs against requirements, business needs against features, user types against business needs, etc. Again, the House of Quality is flexible, limited only by the uses that can be developed.

As the saying goes, 'Garbage In, Garbage Out". The value of the information derived in evaluating the Customer Voice is dependent on the quality of the information that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is used to populate the House of Quality. If insufficient or poor data gathering is done the value of the information will be limited. This is especially true of the importance rating of the requirements and the competitor ratings. This type of information can be difficult to obtain and must be validated before being used. If development decisions are made based on incomplete or invalid information, the decisions made are unlikely to be valid.

The "semantic quality" of the requirements is also extremely important. This refers to all requirements being correct and relevant to the model. Completeness is also a factor in semantic quality and indicates that all-important statements about the domain have been included. (Completeness in abstract interpretation is an ideal and rare situation where, for a given abstract domain A, no additional loss of precision is introduced by approximating the meaning of programs by evaluating their semantics in A. Therefore, complete abstract domains can be rightfully considered as optimal). Evaluating each requirement and determining whether or not there are any problems with the specific requirement can determine correctness. Correctness can be achieved using normal analysis techniques. Completeness is much more difficult to ascertain in that most domains are fairly large and involve many requirements. There is typically no way to ensure that all of the requirements have been specified (although formal methods might allow a mathematical proof). However this is not unlike normal methods of determining the requirements for a system; any business needs that are not encompassed by specified requirements will need to be incorporated at a later point in the process. The more complete the original requirements the more complete and correct and the design and development will be (Roberto).

"Pragmatic Quality" is another factor in the validity of the House of Quality. For a requirement to have pragmatic quality it must be comprehensible to the audience for the work being done. While any requirement may be understandable to those involved in defining and evaluating; it must be understood to people not involved in the process. If a manager has a different understanding of what a requirement means than the analyst no common understanding has been developed and there will be areas of the system that do not meet expectations. Pragmatic quality is improved by improving semantic quality but differences in experience and in nomenclature must be taken into account when evaluating each requirement.

Analyzing the Customer Voice using the House of Quality provides a great deal of information and direction in developing the required system. These include an understanding of which customer requirements are the most important, strengths and weaknesses of both our own and competitors products, areas where we should focus our efforts, areas that require the most work to come up to the standards we have set, and what must be done to meet the customers expectations.

3.1.2 Determining the Product Features

After developing an understanding of the customer voice the next step is to understand the supplier voice. This is the "how" of the system. In this stage design measures or product features that can be used to address the defined customer requirements are developed. After appropriate product features are defined they are evaluated to determine how well each feature addresses each requirement.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Defining the appropriate features requires an evaluation of the customer requirements. The defined requirements are each looked at in turn and features that will address the requirements are defined. The evaluation will depend on the knowledge of the requirements gained in the first step of the process and on existing knowledge of the domain as well as an ongoing assessment as the process is carried out. In situations where an existing system is being upgraded or redeveloped there will be an existing list of features that can be used and modified as appropriate.

There are three essential types of information that are developed when looking at the Developer Voice in the House of Quality. The first of these is a list of Product Features that address the customer requirements. These are the start of determining what a system will include to satisfy the customer(s). Each of these features is then evaluated and rated according to how well they address each customer's requirement. Each product feature may address a single customer requirement or they may address several. The final types of information are Target Values for each Customer Requirement. These are determined by evaluating each customer requirement and the competitor ratings. A target value is set according to the level desired for each requirement. For example, if a requirement was rated as a 5 in importance but each competitor was rated as a 3 it might be appropriate to set a target value of 4; this would beat the competitors and also address the relative importance of the requirement without trying to meet the requirement perfectly (Kulik 1998).

3.1.2.1 Analyzing the Voice of the Designer

This phase of developing the House of Quality could be used to evaluate anything that could be related to the customer requirements. Examples of this would be to try and rate the relative cost of each requirement, to rate the importance of each requirement to each of the departments of a company that would be affected by requirements, or to rate the change that would be required for each department if the requirement were implemented.

As with the customer requirements, semantic and pragmatic qualities are an important consideration in evaluating the product features as defined in the House of Quality. The features must be defined in a way that is understandable to anyone who might need to use the House of Quality for further work. If there is any misunderstanding of what the features mean there are likely to be problems in later stages as the features are implemented.

An additional consideration at this point is the scale used for valuing the relationship between the requirements and the features. The standard QFD scale uses a 0 or Null to indicate no relationship, a 1 to indicate a weak relationship, a 3 to indicate a reasonable relationship, and a 9 to indicate a strong relationship. This is used to stress the importance of the features that strongly address a particular requirement. A feature that has a strong relationship with a high importance requirement, will be measured as having a value of 45 (Strong Relationship – 9 with a high importance – 5 requirement) while a feature that has only a reasonable relationship with the same requirement will have a value of 15 (3 * 5). What this means is that a feature that had a reasonable relationship with 3 requirements would be valued the same as a feature

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

which had a strong relationship with only one requirement. An alternative relationship scale of 1, 2, and 3 would have relative weightings of 15 (for a strong relationship) and 10 (for a reasonable relationship) and would be skewed less in favor of the strong relationships. Either scale (or any other selected) is valid but participants and anyone reading the House of Quality needs to be aware of the effects of the scale chosen. Non-numeric scales can also be used and provide a visual picture of how well features match with requirements. However each symbol represents a numeric value in the scale and must still be used as a number in performing further evaluations of the House of Quality (Boushi).

Analyzing the Designer Voice provides information that can be further evaluated to provide numeric valuations of the House of Quality. Valuing the relationships between the features and the requirements provides a numeric representation of the overall importance of each feature. If a feature has a number of strong relationships listed with it, it is likely to be a valuable feature. The target values give some indication of where work will be required to meet the end goals of the project.

3.1.3 The House of Quality

The final stage of developing a House of Quality is to evaluate the matrix developed and the valuations given to a variety of components in the matrix. Completing this stage provides a great deal of information that can be used in further developing the system and in setting targets for the end product.

The first product of evaluating the House of Quality is a numeric evaluation of the relative value of each feature defined in the House of Quality. The relationship value (typically 1, 3, or 9) for each defined relationship is multiplied by the importance factor for each requirement. This gives weighting to the value of the feature for meeting the overall importance of the requirements in the system. This weighting indicates which individual features will contribute most to meeting the overall importance goals of the customer. These values can also be represented as a Pareto chart to show visually which features are most important. An Overall Importance rating can also be derived at this stage to give an indication of which customer requirements will require the most attention in the design and development stages. This is simply the product of the importance rating of the requirement and the target rating for the requirement. Important requirements which need to have a high target level will be highlighted by this number and provide guidance as to which requirements are the top priorities during construction.

3.1.3.1 Analyze the House of Quality

Additional evaluation or work could be done to the House of Quality after development of the matrices. This is highly dependent on the type of matrix used and is limited only by the amount of information and creative approaches utilized. The initial House of Quality could be used as input to a variety of other Houses of Qualities if desired. One obvious approach would be to get to a more detailed statement for each feature and how it meets the requirements. It is done by making each feature a requirement and developing detailed features to address the higher level features. Another approach would be to do a detailed analysis of each of the features against the features in competitors' products.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Customer evaluations and importance rankings are not always objective. For this reason they must be validated as well. Customers may feel that something is important but if asked to reevaluate it -they would come up with different answers. The ratings of competitors' products must be checked as well. Good marketing can lead to people having views that are not always correct. All of the results are based on input and evaluation but must be validated .The House of Quality is a tool, not a definitive answer to defining requirements and features (Kulik 1998).

3.1.4 QFD Phases

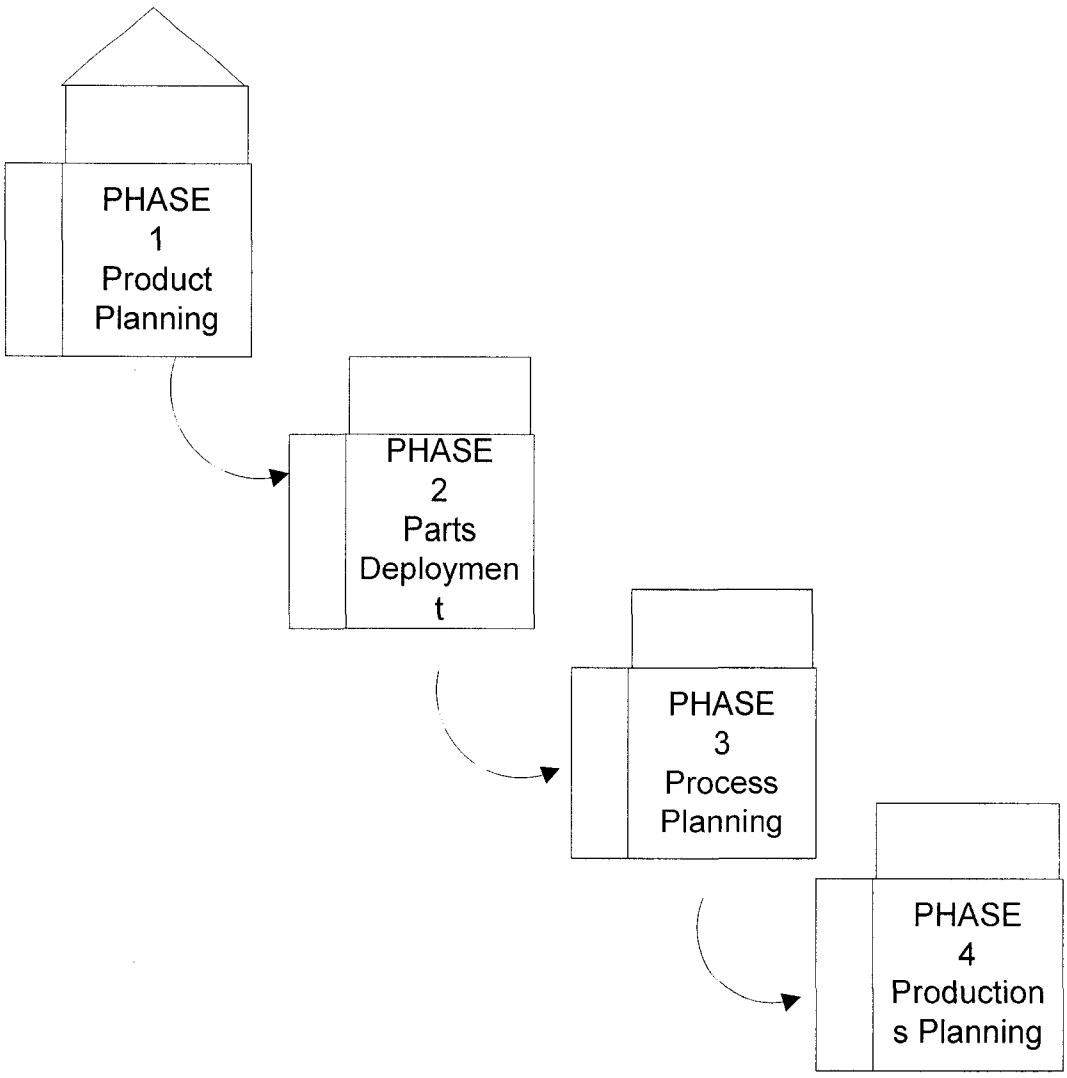


Figure 3.3 the Four Phases of QFD

Four phases are often used to deploy the voice of the customer from product development through manufacturing quality control. For each phase the How's of the proceeding phase are passed along as the what's (requirements) for the next phase. For example the How's carried over from the Product Planning Phase become the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

What's for the Parts Deployment Phase and the design specifications are then converted into the individual part details. QFD starts with customer requirements and then translates them into technical requirements, part characteristics, manufacturing operations, and production requirements as the process moves through each of the four phases.

Phase I: Product Planning

This is the most common phase used in Quality Function Deployment and it is very popular in the U.S. Most activities are centered on the House of Quality. Phase I is the most important phase in defining customer wants in relation to product parameters.

Phase II: Parts Deployment

Phase II is associated with product engineering functions. The design parameters are transferred into part characteristics and target values for fit, function and appearance. It is at this phase that part characteristics are identified that is critical to the execution of the measures from the previous phase. Currently about half the applications in the U.S. using QFD have progressed to this phase.

Phase III: Process Planning

Phase III involves floor level process engineers, production supervisors, and line operators, and represents the transition from design to manufacturing operations. The target values from the previous phase are deployed into process parameters for manufacturing and assembly. This is when process capability levels are developed and activities established for continuous improvement.

Phase IV: Production Planning

This is the final stage in the linkage between the voices of the customer in Phase I through subsequent phases. In this phase the target values from process planning are transferred into production standards. This phase takes advantage of the knowledge of those individuals that build the product on the factory floor. In phase IV all employees of the company and their activities interact to achieve customer expectations(Boushi).

Quality Function Deployment can be used in any of the phases but generally it is more effective when used in an early phase. QFD can enhance an organization's existing design process, but it does not replace that process. It can be integrated into a sequential, concurrent, or a unique design process. The approach is flexible enough that the design team can decide when to start and stop the QFD process. The best place to use QFD would be to focus on the high-risk details of product development, and those product aspects that the normal system of development cannot assure such as problem areas and innovations.

3.1.5 QFD Process

There are nine distinct steps that need to be completed to in the "House of Quality". Each step will fill in one crucial area in the "House of Quality". While some QFD

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

implementers may have more (or less) than nine steps, all of these steps must be completed to complete the House of Quality (Boushi).

The steps are:

1. Customer / Non-customer Requirements
2. Prioritization / Importance Rating
3. Technical Design Specifications
4. Relationship matrix
5. Competitor's Product Analysis / Rating
6. Target rating / Improvement Factor
7. Overall Customer Importance
8. Design Requirement Importance
9. Trade-offs and Synergies

Step 1: Customer / Non-customer requirements

In this step, you go out and get all the requirements from the customer. These requirements can be solicited through various means. Non-customer requirements are also necessary at this point too. These can be the requirements of management or marketing. Each requirement is *what* the customer wants. The requirements may be vague such as "Easy to use". Although vague requirements are okay, they should be clarified further to more specific requirements. Kusiak (1993) suggests that no more than 20-30 categories be specified. This is the most important step of the process because it identifies the "Voice of the Customer".

Step 2: Prioritization / Importance Rating

For each requirement listed, determine the customer's importance rating. The customer assigns a value between 1 and 5 for each requirement. The importance rating is typically done through a "forced choice" (ITI-OH 1995) where the customer must determine the relative value of one requirement against the others. This prioritization can be done through various techniques, one of which is the Analytic Hierarchy Process (ITI-OH 1995).

Step 3: Technical Design Specifications

The designer analyzes each requirement and come up with a measurable technical specification for each requirement. There may be more than one specification for a requirement; but not less. Some requirements can be covered through many specifications. It is important that the specifications meet the requirements in some way. Becker and Associates referred to this as the "Voice of the Engineer" (Becker 1998).

Step 4: Relationship Matrix

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

For each requirement and technical design specification relationship strength is evaluated. If the technical specification has a strong relationship to the requirements a value of nine is given. If the strength is moderate; If the relationship is weak but not nonexistent a one is given. The strength of the relationship indicates how well the technical specification fulfills the customer requirement. Any blank areas upon completion indicates a problem in fulfilling customer / non-customer requirements.

Step 5: Competitor's Product Analysis / Rating

This step involves rating the competitions' products. The customer does the evaluation and judging of the different aspects of the competitors' products compared against the customer requirements that were used in the first step. This evaluation will help in the setting of your own products' target values that is used in the next step.

Step 6: Target Rating / Improvement Factor

Comparing the customer's evaluation of your competitor's' products and the rating of importance given by the customer should assist in the selecting of target values of customer evaluated requirements. If the requirement is not rated very high and the competition scores low in this field, you know that it does not need too much attention. This may bring out some interesting evaluation scores. Perhaps the customer rates speed as very important and then assigns a competitor's product as very fast even though it is, in reality, slower than the others. Here, the customer perceives the product to be fast. This could be a warning sign to better determine the customer's requirements.

Step 7: Overall Customer Importance

This number is the (multiplication) product of the Target Rating / Improvement Factor (from Step 6) and the Prioritization / Importance Rating (from step 2). This calculation is done for the entire Customer / Non-customer requirements. The resultant values will provide an order as to the overall customer importance rating. For the less important requirements, the (multiplication) product will be lower than that of a more important requirement. Because the Target Rating / Improvement Factor is also part of the (multiplication) product, a greater distinction will be given to requirements that have a higher target rating or improvement factor.

Step 8: Design Requirement Importance

The importance of any one technical specification can now be determined by multiplying the customer's importance rating by the relationship strength value and summing up each of these products in their column. This summation gives the absolute importance rating that can be normalized or given as a percentage of the total.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Step 9: Trade-offs and Synergies

For each Technical Design Specification, there might be synergistic benefits realized by another Technical Design Specification or there might be trade offs. It is important to analyze each Technical Design Specification to determine if the implementation of one specification will hinder another specification. This is quite useful when one particular specification may be difficult to implement. In this case, identify synergistic specifications and concentrate on them. This is also useful when the trade-offs are identified. If a particularly important customer requirement is focused on and it has trade-offs against other important customer requirements, careful attention will have to be paid to avoid problems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.0 Why apply QFD to software

In the manufacturing field, QFD is used to focus on the quality aspects of projects. It could also be used in a software engineering environment. The success of any software organization stems from customer satisfaction, and customer satisfaction comes from receiving a quality software product. Thus, concentrating on quality pulls the organization ahead of the intense competition, and ultimately brings success.

Customers want value from their software. They want the product to help them solve problems and seize opportunities. It is also important to understand that customer actually have three types of requirements. These are *normal, expected,* and *exciting* requirements (Zultner 1993). Normal requirements are those that can be gathered by simply asking the customer. Expected requirements are requirements that are not mentioned but are expected. An on-line-help system is an example of this. Exciting requirements are requirements that are unexpected, but highly satisfying when they are delivered. These are the product features that really impress customers, or are made possible by new technology that the customers are not aware of.

To put quality into a software product, a software engineer has to understand what is meant by "software quality". There are two views of software quality -- the *traditional* view and the more *modern* view (Zultner 1993). The traditional view focuses on the minimization of defects. This is accomplished through existing software engineering approaches such as code inspections, reviews, walk thoughts, and testing. With this view, the software engineer understands the causes of defects, and strives to detect and correct them. The modern view of software quality aims at maximizing the value of the software. The software engineer understands the needs of the customer and designs value into the system. The difference between the two views is very important. With the traditional view, the best one can do is to have no defects in the system. However, even if a product has no defects, it is not necessarily of value to a customer. Therefore, the traditional view of software quality is insufficient. Furthermore, over 50% of software development errors occur in the requirements analysis phase (Eriksson 1998). These "defects" cannot be caught by the traditional view.

The software engineer needs to maximize the value in software products. This is accomplished by determining what is of value to the customers. These areas become the priorities of the project, and the team's best efforts are concentrated there. The task of determining what is of value to customers is not easy, and should be done with an approach that is systematic and quantifiable. This is where QFD plays an important role. QFD can be used to accomplish several things. It can be used to evaluate the impact of product features on customer value, and be used for considering trade-offs of product features in the design. It can also be used to set a development strategy or direction. For instance, one can use QFD to determine whether a software package should aim for technical excellence, or have improved ease of use. Finally, the House of Quality in QFD can be used to analyze competitive products as well.

4.1 Software QFD

Software QFD is an adaptation of QFD from its manufacturing roots. It also originated from Japan. SQFD is a front-end requirement solicitation technique that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

can be attached to any software development lifecycle process (Haag 1996). For example, a project team can use SQFD to gather requirements, and then proceed to develop the system using either the waterfall or incremental lifecycle process. SQFD has been used successfully by many large organizations, such as Digital, AT&T, Hewlett Packard, IBM, and Texas Instruments.

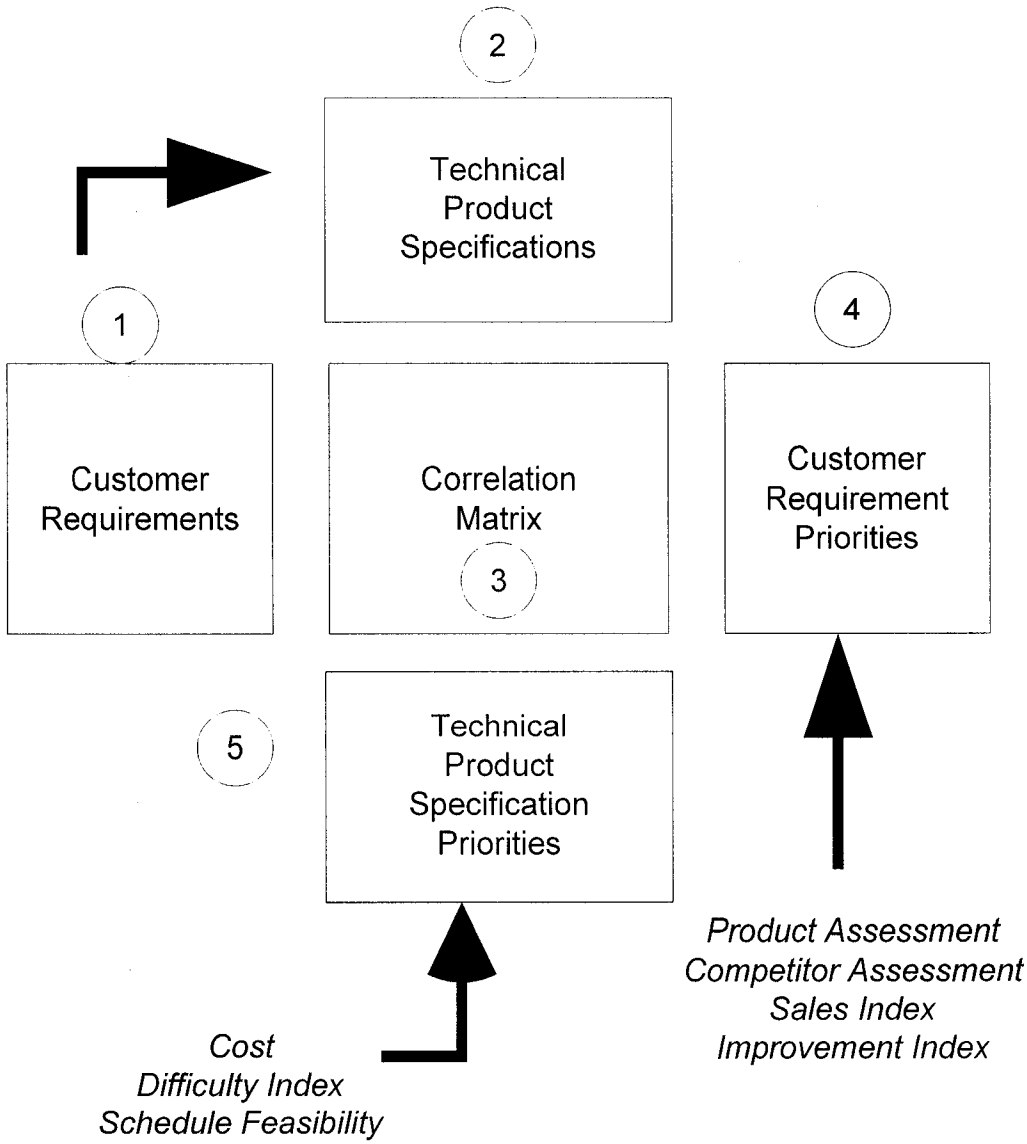


Figure 4.1 House of Quality

Proponents of SQFD say that this technique results in numerous benefits. SQFD results in fewer design changes, and less error are passed from one development phase to the next. Less maintenance is required because of this. At the same time, better communication occurs between departments since QFD teams are cross-functional, involving customers, engineers, sales, management, and so on.

4.2 Benefits of QFD

The QFD process provides a great deal of help in obtaining objective, measurable

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

information that can be used for understanding the product to be developed and how it will meet the customers needs for the product. There is guidance in how to carry out the initial information gathering process and what types of information to gather. QFD allows for a systematic evaluation of customer requirements for a product and features that will meet these requirements. There is a quantification of most of the information processed and this allows for an objective justification for decisions made as a result of that process.

The information gathered can help in resolving design tradeoffs and in setting quality goals and measures for development. If two critical features conflict, the conflict will need to be resolved in the design; this information is not known ahead of time and discovered much further along the process when resolving it would be much more difficult (Boushi).

QFD provides a way of tracing requirements from initial definition to completion. Because all steps are recorded and measured it is possible to revisit all decisions and filter changes to the appropriate parts of the project. This is often lost in conventional development processes as decision points are not clearly recorded and the reasoning behind the decisions are easily lost. QFD forces a focus on the customer needs. Any project, whether commercial or in-house, needs to meet the customer needs to be successful. By identifying and quantifying the customer requirements up front QFD ensures that the real requirements are not ignored and tractability helps ensure that they are still visible at the tail end of the project.

QFD can be an aid in shortening development time as it focuses on the essential needs for the product and the essential features to meet those needs. Once the initial costs for QFD are past the process can reduce costs. Trained personnel with appropriate tools can work quickly towards a good solution to a problem. The better, the initial solution the lower, the overall costs will be. QFD leads to a final system that meets the customer needs well and contains features that meet these needs. Features that do not contribute can be identified and excluded early on. In a commercial situation, the better, the solution meets the customer needs the more successful it will be in the market place.

4.3 Applying QFD in an UML framework

Basically, the requirement model is the beginning point of the objective methodology that forms the base of UML processes. After that, the remaining models of design and implementation and the final testing will be created based on the existing requirement model. So in the requirement model, actor and use cases become crucial because actors represent the functional roles that users can play, and use cases are comprehensive sequences of actions the actors perform with the system to accomplish and complete the task.

As described above, the concept of actor and use case is similar to the types of customer and the notion of "function" in QFD terminology. Therefore, the quantifiable differences between them would be an excellent extension base to OO analysis. That would provide valuable guidance to designers and project managers on where to allocate their most critical resources, and how to make implementation

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

choices that optimize the satisfaction derived by users of the finished system.

Based on these ideas, we can find that QFD matrices are an effective aid for OO analysis and might provide some useful information. On the other hand, all the matrices are easy to use and more descriptive.

In the following section, we will discuss some matrices derived from QFD and further explain the notation "A×B", where A represents the data type in the left column of the matrix, and B shows the data type in the top row of the matrix(Lamia).

Users X Actor Role

			Actor role	
Individual Users	Patron	Librarian	Volunteer	...
Jan	●	●		
Pat		●	●	
Prioritization of role	5	3	1	...

Table 4.1 Users X Actor Role

This Table 4.1 describes the different roles that play in different time. The bottom row: prioritization of role is optional that shows the relative importance of the role. Normally, they are decided by the judgment of the analysis team.
This information is easy to be captured by interviewing with customer in the initial stage of software design.

Actor Role X Use Case

			Use Case		
Actor roles	Search for book	Find book on shelf	Check out book	...	Role wt
Librarian	●I	●I	●S		3
Patron	●I		●I		5
Function Wt.	54	18	72	...	

Table 4.2 Actor Role X Use Case

We show some operations that the actor role joins in this table. We also indicate the importance of each use case by using the same rating method as matrix: Users X

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Actor role i.e. ● =9 and • =3.

In this matrix, the notation "I" means the initiator of the use case and "S" means the actor role will service in that use case. In this example, the role of the librarian in service to the patron during the use case: check out book.

User Demanded Quality X Use Case

		Use Case	
Demanded Quality	Search for book	Check out book	...
Must be fast	●		
Must be mistake proof		●	

Table 4.3 User Demanded Quality X Use Case

This Table 4.3 describes the demanded quality for each use case. Actually, it is an important matrix because this information of demanded quality is very helpful for designers to implement the software design.

Use Cases X Objects

		Object			
Use Case	Patron	Book	Terminal	Librarian	...
Search for book	●	●	●		
Check out book	•		●	●	

Table 4.4 Use Cases X Objects

This Table 4.4 shows each object that participates in the use case of the library system. We recommend using the specific name of an object in place of the vague name in that matrix because exact definition will enhance the quality of the software design.

We also can create some new notations to represent whether the object is created, removed, modified, or provided information to the use case such as "C"-created, "R"-removed, "M"-modified, "S"-provide.

Use Case X Data Attributes

		Data Attribute				
User Case	Title	Author	Subject	Call number	Library card number	...

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Search for book	●	●	●	●		
Check out book				●	●	

Table 4.5 Use Case X Data Attributes

This Table 4.5 shows all the data that will be used in carrying out various use cases. We don't describe the information of the objects in the matrix because we want to simplify the analysis process. By combining the "Use cases X Data attributes" with "Use case X Objects", we have captured much more information to be used in the software design.

Objects X Data Attributes

		Data Attribute			
Object-class	Name	Address	Library card number	Employee ID	...
Patron-class	●	●	●		
Librarian-class	●	●		●	

Table 4.6 Objects X Data Attributes

This Table 4.6 shows all the data that will be used in the objects. So the matrix will be very useful to construct an abstract superclass type because some objects might share common data attribute.

Objects X Objects, Showing Entity Relationships

		Object			
Object	Patron	Book	Shelf	Floor	...
Patron		0.n			
Book	0..1		1	1	
Shelf		0.n		1	
Floor		0.n	1.n		

Table 4.7 Objects X Objects

This Table 4.7 describes the association relationship between pairs of entities in the library system that can be used in defining the associations among objects.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In addition, the matrix also can add some processing rules of the library system, such as no one can borrow one popular book for more than one day.

Object x Classes

		Class	
Object	Person	Holding	...
Patron	●		
Librarian	●		
Book		●	
Periodical		●	
...			

Table 4.8 Object x Classes

This Table 4.8 shows relationship between leaf-node object classes and superclasses. One advantage of this matrix is that it can clearly identify all superclasses that contribute to object instances and helps find potentially conflicting attribute or method definitions.

Use Cases X IEEE Quality Factors

			IEEE Quality Factors			
Use Case	Efficiency	Integrity	Reliability	Survivability	Usability	Correctness
Search for book	●	●	●			●
Check out book			●			●

			IEEE Quality Factors				
Use Case	Maintainability	Verifiability	Expandability	Flexibility	Interoperability	Portability	Reusability
Search for book				●			

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Check out book						●	●

Table 4.9 Use Cases X IEEE Quality Factors

This Table 4.9 shows some quality factors that could be considered when carrying out a use case. That matrix will be helpful for designers and engineers to check whether a good quality product has been developed or not.
On the other hand, some quality factors in that matrix could be adopted in the cost benefit method (in the later chapter) to analyze the function of the system.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.0 New structure method

In this section, we will discuss a new structure method that might provide some useful information to aid in the OOA.

The method we will introduce can prioritize customer and user requirements, analyze tradeoffs in a way that both increases return on R&D investment and increases customer's user satisfactions. The core of the method is the "need-opportunity" matrix. It is organized into four quadrants to aid in analysis and prioritization of features-" nice to have", " add value", "must do", and "defer" (Kulik 1998).

1. Features and characteristics in the "Add Value "quadrant define the direction of the project, offer the greatest potential return on investment and will be the focus of implementation planning and effort.
2. Features and functions in the" Must Do "quadrant should be included as part of the scope of the product or system to be implemented and can be good candidates for cost reduction. In other words, it should be a minimum necessary to meet customer requirements in some cases; lower-priority "Must Do" features can help an organization understand evolving customer needs. In additional, the "Must Do" Features and characteristics may enter this quadrant from the "Add Value" or "Defer" quadrants, or "Defer" quadrant.
3. A limited number of features and characteristics in the "Nice to Have" quadrant can be selected based on available resources and forecast trends in customer or user needs. It can evolve to "Add Value" and further offer a competitive advantage.
4. Features and characteristics in the "Defer" quadrant should be eliminated from the plan wherever possible. These features may move to the "Must Do" or "Nice to Have" quadrants.

We will explain how to use this method with a case study. In the figure, each character represents a customer's need and the circle closes to the character indicates the relative importance of the needs.

A. Each elevator has a set of m buttons, one for each floor.

B. This button illuminates when presses and causes the elevator to visit the corresponding floor.

C. The illumination is canceled when the elevator visits the corresponding floor.

D. Each elevator has a button for emergency

E. Each elevator has a telephone.

F. Each floor, except the first floor and top floor has two buttons, one to request an up-elevator and one to request a down-elevator. These buttons illuminate when pressed. The illumination is canceled when an elevator visits the floor and then moves in the desired direction.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

G. When an elevator has no requests, it remains at its current floor with its doors closed.

H. Each elevator has a mirror

I. Each elevator has a picture frame.

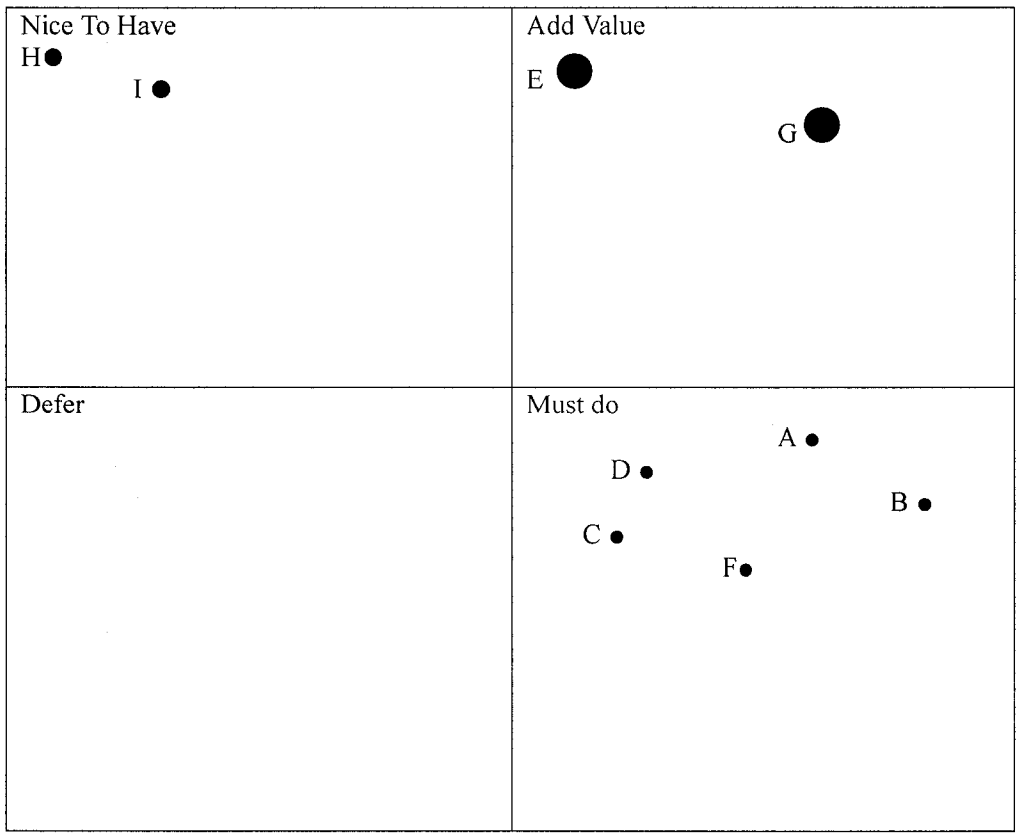


Figure 5.1 Need-opportunity Matrix

This method mainly uses the need-opportunity matrix, the core of the method, to prioritize the customer's requirements and analyze the tradeoff on investment. In the matrix, we also can add arrows to the circles to represent expected evolution of requirements. On the other hands, the need opportunity matrix can provide another useful function in analyzing the return on R&D Investment. The potential return on R&D investment is represented in the Needs-Opportunity matrix through the size of the circle for each feature and function. As shown in the below figures, larger circles signify greater potential return on investment, and smaller circles represent less potential.

Based on the two functions provide a by the need-opportunity matrix, program or project managers can use this method to promote OOA. In this matrix, larger circles signify greater potential return on investment, and smaller circles represent less potential.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.0 The limitation of UML and RUP

6.1 The limitation of UML

UML models are different from many people's understanding because UML adopts lots of distinct conceptions, glossaries and diagrams in order to describe the complicated external world in detail. Furthermore, UML lacks one refined core and good boundary and its definitions are not exact and sometimes misleading. Also, its language structure and semantics still have some weaknesses. Therefore, both customers and designers can't understand it completely.

6.1.1UML can't really communicate with customers

It is difficult for the customers to use UML to communicate with the designers because its style to describe the system is far beyond the customers' understanding. Customers are always confused when they read the paper filled with lots of glossary and symbol of software.

(1) UML lacks techniques for requirement modeling.

UML takes off the function models of structure method so that it may be identical to object-oriented methodology. So UML can't be used to find out customer's needs in detail and further affect the communication between customers and developers.

(2) UML lack techniques for domain modeling.

One goal of domain modeling is to make customers realize the system's business modeling; however, UML models are like a sealed book for customers. Therefore, customers can't give some suggestions to the business model.

(3) UML is short in describing the system performance.

UML only describes " how to do", but it doesn't talk about " how is it". The former means logic flow and the latter is performance indicator. In fact, the performance indicator is nonfunctional properties (NFP), such as safety, availability, reliability and temporal correctness. It is very important in reality when people are concerned about the goal and direction of the software development.

6.1.2 UML can't effectively direct designers to program

It is really difficult to get an UML-like design into a state that it can be handed over to programmers because UML doesn't support elaborate analysis design. The truth is that the UML-like design looks very good on paper yet be seriously flawed when you use it to implement software. So programmers spend considerable time translating the model into code.

6.1.3 UML can't describe the software system completely

Use case diagram:

Use case diagrams merely model the high-level functionality as one or more actors perceive it, it doesn't give more details about that. So how to integrate use case

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

diagram with other UML diagrams becomes an important problem.

Sequence diagram and Activity diagram:

Both sequence diagrams and activity diagrams are not able to model past states of objects and evolutionary patterns. Also, they are not equivalent with each other in formalness and content. On the other hand, there is no constraint imposing the nature of the role nor the consistency amongst the roles defined (e.g. all objects, all organizational units, etc.) So this deficiency restricts the application of the concept of responsibility to activity diagrams.
Sequence diagrams lack a representation for conditional activity.

State diagram:

Events don't correlate with external actors, class and package in state diagrams. Moreover, it is hard to correlate state diagrams with sequence diagrams.

6.2 The limitation of RUP

RUP still has some problems even though it is a good process method.
(1). RUP is only a development process; it doesn't cover all the content of software development. For example, it doesn't have methods to support software execution.
(2). RUP doesn't have development structure for multiple subjects. Therefore it reduces the impossibility of reusability implemented in software development.
(3). We can use other software processes for example Open and OOSP to aid in RUP.
(4). Use Case Driven Analysis, employed by RUP to implement process design, still has some disadvantages. The main disadvantage of UCDA is the lack of synthesis. The Use Case Model that we get from UCDA is just a loose collection of uses cases.

6.3 Solution

As mentioned above, UML and RUP still have many limitations. Some of these limitations might affect the quality of software design; even cause the failure of the project. Therefore, in this section, we try to employ some new techniques or methods to solve these kinds of problems.

6.3.1 Problem 1: UML can't be used completely to communicate with customers because it lacks the techniques to model requirement, domain and software.

Solution 1: QFD is a very effective tool in the initial stage of the software development because it really understands the needs of the customer and then translates them into internal product specifications. In additional, QFD plays an important role in Software Engineering because it gives a systematic and quantifiable approach to determine what is valuable to the customers. Finally, these techniques can strengthen the communication with customers and promote software quality. Therefore, we can adopt the information provided by these matrices to improve the UML notation and promote software quality.

6.3.2 Problem 2: Use Case Driven Analysis, employed by RUP to implement process

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

design, still has some disadvantages. The main disadvantage of UCDA is the lack of synthesis.

Solution 2: The Use Case Model that we get from UCDA is just a loose collection of uses cases. What we really would like to get from requirements analysis is a model that captures the functional requirements and system usage, without any design aspects. So we try to use Usage Oriented Requirements Engineering (UORE), extension to UCDA, to enhance the Rational Unified Process (RUP).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

7 .0 New Process and language

The Unified Modeling Language (UML) is a powerful notation for building software blueprints. The diagrams capture business process information. Since the diagrams show both general and detailed information, they demonstrate that UML is capable of displaying various kinds of information. Flexible, easy to comprehend, and easy to build are traits that make UML diagrams an excellent choice for business process modeling.

A methodology for applying UML modeling techniques within the OO standards development process is needed. Since UML is 'only' a modeling language, our system has selected the Rational Unified Process as candidate process to start with. The Rational Unified Process (RUP) is a software design methodology created by the Rational Software Corporation.It describes how to effectively deploy software using commercially proven techniques.

In this section, In order to apply the new useful methods mentioned in the previous section, we follow the Rational Unified Process to experience the whole process of building software. Business modeling and requirements (two flows) are thought as more important at the first stage (inception) of the RUP and will be discuss in turn to explain how to implement a new process. Therefore, we will start our new process description with them.

7.1New process

7.1.1The Business Modeling Workflow

The Unified Modeling Language (UML) is a powerful notation for building and expressing software model. Moreover, UML diagrams are an excellent choice for business process modeling because they are capable of displaying various kinds of information and can be applied to various phase of the business system lifecycle, from the requirement to implementation. Customers also can capture information by UML diagram from both static and dynamic view. Each UML diagram represents different functions and describes different metadata of the business system.

.

A business process model is a set of components that shows a set of activities. The purpose of creating a business process model is to better understand, analyze and improve a business process. However, we still need to find others tools or methods to aid in the UML diagrams to model business systems. We will try to use QFD-style matrices derive from QFD to support the description of business modeling.

The purposes of the business modeling are the modeling of the business context and the scope of system. Common modeling activities include the development of (Craig 1999):

- A context model (often a data flow diagram) showing how system fits into its overall environment
- A high-level business requirements model (often an essential use case model)
- A domain model (often a class diagram) depicting major business classes or entities
- A business process model (often activity diagram) depicting a high-level

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

overview of the business process to be supported by your system. This diagram is one level of detail greater than context diagram

7.1.1.1Context model

A context diagram is a top-level data flow diagram. It only contains one process node that generalizes the function of the entire system and its relationship to external entities.

Here, we take a banking system as an example to explain what is the context diagram in the business modeling. In the next chapter, we will use our own system (elevator system) to examine a case study in order to represent how to apply the new process and language.

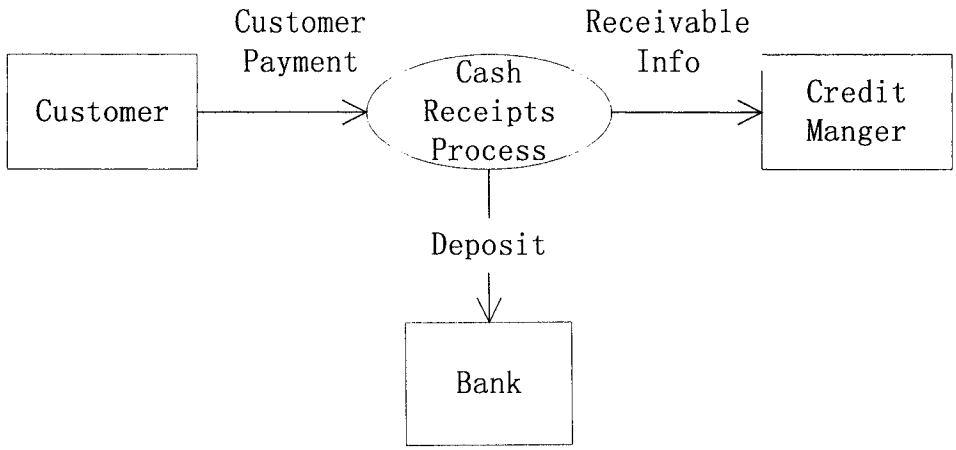


Figure 7.1 Context Diagram

Context document

The figure 7.1 is a context diagram of cash receipts. This context diagram defines the scope of a business model and identifies relationships among customer, bank and credit manager. This representation takes the form of a domain picture aiming to give an overall understanding of the domain. It focuses on describing stakeholders and their relationships and identifies stakeholders concerns. It typically covers key value-chains and information flows.

7.1.1.2 Use case model

The purpose of business modeling is to understand the structure and dynamics of the operations within a domain. It helps to ensure that all users, standards developers and software providers have a common understanding of the domain. In addition business

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

modeling is used to derive the high level requirements needed to support the subsequent detailed analysis and eventual solution. The business modeling workflow starts with a high level definition of the vision and scope of the domain to be considered (see Fig. 7.2). Furthermore, important terms used in the business should be covered in a glossary (e.g. the BuyerID: Seller assigned identification by which the seller uniquely recognizes a buyer). The vision and scope statement should allow derivation of the business actors (roles of the organizations involved in the considered business transactions) and the use cases (main business transactions under consideration). Since the scope of the system is the inter-organizational communication between involved organizations, the use cases focus on communicating processes between the actors and not on the internal operations performed by each actor (see Fig. 7.3) (David 1998).

Having found all use cases, the next step is to detail each use case. This covers a description of main activities performed in a use case and a high level description of information being exchanged. For example: To request a registration the buyer sends a registration request including his name and address, contact information and credit card information. This information could be used to design a first object model for each use case.

The vision and scope of 'Order from Catalog' is described by five business transactions depicting the process of a Buyer executing a catalog order with a Seller. "Request Catalog" is an optional business transaction. A Seller may offer to provide to any potential Buyer an electronic version of the current Seller's catalog on request.
"Register" depicts a first time Buyer initiating a relationship with a Seller by providing required buyer information, confirmed by receiving a Seller's Buyer ID from the Seller. "Request Price" (provide a price quote to the Buyer for selected product(s) on request) is an optional business transaction where the Seller may offer a price quote to a Buyer after a valid Seller's Buyer ID has been assigned. "Order Product" depicts the process of a Buyer ordering items from a catalog, having previously established a relationship with the Seller by providing Buyer information and receiving a Seller's Buyer ID (refer to "Register"). "Request Order Status" is an optional business transaction where the Seller provides order status information to the Buyer on request.

Figure 7.2 Visions and Scope Statement for 'Order from Catalog'

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

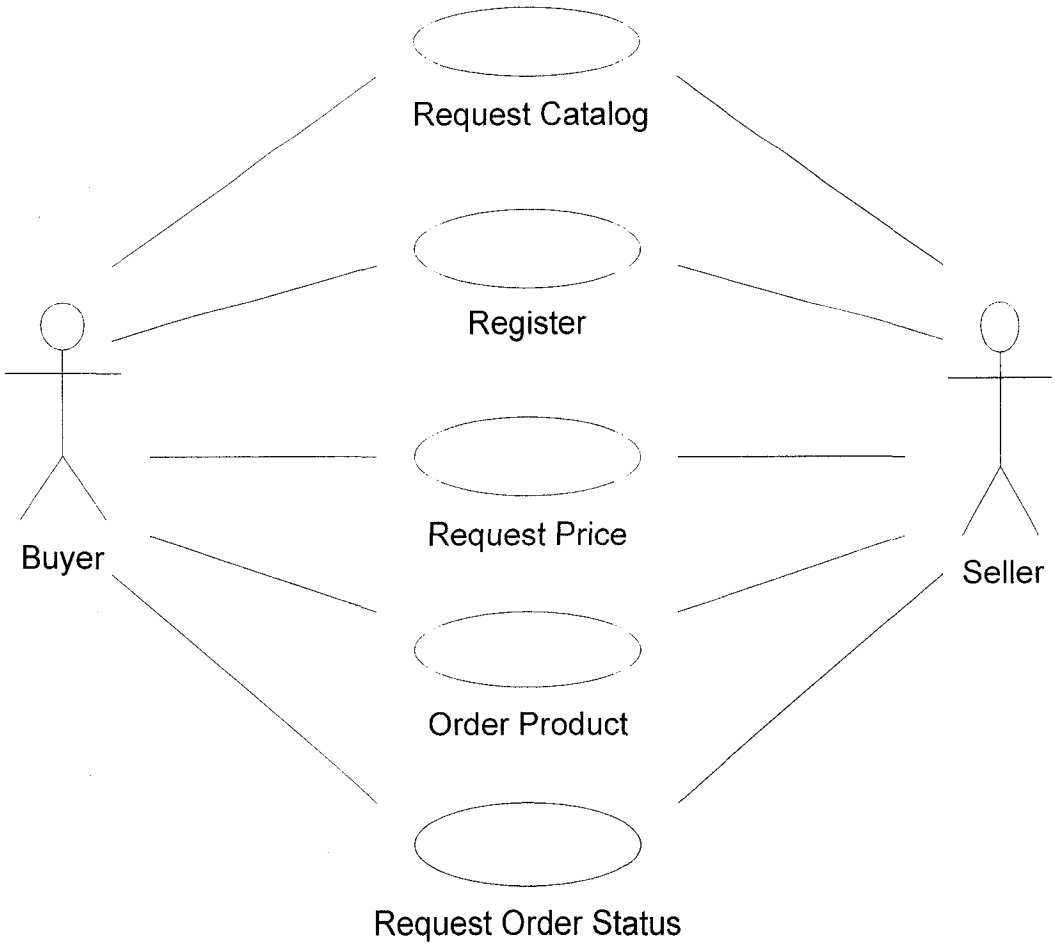


Figure 7.3 Use Case Diagram

7.1.2 Requirement workflow

In the beginning of requirement analysis in the system process, we will use one new method (cost benefit analysis) to deal with the customers' needs so that the designer can concentrate on the more important parts in the subsequent phase.

7.1.2.1 Cost benefit analysis

The goal of this method is to classify and prioritize the user' needs into different categories based on some principles in order to provide some reference for the process of system design. Moreover, we can use UML extension mechanism to represent those kinds of classification information into any model in order to make designers understand which one is more important in the following phase of the system. This method consists of 5 activities.

(1) Gather domain knowledge.

It is well known that customer requirements are essential parts in the system design. Here, we omit the process to gather and organize customers' needs and won't discuss related methods for example affinity diagramming, contextual inquiry ... we will assume that we have already collected all the customers' needs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(2) Evaluate the product.
In this step, after gathering the domain knowledge of a system, we need to find some experienced experts from different areas to evaluate the product from different respective in order to enhance the product.

(3) Categorize the issues
We need to use some techniques to group the problems that we captured from customers before we prioritize them. We use an affinity diagram to implement the functions. The advantage of the method is to expand our focus and give us a high level view of the problems area.

(4) Prioritize the categories
According to how important it is to fix them (from the users' perspective) and how difficult it is to fix them (from the developers' perspective).
We prioritize the problems into 4 domain, they are "High-value"(contains very important issues that require less effort to fix), "Strategic"(contains very important issues that require more effort to fix), "Targeted"(contains less important issues that require less effort to fix), and "Luxuries"(contains less important issues that require more effort to fix).
The following figure is an example to prioritize the category. This example is an evaluation that examines an affinity diagram(Kulik 1998).

We list some features below:
1. Facilitate Users' Tasks
2. Support Users' Mental Model
3. Convey Strong Sense of Place
4. Lay Out Information Logically
5. Provide Clear Cues and Instructions
6. Correct Errors
7. Provide Feedback
8. Provide Consistent Controls
9. Ensure Visual Design/Branding Are Appropriate
10. Provide Clear Languages

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

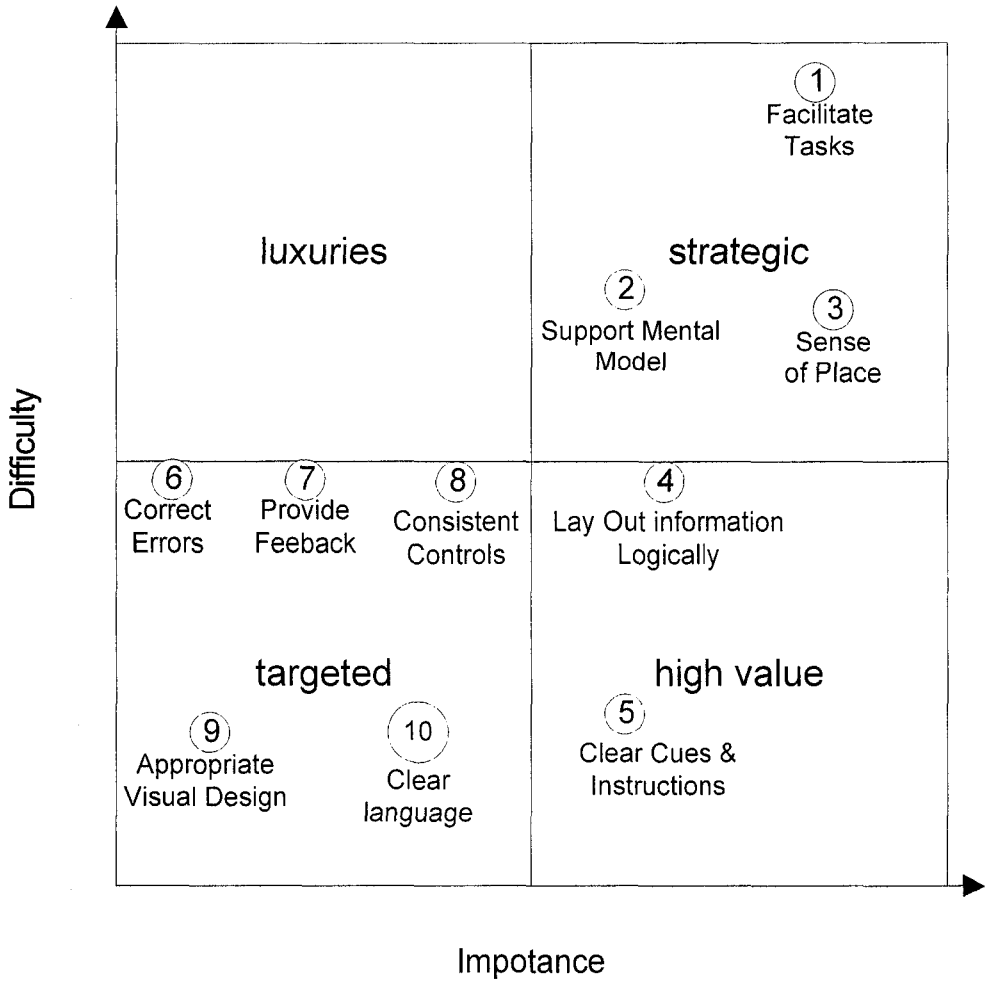


Figure 7.4 Cost Benefit Chart

Actually, the categories in the strategic and targeted quadrants are more important. The problems in the strategic quadrant may require structured rethinking, or significant redesign of a product. For example, the problems in the "Facilitate Tasks" category may require more research on what users' tasks are and how those tasks could be streamlined. In contrast to strategic issues, targeted issues may have solutions that are easier to envision and implement. For example, if the problems in the "Clear Language" category relate to the use of jargon and unfamiliar terminology, the unfamiliar terminology should be replaced with more common words or phrases. On the other hand, we recommend that clients address the categories in the luxury category last, since they represent the lowest ROI (Return on Investment).

(5) Write the report, including recommendations for solving the problems.
The last activity is to generate recommendations for designers in the latter phase of system after we've categorized and prioritized the problems. It will help usability professionals communicate more effectively with decision-makers about usability problems and solutions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

7.1.2.2. Use case model

The purpose of this discipline is to engineer the requirements for the project, including the identification, modeling, and documentation of those requirements. The main deliverable of this discipline is the Requirements Model, which encompasses the captured requirements. This stage normally takes a use case representing part of the business domain modeled in the business modeling workflow and refines the output for the area selected for the requirements modeling project.

Capturing a common vocabulary in a glossary is of great importance in the requirement workflow. Consider for example the term 'delivery date'. It seems that everyone might know what a delivery date is. But there is still chance for misinterpretations: Is it an exact, earliest, latest delivery date? Thus, a semantically complete definition must be stated in the glossary.

The next step of the requirement workflow is to *find the actors and use cases* (see Fig. 7.4), according to the boundary definition in the vision and scope statement. Users might be involved in the operation of the internal system, which is not considered in the system in question. But input and output to the use cases is always sent/received by the information systems themselves.

Consequently, the inter-organizational system has always to interface directly to the organizations' internal systems. To denote this fact, the use case model of the requirements does not depict actors, but the interfaces to the organizations' internal systems supporting the transactions. Taking a closer look on Fig. 7.5 it is easy to recognize that the definition of the use case 'Register Buyer' has been refined, because the use case takes advantage of another use case namely 'Verify Credit'. This is due to the fact that a seller wants to verify whether a buyer is credit-worthy or not. For this purpose, the seller contacts his bank to do this verification. Since this verification does not belong to the core processes of an order from catalog it is outside the defined system boundary (Madsen 2000).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

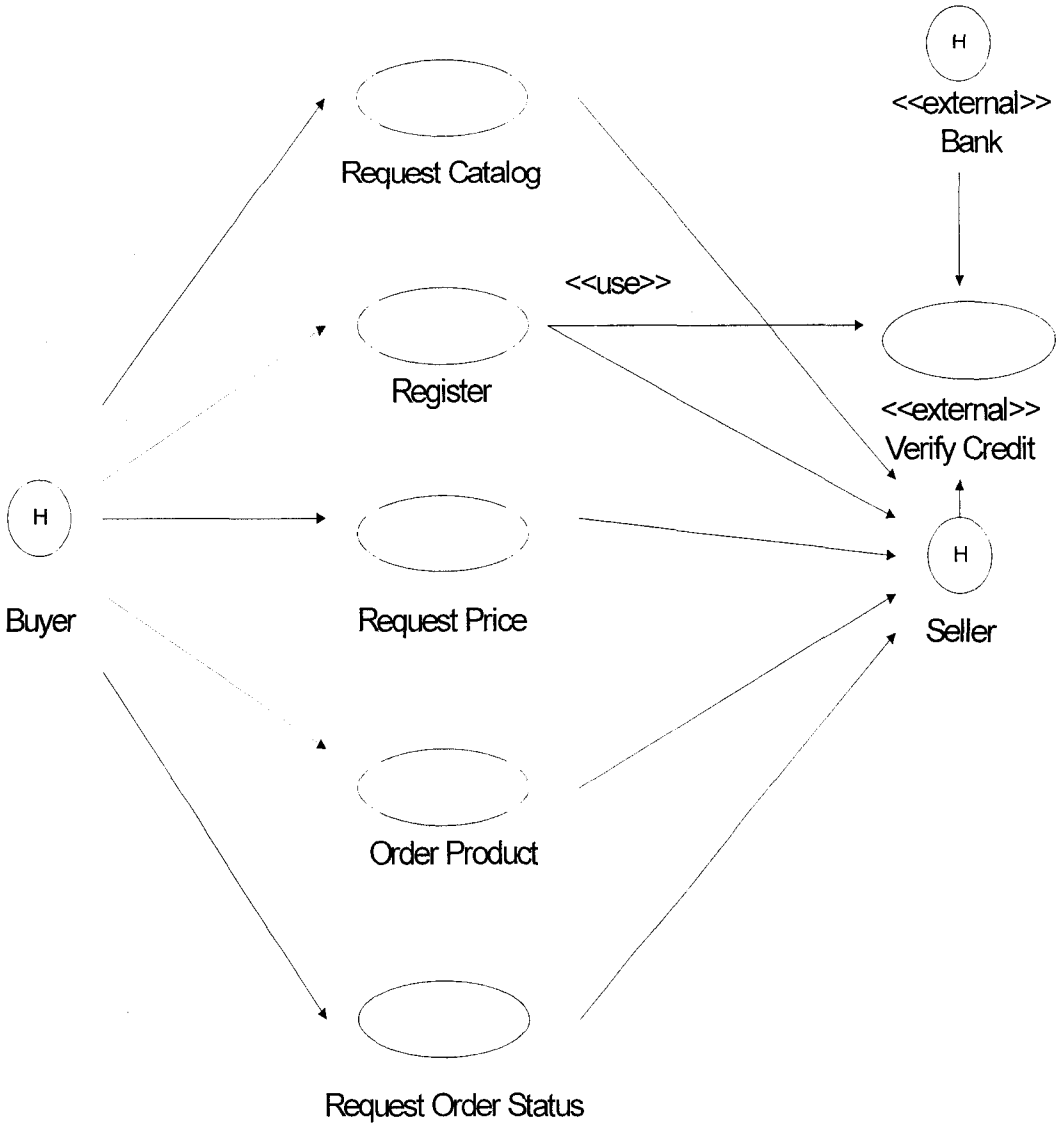


Figure 7.5 Use Case Diagram: order from catalog

Accordingly, the use case 'Verify Credit' must be defined in another system of transactions. Therefore, the use case utilizes it and the interfaces for the bank are stereotyped as 'external'. Nevertheless, it is necessary to analyses the required inputs and outputs from/to the external system. The main function of the requirement workflow is to describe each use case in detail. We have developed a template for the purpose of a detailed use case description. Fig. 7.6 depicts the instantiated template for the use case 'Register Buyer'. The template has been designed to cover the following facts: For each use case the involved interfaces (actors) have to be defined. It must be clear which preconditions must be met before the use case can start and what initiates the start of the use case. Accordingly, one or more events must be specified which terminate the use case. The post conditions met by each of the end states have to be clarified. Between the start event and the end event certain activities have to be fulfilled within the use case. Note that a use case can cover more than one scenario. This means that there might exist different paths through a use case (sometimes leading to different end states).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Use Case Name:	Register Buyer
Summary:	In order to do further business with the Seller (obtain price quotes or order products), the Seller requires the Buyer to register and obtain a Buyer ID. Therefore, the Buyer provides the personal an credit information required for registration, and the Seller issue a Buyer ID
Interfaces/Actors:	Buyer IS, Seller IS (internal) Bank (external)
Preconditions:	None
Begins When	Buyer initiates the Registration Process
Description:	The Buyer initiates the registration process and documents the following information: Bill To details: Buyer name Bill to address (street, city, zip, country) Contact name (first, middle initial, last) Contact phone Ship To details (if different from Bill To info): Ship to address (street, city, zip, country) Ship to contact name (first, middle initial, last) Ship to contact phone Credit card info: Credit card number Credit Cardholder Name Credit Card Issuer Name Credit Card Type Credit Expiration Date Encrypted signature Respond-by date (date by which the Buyer wishes to receive the Buyer ID) The Buyer then sends this information to the Seller. When the Seller receives the request, the Seller checks the respond-by date. If the date has passed, the request is discarded. If the Respond-by date has not passed, the Seller validates the Buyer credit information (Uses Verify Credit Use Case). If the credit information is not valid, the Seller sends the Buyer a rejection notice containing the following information: Rejection reason code Rejection reason description If the Buyers credit information is valid, the Seller creates a Buyer ID for the Buyer. The Seller then sends a notice to the Buyer with the Buyer ID
Ends when:	The Buyer receives a response from Seller, or the respond-by date is exceeded
Exceptions:	None
Post	Buyer has a Buyer ID, a rejection of the Registration Request, or

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

conditions:	the request has been discarded

Figure 7.6 Use Case Description of 'Register Buyer'

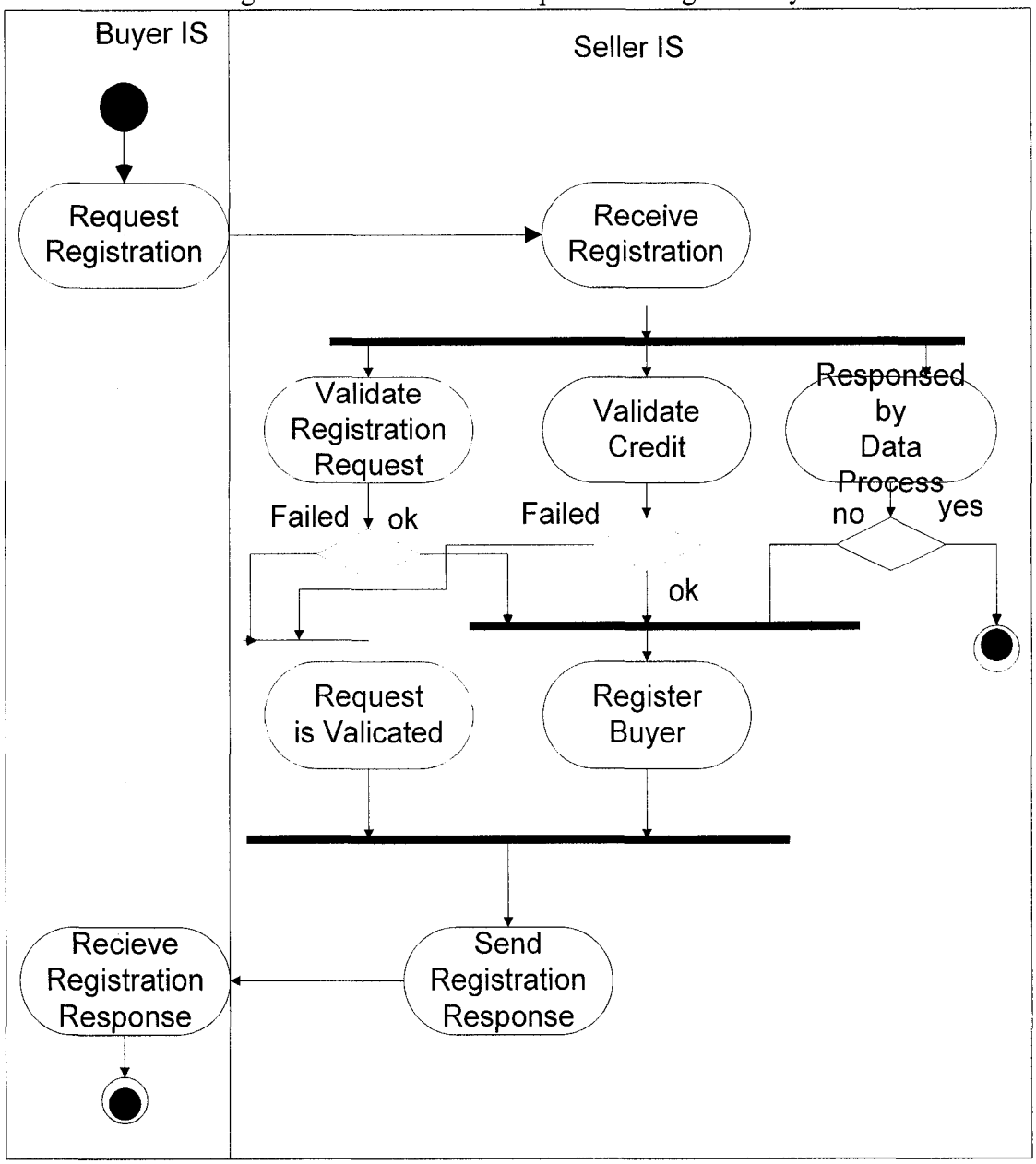


Figure 7.7 Activity diagram of Register Buyer

The use case description has to capture all possible scenarios through a use case. To give a better understanding of the activities performed in a use case the textual description within the use case template is accompanied by an activity diagram for each use case. For each scenario, the activities are given in the order they are regularly performed. It must be evident which conditions/decisions lead to different scenarios. Furthermore, it must be clear which interfaces (actors) are involved in each activity. This can be defined by using swimlanes in activity diagrams (see Fig. 7.7)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(Randy 1996).
Finally, each use case description must cover a description of the business objects that are subject to the activities of the use case. The description in the use case template must allow derivation of the business objects structure in a class diagram.

7.1.2.3. Initial risk analysis

FMEA (Failure Mode and Effects Analysis) is an easy to use and yet powerful pro-active engineering quality method that helps us to identify and counter weak points in the early conception phase of products and processes.
We will use the method (FMEA) to identify all types of risk including commercial risk (failure to give the intended return on investment), business risk (impact on the business of failure of the Product, either before or after deployment), program risk (failure to deliver on time), development risk (Product development is more difficult or costly than expected) and support risk (high cost of user support or maintenance because of product fragility) related to the proposed product. In addition, risks relating to staff implications (users, support staff, management) will also be considered - including operating procedures, redundancy, retraining, morale, re-deployment, new management structures, etc.

System event	Priority Wt	Failure Modes	Likelihood	Detection Mechanisms	Counter Measures
Search for book	54	Title misspelled by user	Medium (or other metric)	Spell check words	Suggest corrections to user
Check out book	72	Bar code misread	Low	Check digit	Request rescan
Find book on the shelf	18	Book is not available	Low	Check record	Find again

Table 7.1 Failure Mode Analyses

This matrix is derived from QFD called Failure Mode and Effects Analysis (FMEA) table. It is a really valuable tool for RUP. It improves the quality, reliability, and safety of processes, increases customer satisfaction and reduces product development timing and cost / support integrated product development.

In addition, FMEA is a much more robust approach to understanding and dealing with failure modes of the system. It also is more appropriate as a reliability enhancement tool.

Finally, we take three use cases as examples to illustrate how to implement FMEA method in this table. We not only describe the failure models that might occur in the library system but also give the solution to that problem. In other words, when possible failures are identified, the details to solve this problem are entered in the FMEA table and dealt with accordingly.

7.1.2.4. UORE (usage oriented requirement engineering)

RUP is based on the use case driven. The use case driven analysis advantage is to help to cope with the complexity of the requirements analysis processes. By

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

identifying and then independently analyzing different use cases we may focus on one, narrow aspect of the system usage at a time. But it still has some disadvantages. The lack of synthesis is probably the main drawback of UCDA. What we really would like to get from requirements analysis is a model that captures the functional requirements and system usage without any design aspects.
In general, UCDA (use case driven analysis) does not fully address the following issue:

- Use cases are not independent. They may overlap, occur simultaneously, or influence each other.
- Use cases occur under specific conditions. They have invocation and termination contexts.
- The level of abstraction of use cases and their length are matters of arbitrary choice.
- The use cases can, in practice, guarantee only partial coverage of all possible system usage scenarios.

So we will integrate some new techniques and further evolve the use case driven analysis so that we can overcome these kinds of limitations. We will extend the use case driven analysis with synthesis phase; moreover, use cases are formalized and integrated into a synthesized usage model. The synthesized usage model captures functional requirements and system usage in a more formal way than user case driven analysis. The Synthesized usage model is intended to be a part of requirement specification, and a reference model for validation and verification.

The process of UORE (usage oriented requirement engineering) consists of two phases, analysis and synthesis. The analysis phase has an informal requirement description as input, and produces the use case model containing description of actors and use cases. This model, in turn, is used as input to the synthesis phase that formalizes the use cases, integrates them, and creates the synthesized usage model.

7.1.2.4. 1Analysis phase

The analysis phase of UORE consists of two interrelated activities (Addison 2002):

1. Identification of use cases and actors.

2. Unification of terminology.

Use case model

(1) Use case specification

In UORE an actor represents a user that belongs to a set of users with common behavior and goals. Unlike objectory, it treats the use case as classes. We regard them just as examples of system usage. On the other hands, each use case describes the system behavior, as seen by one actor only. This single-actor-view approach makes the use case concept simpler. This provides a clear criterion for the construction uses case descriptions and the reduction of associated complexity. In UORE, the description of each use case contains a list of conditions defining a context in which the specific flow of events of the use case can occur.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Here, we already get the use cases, actors from the use case model. Hence, we are going to formalize the use cases, integrate them, and create a synthesized usage model in the synthesis phase.
The synthesis phase consists of three activities:
1. Formalization of use cases
2. Integration of use cases
3. Verification
The formalization activity aims at producing a formal use case specification (UCS) for each use case identified in the analysis phase. The product of this activity is a collection of UCS's, represented in the formal, graphic language of message sequence charts. Each UCS expresses the temporal ordering of user stimuli, system responses, and atomic operations.
The formalization activity has the following steps:
1. Identification of abstract interface objects.
2. Identification of atomic operations.
3. Creation of one UCS for every use case.
The integration activity aims at merging different use case specifications and producing a synthesized usage model. The SUM consists of a collection of Usage views, one for each actor. The integration activity consists of the following three steps:
1. Identification of user and system actions
2. Creation of abstract usage scenarios.
3. Integration of abstract usage scenarios.
The purpose of the verification activity is to obtain a consistent and complete SUM.
There are two verification steps related to the formalization activity and integration activity respectively:
1. Verification of UCS
2. Verification of SUM
The UORE method solves the problems in the use case driven analysis and provides a more useful way to aid in the software process design. Certainly, it will go though the whole process and enhance the quality of the software(Addison 2002):.

7.2New language

UML is a general purpose modeling language, its notation is very powerful to express software system model. UML diagram maybe the most important part in the UML. It can model software system from static and dynamic viewpoint, and consists of lots of different diagrams. Although UML diagrams are very popular notations for software system modeling, it still has some limitations. In this section, we will use some useful additional methods to evolve the language.
Our example will be built around the principal UML diagrams to show how to enhance the diagram's functions in modeling software systems.

Use case diagram

Use case diagram is a very important diagram in UML and RUP development. However, a use case diagram still has some limitations as mentioned above; therefore,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

we will try to integrate QFD-style matrices with use case diagrams in order to solve these problems.

(1) "User X Actor " matrix points out relative importance of the role by value, so we can add that information in use case diagrams to indicate the importance of each role. Therefore, we adopt "Tagged value" (one of UML extension mechanisms) to insert that information into a use case diagram. The benefit of this improvement is to add more details about the role in the use case diagram so that those use case diagrams can represent more information and are easy to understand.

(2) "Actor X Use Case" matrix describes the function weight of each use case. We also adopt the same method in (1) to represent information in the use case diagram.

(3) "User Demand Quality X Use Case" matrix depicts the user demanded quality for each use case. Customer requirements are extremely important in the beginning stage of software design. Normally, customers will bring forward lots of requirements, but not all of them are valuable. So we use a need-opportunity matrix to prioritize user demands, then combine the more valuable customer requirements with the "User Demand Quality X Use Case " matrix, and finally we represent this information in use case diagram by the "constraint" extension mechanism.

(4) We can use the UML extension constraint to represent the information derived from the KANO method described in section 5. What we want to do is to make designers understand the use case easier, and increase the amount of details.

Actually, there are no main limitations in the use case diagram. The only problem is how to map use case diagram into other diagrams because the use case diagram is a high-level description.

We can try to use flow charts to solve these problems because a flow chart has powerful ability to represent. Flow charts tend to provide a different aspect to a process or a task. Flow charts provide an excellent form of documentation for a process, and quite often are useful when examining how various steps in a process work together.

Class diagram and object diagram

Class and object diagrams are directly capable of representing business entities as objects. Objects are logically manipulated during business processes. The shared attributes and operations of objects are defined in classes. A class is a description of a set of objects sharing the same attributes, operations, relationships and semantics. Class diagrams can also represent different types of relationships amongst classes.

Given the static nature of classes/objects, these diagrams seem to be adequate for defining business entities; however classes can be stereotyped in order to enhance class modeling for business organizations.

The information derived from QFD-style matrix is shown below:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(1)"Use Cases X Actor Role" matrix describes the relative importance of each role and also can be represented in class diagram by the "tagged value" extension mechanism.

(2)"Object X Classes" matrixes show the relationship between the objects and superclasses, so all the objects can be represented as a new class by the stereotype extension mechanism. Furthermore, the "Object X Data Attribute" matrix can provide attributes of each class, so we can add them into class diagram. "Object X Object" matrix will describe the relationship among object and can be shown in the class diagram.

Object Diagram

Object diagrams are also closely linked to class diagrams. Just as an object is an instance of a class, an object diagram could be viewed as an instance of a class diagram. Object diagrams describe the static structure of a system at a particular time and they are used to test the accuracy of class diagrams.

Object name : Class
Named object
: Class
Unnamed object
Object name : Class::Package
Named object with path name

Table 7.2 Object Definitions

Object names

each object is represented by a rectangle that contains the name of the object and its class underlined and separated by a colon.

Object Name : Class
Attribute type = 'Value' Attribute type = 'Value' Attribute type = 'Value' Attribute type = 'Value'
Object with attributes

Table 7.3 Objects with Attributes

Object attributes

object attributes are list in a separate compartment with classes. However, unlike classes, object attributes must have values assigned to them.

The information can be captured from QFD-style matrix shown below:

(1)"Object X Data Attribute" matrix, "Object X Object" matrix and " Object X Class"

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

matrix show the common data attributes, association relationships between objects and relationships between the object and superclass. This kind of information can be combined with an object diagram.

(2) "Object X Data Attribute" matrix and " Object X Class" matrix also prioritize the importance of the object, so we can add these details into an object diagram by tagged value.

Interaction diagram

An interaction diagram is a diagram that shows an interaction, consisting of a set of objects and their relationships, including messages that may be dispatched among them. Activity diagrams show the flow from activity to activity while interaction diagrams emphasize the flow from object to object. There are several ways to capture detailed business process information that provide more information in order to supplement use cases. Two of the possibilities within UML are sequence diagrams and activity diagrams.

Activity diagram

An activity diagram illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. Because an activity diagram is a restricted form of state chart diagram, it uses some of the same modeling conventions(Booch 1998).

We use the following method to improve upon the activity diagram:

(1) The disadvantage of traditional activity diagrams is that they do not make explicit which objects execute which activities, and the way that the messaging works between them. The method that we adopt to solve this problem is swimlanes. Swimlanes can group related activities into one column, and then we put the object name on the top of each column to indicate which objects execute which activities. The following diagram shows this method.

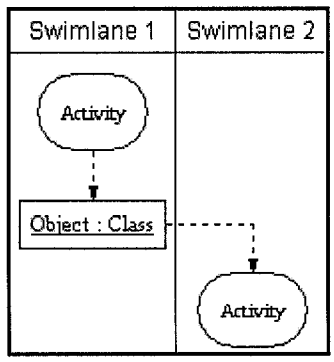


Table 7.4 Activity Diagram

(2) There is no constraint imposed on the nature of the role nor the consistency amongst the roles defined (e.g. all objects, all organizational units, etc.) So this deficiency restricts the application of the concept of responsibility to activity

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

diagrams." User X Actor Role" matrix and "Actor Role X Use Case" matrix can put supply information to set constraints on the objects in the activity diagram, we can use the constraint extension mechanism to represent this information. Moreover, the " Object X Class" matrix shows the relationship between object and superclass; we also can add this detail into activity diagrams. Additionally, to some degrees, " Actor Role X Use Case " matrix shows the order of the use case, so we can add this information into activity diagrams by tagged value extension mechanism in order to model the past state of the object and other units.

Sequence diagrams: captures time-related behavior; and can help optimize message traffic through out the system.
Sequence diagrams are a kind of interaction diagram that emphasizes the time ordering of messages. A message is a specification of a communication between objects that conveys information with the expectation that activity will ensue.

Sequence diagram still have some limitations, we use the following techniques to solve them:
(1) It is difficult to describe all the scenarios of a use case in one sequence diagram. Again, we use swimlanes to improve this diagram.

State diagram
A state chart diagram shows the behavior of classes in response to external stimuli. This diagram models the dynamic flow of control from state to state within a system.

The matrix we can use to suppose the state diagram is shown below:
(1) In the "Actor Role X Use Case " matrix, it shows the functions and operations in which each actor role participates. Each use case represents a coherent, useful function that the system performs for a user playing one (or more) of the actor roles. Referring to the limitation of state diagram, which its events don't correlate with external actors, classes and package; we can use constraint extension mechanism to represent the actors in state diagram(Booch 1998).

7.3 Summary

In this section we propose a new process based on the Rational Unified Process, UML and other techniques. The requirements of modeling using RUP and UML contribute to a consistent design of common business objects in the following way. The business workflow helps in understanding the focused business domain. The requirement workflow describes the specifics of the business domain. The vision and scope statement of the requirement workflow together with a use case diagram and supplementary use case definitions allow the exact identification of the boundaries of a transaction.
Consequently, the adapted Rational Unified Process and UML provide meaningful concepts for modeling transactions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

8.0 Extended example

Now that we have had some understanding about new methods mentioned above, we are going to show how these methods can be applied in enhancing UML and RUP in a hypothetical example (it is just one way directional to apply new methods into UML and RUP).

Elevator Problem

This case study assumes an existence of an elevator company that is designing a new elevator product. The following description summarizes the market for this new product (Dannis):

The elevator system designed in our project is an "ideal" elevator in which some of the technical corners are cut. Our elevator has the basic function that all elevator systems have, such as moving up and down, opening and closing doors, and of course, picking up passengers. The elevator system shall provide vertical transportation, for personnel needs, between two or more floors of a multi-floor office and or residential building. The elevator system shall be able to support transportation for at least 10 and no more than 20 stories. The elevator shall be able to provide transportation for 10 to 20 people per floor. This elevator system shall provide vertical transportation in a rapid, safe, reliable, and cost-effective manner. There are car call buttons in the car corresponding to each floor. For every floor except for the top floor and the lobby, there are two hall call buttons for the passengers to call for going up and down. There is only one down hall call button at the top floor and one up hall call button in the lobby. When the car stops at a floor, the doors are opened and the car lantern indicating the current direction of the car is illuminated so that the passengers know the current moving direction of the car. The car moves fast between floors, but it should be able to slow down early enough to stop at a desired floor. In order to certificate system safety, an emergency brake will be triggered and the car will be forced to stop under any unsafe conditions.

Elevator System Operational Concept

(1) Vision

The new elevator system is directed at the major market niche of standard 10 to 20 story office buildings. This product is not to address the low end and high ends of the 10 to 20-story office building market, but the center of this market. Marketing estimates are that 100,000 of these buildings are being constructed each year. Each such building will require six to twenty elevator cars and associated control systems and maintenance/operations support. This market is envisioned to be very price competitive but requiring that basic threshold of performance and cost is met.

(2) Mission Requirement

The mission requirement for this new product is to capture 20% of the market of new building starts. Since our Company is not currently one of the major market leaders, this mission requirement will primarily have to be achieved by providing superior operational performance at less operational cost than the products of the major vendors. Our performance and cost goals are to have 20% better performance on a weighted performance index at 80% of the operational cost compared to the current products of the major vendors.

(3) Operational Phase Scenarios

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A Passenger (including mobility, visually and hearing challenged) request an service, receive feedback that their request was accepted, receive input that the elevator car is approaching and when an entry opportunity is available, enter elevator car, request a floor, receive feedback that their request was accepted, receive feedback the door is closing, receive feedback about what floor at which the elevator is stopping, receive feedback that an exit opportunity is available, and exit the elevator with no physical impediments.

B A passenger enters the elevator car, as described in 1, but finds an emergency situation before an exit opportunity is presented, and notifies the police or health authorities using that communication equipment that is part of the elevator. Elevator maintenance personnel create an exit opportunity.

C Passengers are receiving transportation in the elevator system when a fire breaks out in the building; the building alarm system sends a signal to elevator system to stop the elevator cars at the nearest floor, provide exit opportunity, and sound a fire alarm. Passengers leave elevator cars. Elevator cars are reactivated by special access available to maintenance personnel.

D Passengers are entering (exiting) an elevator car when doors start to shut; passengers can stop doors from shutting and continue to enter (exit).

E The elevator car stops functioning and sends a signal for service. Passengers in the elevator car push an emergency alarm that notifies building personnel to come and help them. Elevator maintenance personnel create an exit opportunity.

F Too many passengers enter an elevator car and the weight of passengers in the elevator car exceeds a preset safety limit; the elevator car signals a capacity problem and provides a prolonged exit opportunity until some passengers exit the car.

G Maintain a comfortable environment in the elevator by sensing the temperature in the elevator car that is based upon heat loss/gain of the passengers and the building and then supplying the necessary heat loss/gain to keep the passengers comfortable.

H A maintenance person needs to repair an individual car; the maintenance person places the elevator system in "partial maintenance" mode so that the other cars can continue to pick up passengers while the car(s) in question is (are) being diagnosed, repaired, and tested. After completion the maintenance person places the elevator system in "full operation" mode (Dannis).

I Electric power is transferred to the elevator from the building.

8.1 QFD-style matrix

We will adopt QFD-style matrices to analyze the elevator system and further capture more useful information so that we can use them in the subsequent UML diagrams.

Users X Actor Role

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Individual users	Passenger	Elevator Operator	Cleaner	Technician	...
Jan	●				
Pat	•	●			
David	•		●		
Calvin	•			●	
Prioritization of role	3	9	1	1	

Table 8.1 User X Actor Roles

Table 8.1 lists the different roles and the correlation strength between Actor Role and Individual Users. The circle in the matrix indicates the correlation strength between Actor Role and Individual Users. ● is higher than •. The bottom row shows the relative importance of the roles. In this example, the Elevator Controller role has the highest priority, followed by the Cleaner, Technician and Passenger. In this thesis, we use the 9/3/1-rating scheme for both customer requirements and their relation to metrics.
In the top row of this table, we list the actor roles that carry out the use cases of the elevator system. However, the actor roles still need to be further validated. I.e. when people go through this table, they want to know the specific person shown in the elevator system so that they can understand the table concretely. In addition, the actor roles in the table also indicate the usage of use case and how many different people shown in the elevator system.

In fact, the "Prioritization of role" in the last row of the matrix is optional; it really depends on what the analysis team needs and what kinds of methods they want to adopt.

Table 8.1 not only adopts the quantitative analysis to indicate the Prioritization of actor roles but also uses the circles to illustrate the association among the actor roles. In this matrix, some individual users have more than one-actor roles. Therefore, we can find that some actor roles are a subclass of other actor roles. In this case, cleaner, elevator controller and technician are the subclass of passenger.

Table 8.1 is easy to assemble by interviewing individuals and recording the actions performed in their daily activities. The circle and numbers in the matrix can allow designers to more exactly and concretely understand the importance of the actor roles and further ensure how to allocate the workload within the project. The matrix can be incrementally developed because it is too difficult for people to categorize all their activities extemporaneously.

Actor Role X Use Case

Use Case

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Actor Roles	Control Elevator	Request Elevator	Call for Help	Fix Elevator	Activate Elevator	Clean Elevator
Operator	●I		●S		●I	
Passenger		●I	●I			
Cleaner	●I					●I
Technician	●I			●I		
Door	●I					
Motor	●I					
Function WT.	85	27	108	9	81	9

Actor Roles	Open/Close Door	Stop Elevator	Go up/down	...	Role wt.
Operator	●I	●I	●I		9
Passenger			●S		3
Cleaner					1
Technician					1
Door	●S				1
Motor		●S	●S		1
Function WT.	90	90	90		

Table 8.2 Actor Role X Use Case

In the Table 8.2, we show the function that each actor role performs and depicts the reliance between the actor roles and the use cases. In addition, we calculate the function weights for each use case. Using the standard encoding of ●= 1 and ⬤=9 for the correlation symbols, and the role weights derived from the Users × Actor Role table, we arrive at the function weights by multiplying the correlation strength and summing the columns. Furthermore, the value shown in the bottom row of the table is good to test resource allocation, for example, the use case " fix elevator " and "clean elevator" will get less and limited test resource because of their low score. Also, there are some others symbols for example "I" and "S" in the matrix. In fact, these symbols mean that some use cases are initiated by one actor role, but allow the participation of other roles in the processes of execution. For example, the role of Door initiates the use case control elevator, and then provides a service to the Elevator Operator during

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Open/Close Door.

A use case is a specific way of using the system to accomplish an identifiable task. This matrix lists the main use case of the system and points out what takes place between the system and actors. In additional, it shows what actor roles are the initiators of the user case and what actor roles will get service from the use case. Therefore, we can know the association amongst the use cases described in this matrix.

User Demanded Quality X Use Case

	Use Case					
Demanded Quality	Control Elevator	Clean Elevator	Fix Elevator	Open/close elevator	Go up/down	...
Must be fast		●				
Must be proficient	●					
Can't make mistake			●			
Keep a appropriate speed				●	●	
...						

Table 8.3 User Demanded Quality X Use Case

In the Table 8.3, we show the user-demanded quality characteristics corresponding to the some use cases that the system performs. All the demand quality attributes shown in the matrix are important information for UML diagrams. In Chapter 8 case study, we will try to integrate this kind of information with the cost benefit chart and the use case diagram.
In fact, OOA lacks this kind of information, so this information will be complimented for OOA. In additional, this matrix is very useful for designers when they consider the market service and competitive advantages.

Use Cases X Data Attributes

	Data Attributes					
User Case	Floor	Speed	Weight	State	Direction	...
Control elevator	●	•	●	●	●	
Request elevator				●		
Open/close door				●		
Go up/down	●	●		•	●	

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Stop elevator	●	●		●	●	
...						

Table 8.4 Use Cases X Data Attributes

The above table shows different data attributes used in various use cases, for example floor, speed; the state of data attributes -for example, emergency-stop state, door state and request-floor state.
Finally, we will have all the information needed to construct the process model by combining this matrix with the Use Case × Classes and Classes × Data Attribute matrices. Therefore, that information can be used in UML diagrams to describe the use case or class attributes.

Classes X Data Attributes

	Data Attributes									
Classes	Name	Weight	Work category	Employee ID	Shape	Color	Sex	Height	Usage	...
Passenger	●	●	●	●			●	●		
Technician	●	●	●	●			●	●		
Cleaner	●	●	●	●			●	●		
Operator	●	●	●	●			●	●		
Carcallbutton					●	●			●	
Hallcallbutton					●	●			●	
Carpositionindicator					●	●			●	
CarDirectionIndicator					●	●			●	
...										

Table 8.5 Classes X Data Attributes

The Table 8.5 describes different classes and data attributes for each class. We can easily get the superclass by explicitly identifying classes that share data attribute definitions.
This matrix might provide some details to the system description and will be valuable producing a class diagram.

Classes X Classes, Showing Entity Relationships

	Classes				
Classes		Button	Indicator	Door	Motor
Button			1		

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Floor		0.2	0.2	1	
Door					
Light		0.n	0.n		

Table 8.6 Classes X Classes

The Table 8.6 shows the association relationship between pairs of entities in the system e.g. "A Button is associated with zero to many Floor", or "A Floor is associated with exactly one Door." This is an important relationship used in defining the set associations among classes. We can represent this kind of information among classes into UML diagrams for example (sequence diagram, collaboration diagram and classes Diagram). The UML extension mechanism is the best way to describe this information.

Classes x Superclasses

		Superclass	
Classes	Button	Indicate	...
CarCallButton	●		
CarPositionIndicator		●	
HalCallButton	●		
CarDirectionIndicator		●	
...			

Table 8.7 Classes x Superclasses

This Table 8.7 shows the relationship between classes and superclass and provides clear information to tell all (each) superclasses when multiple inheritance should be considered.

Use Cases X IEEE Quality Factors

	Quality Factors						
Use Case	Efficiency	Integrity	Reliability	Survivability	Usability	Correctness	Maintainability
Control elevator	●		●		●	●	●
Request elevator			●	●		●	
Call for help			●	●		●	

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fix elevator	●		●			●	
Activate elevator	●					●	
Clean elevator	●						
Open/Close Door			●			●	
Go up/down			●			●	
Stop elevator			●				

Quality Factors

Use Case	Verifiability	Expandability	Flexibility	Interoperability	Portability	Reusability	...
Control elevator				●			
Request elevator	●						
Call for help	●						
Fix elevator							
Activate elevator	●				●		
Clean elevator					●		
Open/Close Door	●			●			
Go up/down	●			●			

Table 8.8 Use Cases X IEEE Quality Factors

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The Table 8.8 represents the use cases and their related quality factors. In fact, this matrix can be used to map the functions of the system to a set of quality factors.

The quality factors of a system are not essential parts to rate the system's quality. They are only necessary condition to reach the goal. However, the general and standard quality factors will be very helpful for designers and engineers to implement the system and enhance the software quality. The information derived from the table about quality factors is really useful. We can adopt tag values and constraints methods (UML extension mechanisms) to represent them in UML diagrams(Lamia).

8.2 Requirement workflow

Customers' needs are the most important parts in the beginning stage of the software system design. However, the designers need to evaluate all of the needs provided by the customers so that they can make sure what is the priority in the whole process. In the following section, we try to define one method to classify and prioritize the customers' need.

8.2.1 Cost benefit analysis

8.2.1.1 Gather domain knowledge

First of all, we will use a technique such as contextual inquiry to document the business goals and interviews with users, and then we will employ a technique such as personas to organize the information.

In fact, this step has the greatest variability in duration. If we have only limited access to our customers and domain knowledge, it may take us a few hours. Otherwise, it may consume a few weeks.

Contextual inquiry

Contextual inquiry (Beyer 1998) is best used in the early stages of development, since a lot of the derived information is subjective--how people feel about their jobs, etc. In addition, Contextual inquiry is a hybrid between face-to-face interviews and observations in which the customer and the researcher are equal partners in investigating and understanding the usage of a product.

In our example, using contextual inquiry, we need to visit some companies and see how their elevator system works. We need to take in not only physical arrangements such as the location of the elevator system, or the structure of the motor inside, but also an operation mechanism, such as how to process customer requests or the level of an emergency. All of this will help to define a context for their activities and implement the design of the elevator system. Furthermore, We also need to ask questions to the users and listen to their gripes about their existing product; the customers consist of passengers, technician, cleaners, and elevator operators. In fact, they are the end-users of the elevator system. We can ask them what would make their jobs easier; what design changes would help them because they are partners in the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

design process.

Contextual inquiry			
Location ① • Inside/outside • Problems • Safety issues • Environment(temp, humidity) • Period of time-usage, frequency of use	People ② • Who else uses the product? • Who has an important relationship with the user?	Culture ③ • What are the working methods? • What styles of communication prevail? • What is the operation mechanism to process request?	Value ④ • What values are important to the users? • What does the customer like, dislike, hate, love, tolerate, desire etc? • What represents success and failure?

Table 8.9 Contextual inquiry

Results

The following results are the notes that we took during the interviews with the end-users:

Passengers: they are the people who reach a destination by elevator.

Maybe they are the student who goes upstairs or downstairs by elevator, the resident who are in or out by elevator or the white-collar worker who access their work by the elevator.

Passengers' words:

1. I think placing a mirror in the elevator would make passengers feel more comfortable.
2. I always feel dizzy when I take elevator, so maybe the speed of the elevator is too fast.
3. The work efficiency of the technician and cleaner is very low.

Elevator operator: they are the people who operate the elevator

They utilize the operation panel, keys and telephones to implement their work.

Their work environment is limited to the room of elevator.

Operators' word:

1. The operation panel of the elevator should consist of an emergency button and telephone for emergency calls.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Cleaner: The cleaner's job is to clean the elevator
They use the duster cloth, vacuum and detergent to finish the job

Technician: They are the people who fix the elevator's problems
They utilize the meter, screwdrivers and forceps to complete the work.

So what we captured from the notes is shown below:
The elevator has an Up arrow button and a Down arrow button placed, depending on the floor. On pressing the button it turns on. A display is provided for the potential passenger and the passenger to identify the moving direction and the current floor. When the elevator car is moving in the opposite direction of the request and has no passenger getting off at the requested floor but has few passengers going ahead, will not come to a halt during that direction. It is equipped with sensors and smoke detectors for the passenger's safety. If the total weight on the elevator car exceeds the maximum capacity, it displays a request message asking a passenger to step out. A board is placed inside the elevator showing "Maximum capacity 10 passengers (or) 2000lbs". The elevator halts at each floor for which the floor button is turned on as it passes through. When passengers experience a emergency, they can press the emergencies button or use the telephone to call for help. There are some room that can be used to install mirror or a picture frame in order to alleviate passengers' mood. The elevator should keep at an appropriate speed to avoid making the passenger uncomfortable as it goes up/down.

In conclusion, the main difference between the contextual inquiry and the traditional interview is that the inquiry demands a partnership between customer and the product development team. The product developers bring special product knowledge, and the customers bring special knowledge about their activity or specific needs. Both can be viewed as experts and the inquiry is a joint search for information.

Personas

In software development, a thorough understanding of end-user's needs is paramount. Throughout the design phase of software development, a wide variety of design decisions must be made. Each of these decisions considers options that could either advance or hinder the ease of use of the end product. So User Personas are defined to illustrate aspects of different types of people that will be involved with the software. This section of the document identifies and describes the various User Personas that will be considered during the design stages of the product.

The way you communicate the personas and present your deliverables is key to ensuring consistency of vision. Without that consistency, you'll spend far too much time arguing with your colleagues about who your users are rather than how to meet their needs (Alan).

Creating personas

1. Develop a list of personas

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Our persona investigation for the elevator system began by gathering the knowledge of the elevator system. We spent several weeks visiting the companies and interviewing the users, figuring out the operation mechanism and features of the elevator. The information reaped some useful demographic findings about our target user group and helped define the interview sample for building personas.

At last, we were good to go—our market research review was completed and our persona interviews were lined up and scheduled. All in all, the persona interviewing process took about 3 weeks to complete. Interviews ran 1-2 hours each, and most were rich with details. Based on the subject interviews' goals, we created three personas for the project: Bob, Robert, and Annie.

The goal is to create between three and twelve personas for a project, with one to three of these selected as the primary persona(s). The primary persona is the individual whose needs drive the design.

The personas will describe the profile of various individuals involved with and affected by the product, not just the person sitting at the computer using the software constantly. Each persona is a fictional character; this is very important. Even though it would be easy to just take a picture of the client and users for the personas, in practice this approach is not effective. Real people may share characteristics, but each has unique quirks. By taking a composite of characteristics, the persona becomes a more useful tool (Alan).

2. Define the personas

The Persona Chart for Bob (Primary Persona)
Gender: male
Age: 35
Occupation: Controller
Home life: Divorced, no children.
Education: BS in power
LIFESTYLE
Activities: Goes out to dinner twice a week, four times a month for a nice dinner and a bottle of wine with a girlfriend. Fishes at local lakes, canoes, hikes, tries to take his kids on a different outing each weekend "to keep our time together special."
Ultimate goal: To discover new things to do with girlfriend. To be a good, caring father in an increasingly crazy and busy world.
Email: blade@hotmail.com
Quote: "I'm an explorer. I'm the kind of guy who wants to know every road in the county and where it might take me."

Robert
Gender: Male
Age: 27
Education: BS electronic Engineering from Texas Tech

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Occupation: Design Engineer I, 4 to 5 years
LIFESTYLE
Activity: Spends _ time in the office, _ time in the field, Spends 3 hrs/day avg. on the computer
Skill: Relatively familiar with computers; Level of Occupational Experience: moderate
Level of Computer Experience: moderate; Software Run-time Tolerance: low
Software Reliability Expectations: high
Ultimate Goals:
1. Make boss happy.
2. Leave work at 5 every day.
3. Low failure rates.

Annie
Gender: Female
Age: 42
Education: high school
Occupation: cleaner
LIFESTYLE
Activity: Spends 2/3 time in the office, 1/3 time in the field, Spends 2-hrs/day avg. on the computer
Ultimate Goals:
1. Delegate as much work as possible
2. Leave work before dark
3. Get promoted to Senior Management level

In this product, Bob has been the primary persona who has many more chances to interact with the system. All the activities of Bob will affect the performance of the elevator. Compared to Bob, other personas have less effect on the product.

Now we have a better idea of how to document your personas. In fact, It is simply an important step on the way to designing and building better products. Also, what is described above is a cookie-cutter approach towards documenting your personas. Each project will have different documentation requirements to make different points, but the underlying principles stay the same. As a designer, it is up to you to determine how much persona detail is sufficient and how to set up the personas and their presentation so that you pre-empt confusion and questions. You also need to provide a quick way to familiarize newcomers to the persona set, and find ways for your colleagues and clients to keep focused on the personas throughout the project (Alan).

8.2.1.2 Evaluate the product

For the elevator system, we would adopt a heuristic evaluation that can allow specialists to use a list of heuristics (Nielsen 1994), or guidelines, to evaluate products in the design process. Certainly, any other techniques could be used in this step such as usability testing, interviews, walkthroughs, and surveys.

We will assemble some evaluators that include members from different groups (at

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

least one member from Engineering and Design Groups) to do this evaluation. In fact, members from other different groups enhances the quality of the evaluation as each person approaches the product with a different perspective and finds problems related to his or her discipline.

These are ten general principles for user interface design. They are called "heuristics" because they are more in the nature of rules of thumb than specific usability guidelines.

Visibility of system status
The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
No, the visibility of this system is not good enough to inform users what is going on next.
For example, passengers don't know what happen in the next, how long they will wait, and what the elevator system processes after they press the hallpositionbutton to request. So the elevator system will be better if they install the direction indicator, position indicator and time indicator in each floor.

Match between system and the real world
The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

Yes. These buttons illuminate when pressed and cause the elevator to visit the corresponding floor. In addition, the door will retrieve automatically when it senses that people enter.

User control and freedom
Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Supports undo and redo.

Yes, the system provides service for supporting undo. For example, these buttons illuminate when pressed, and then the illumination will be cancel if pressed again.

Consistency and standards
Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
No. The elevator has a set of buttons, one for each floor. Furthermore, there are some notations beside the button to explain the usage of the button.
Error prevention
Even better than good error messages is a careful design that prevents a problem from occurring in the first place.
Yes. If the total weight on the elevator car exceeds the maximum capacity, it displays a request message asking a passenger to step out
Recognition rather than recall
Make objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Yes. The elevator is relatively simple system so that the customer uses the system easily.

Flexibility and efficiency of use

Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

No. The elevator doesn't provide that service to speed up the interaction.

Aesthetic and minimalist design

Dialogues should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Yes. We will install mirror and picture frame inside the elevator.

Help users recognize, diagnose, and recover from errors

Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

Yes. The elevator informs the user by different rings if errors occur.

Help and documentation

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

Yes. There are emergency buttons and telephone used to call for help in the elevator system.

8.2.1.2.1 Estimate system requirements

The mission requirement for our elevator system is to capture 20% of the market of new building starts. Furthermore, our performance and cost goals are to have 20% better performance on a weighted performance index at 80% of the operational cost compared to the current products of the major vendors. Thus, we will accurately estimate the features of our new elevator system in the following sections.

A. Each elevator has a set of m buttons, one for each floor. Each floor, except the first floor and top floor has two buttons, one to request an up-elevator and one to request a down-elevator.

B. These buttons illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when an elevator visits the floor.

C. The door will retrieve automatically when it senses that people enter.

D. The elevator has the button for emergency

E. Each elevator has a telephone.

F. When an elevator has no requests, it remains at its current floor with its doors closed

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

G. Each elevator has a mirror

H. Each elevator has a picture frame.

I. The operator must be proficient in controlling elevator
J. The elevator goes up/down at an appropriate speed.
K. The elevator has the direction indicators, position indicators and time indicators in each floor.
L. These buttons illuminate when pressed, and then the illumination will be cancelled if pressed again
M. There are some notations beside the button to explain the usage of the button.
N. If the total weight on the elevator car exceeds the maximum capacity, it displays a request message asking a passenger to step out.
O. The elevator informs the user by different rings if errors occur.

(1) Collect cost data

Cost data must be collected for estimating the features of the new elevator system. Three sources of data are historical organization experience, current system costs and market research. This is one of the most difficult steps in a CBA (cost benefit analysis), but also on of the most important.

a. Historical Organization Data

Historical data of the former elevator system may be used to estimate the purchase prices of software and services relate to some features of the new elevator system. The numbers will probably need to be adjusted to account for differing quantities and qualities for the proposed system. In our company, we have the past 10 years historical data that can be reference to support estimating features of our current elevator system.

b. Current System Costs

Our company will take 3 years to implement the new elevator system, hence the hardware and software for the features design will be account for the current system cost of our new system.

c. Market Research

Market research also is an important factor used to estimate the cost of the feature of our system. Our company just captures 10% of the market in the last year because we are not the market leaders. However, we try to capture 20% of the market in 3 years by integrating some new features with the new elevator system. We will provide a realistic price for our elevator system based on the market research.

(2) Estimate costs

Many factors must be considered during the process of estimating the costs of features design associated with our elevator system. The following factors will be addressed: Activities and Resources, Personnel Costs, Indirect Costs, Depreciation, and Annual Costs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a. Activities and Resources

Identify and estimate the costs associated with the initiation, design, development, operation, and maintenance of features. Our approach is to identify the activities performed and estimate the cost of the resources associated with each feature. The activities identified below should be addressed.

- Features Definition
- Work Process Evaluation
- Cost Benefit Analysis
- Features Implementation
  - Design
  - Development
  - Operation
  - Maintenance
- Features Performance Evaluation

A sample list of activities and the required resources (cost elements) is provided below.

ACTIVITY	TASK	COST ELEMENTS
Features Initiation	Features Definition	Analysts, Managers, Processors, Customers
	Work Process Evaluation	Analysts, Managers, Processors, Customers
	Processing Requirements Definition	Analysts, Managers, Processors, Customers
	Prepare Cost Benefit Analysis	Analysts, Managers, Processors, Customers
Features Design	Develop features Design	Analysts, Managers, Processors
	Approve features Design	Analysts, Managers, Processors
Features Development	Develop and Test Programs and Procedures	Analysts, Managers, Processors, Programmers, Computers, Software
	Develop Transition Plan	Analysts, Managers, Processors,
	Implement New System & Procedures	Analysts, Managers, Processors, Programmers, Computers, Software
Features Operation	Operate New System	Analysts, Managers, Processors, Programmers, Computers, Software
Features Maintenance	Correct Errors & Make Changes to the System	Analysts, Managers, Processors, Programmers, Computers, Software
Features Evaluation	Evaluate System Performance Compared to Expectations	Analysts, Managers, Processors, Customers

Table 8.10 System Cost Matrix

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

b. Personnel Costs

There are 20 personnel in our company. Their job is to complete the main process of features design. The reference data used to estimate personnel costs is from the former elevator evaluation.

(1) The total personnel cost for full or part-time employee is 290,340, broken down as follows:

(a) The total personnel cost for full or part-time permanent employee 160,890.

(b) The cost factor to be used for employee insurance and health benefits based on actual cost are 10,480.

(c) The cost factor to be used for employee miscellaneous fringe benefits (workmen's compensation, bonuses and awards, and unemployment programs) is 93,650.

(2) The total personnel cost for intermittent or temporary employees is 25,320.

	full or part-time employee	intermittent or temporary employees	Personnel costs
A	10,874	1,138	12,012
B	13,384	1,273	14,657
C	12,877	1,333	14,210
D	11,432	1,088	12,520
E	27,393	2,734	30,127
F	26,938	2,683	29,621
G	22,394	1,034	23,428
H	20,388	1,188	21,576
I	18,384	1,634	20,018
J	17,449	1,476	18,925
K	28,283	2,864	31,147
L	21,999	1,544	23,543

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

M	16,394	1,393	17,787
N	15,757	1,255	17,012
O	26,394	2,683	29,077
Total	290,340	25,320	315,660

Table 8.11 personnel costs

c. Indirect Costs

Direct costs, such as direct labor and direct material, are costs incurred in a process that is "hands on," that directly produces the output. Indirect costs (often referred to as overhead costs) are incurred in a support role (all costs that are not direct). Typical overhead items in our new elevator system are indirect labor, indirect material, and fixed costs such as rent, advertising, taxes, and utilities. Overhead is often expressed as a percentage of direct labor. In our elevator system, the indirect costs are 58,068.

	Indirect Costs
A	3,283
B	3,455
C	3,755
D	3,133
E	5,539
F	5,344
G	3,133
H	3,344
I	3,211
J	3,699
K	5,699
L	3,199
M	3,699
N	2,087

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

O	5,488
Total	58,068

Table 8.12 Indirect Costs

d. Depreciation

Depreciation is defined as lowering the estimated value of a capital asset (usually only those items valued at $+5,000 or more). Depreciation is also defined as the method used to spread the cost of tangible capital assets over an asset's useful life (the number of years it functions as designed). It is computed by comparing the original cost (or value) with the estimated value when it can no longer perform the function(s) for which it was designed, its residual or salvage value.

	Depreciation
A	349
B	391
C	344
D	324
E	560
F	540
G	329
H	349
I	339
J	379
K	579
L	329
M	359
N	319
O	510

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Total	6,000

Table 8.13 Depreciation

Table 8.13, Tangible Asset Depreciation, illustrates straight-line depreciation of a $11,000 asset (hardware, software and office supply), with a useful life of 3 years, and a residual or salvage value of $5,000.

e. Activity Costs

All cost elements must be identified and estimated for each year of the system. This is necessary for planning and budget considerations. Table 5, Activity Cost Matrix, illustrates the cost estimates for features for the elevator system.

	Initiation	Design	Development	Operation	maintenance	evolution	Cost
A	3,483	5948	47033	10,874	4,887	5,837	78,062
B	3,455	5784	46744	10,384	4,777	5,787	76,931
C	3,555	5988	45776	10,877	4,662	5,982	76,840
D	3,133	5388	46558	10,432	4,776	5,666	75,953
E	5,239	9103	71664	20,393	6,838	7,838	121,075
F	5,344	9073	72659	20,638	6,836	7,766	122,316
G	3,533	5894	45593	10,394	4,677	5,377	75,468
H	3,544	6539	47885	10,388	4,772	4,922	78,050
I	3,611	5936	46852	10,384	4,562	5,442	76,787
J	3,560	5734	47331	10,409	4,772	4,822	76,628
K	5,279	9009	71659	21,283	6,873	7,963	122,066
L	3,699	5564	47343	10,999	4,856	4,886	77,347
M	3,690	5700	46325	10,394	4,983	4,743	75,835
N	3,587	5165	46249	10,757	4,856	4,986	75,600
O	5,288	9175	70329	21,394	6,873	7,983	121,042
Total	60,000	100,000	800,000	200,000	80,000	90,000	6,100,000

Table 8.14 Activity Cost Matrix

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Personnel costs	Indirect costs	Depreciation	Activity costs	Total costs	Costs of the major vendor
871,020*	58,068	6,000	610,000	1,545,088	1,931,360

Table 8.15 total cost of features

In the table 8.15, we estimate total cost of features of our elevator system. We finally can reduce 386,272 for the operation cost and that is 80% of the operation cost of the major vendor.

* It is the personnel cost for 3 years i.e. 290,340 * 3=871,020.

(3). ESTIMATE BENEFITS

Identifying and estimating the value of benefits will probably be the most difficult task in the CBA process. Six specific activities are addressed in this section.

a. Define Benefits

Benefits are the services, capabilities, and qualities of each system, and can be viewed as the return from an investment. Some benefits for elevator systems are:

Accuracy – our elevator system provide better accuracy by reducing the number of data entry errors or eliminate some data entry that would, in turn, result in fewer data entry errors.

Availability – we will spend 3 years in developing and implementing the system.

Efficiency – our elevator system provide faster or more accurate information processing.

Maintainability - the maintenance costs of our system is less than others.

Reliability – our system can provide greater hardware or software reliability.

b. Identify Benefits

In fact, benefits are from both the company and its customers. Normally, the benefits to the customers will be much less than the benefits for the company that is developing the system.

Some benefits for our company include flexibility, system strategy, risk management and control, organizational changes, and staffing impacts. In our elevator system design, we won't allow personnel to perform jobs with little or no extra training and try to implement the system on schedule.

Possible benefits to customers include improvements to the current services and the addition of new services. The current services include 7 x 24 service, telephone inquiry and online technician for solving the problem. On the other hands, our company will provide some new services to help customers to understand our

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

elevator product in depth. For example, our company has created a website to introduce our elevator product; hence customers can capture all the information for example operation mechanism, structure, and function of the elevator system by the website.

c. Estimate Intangible Benefits

We will use a subjective, qualitative rating system to estimate the intangible benefits. The entire rates are done by a group of specialist familiar with the current elevator system. Having five people do the evaluation would be ideal, and three evaluators should be a bare minimum. The numerical values assigned to the ratings then will be summed and averaged to obtain a score for each benefit. Table 6, Quantify Benefits, shows the scores for benefits A - O from four reviewers using a scale of 1 to 5.

Benefit	Reviewer 1 Score	Reviewer 2 Score	Reviewer 3 Score	Reviewer 4 Score	Reviewer Average Score
A	5	4	4	3	4.00
B	4	5	3	4	4.00
C	5	5	5	4	4.75
D	4	3	5	5	4.25
E	5	3	4	5	4.25
F	3	4	5	5	4.25
G	5	4	5	5	4.75
H	5	5	5	4	4.75
I	5	3	4	4	4.00
J	4	5	4	3	4.00
K	4	5	5	5	4.75
L	4	3	5	5	4.25
M	4	4	3	3	3.50
N	3	3	5	3	3.50
O	4	4	3	3	3.50
					4.27

Table 8.16 Quantify Benefits

Finally, we got average score 4.27 from the rating system. Compared to the score 3.56 of the major vendor, we have 20% better performance than they do.

8.2.1.3 Categorize the Problems

We will use affinity diagramming to classify the problems into manageable categories because of the number of problems identified in the evaluation can be very large. This technique provides us with a chance to see patterns in the problems—which problems tend to group together and which seem related. These patterns, in turn, give us a more

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

integrated view of the problem space. Instead of focusing narrowly on each individual problem, we can expand our focus and get a high-level view of all the problem areas.

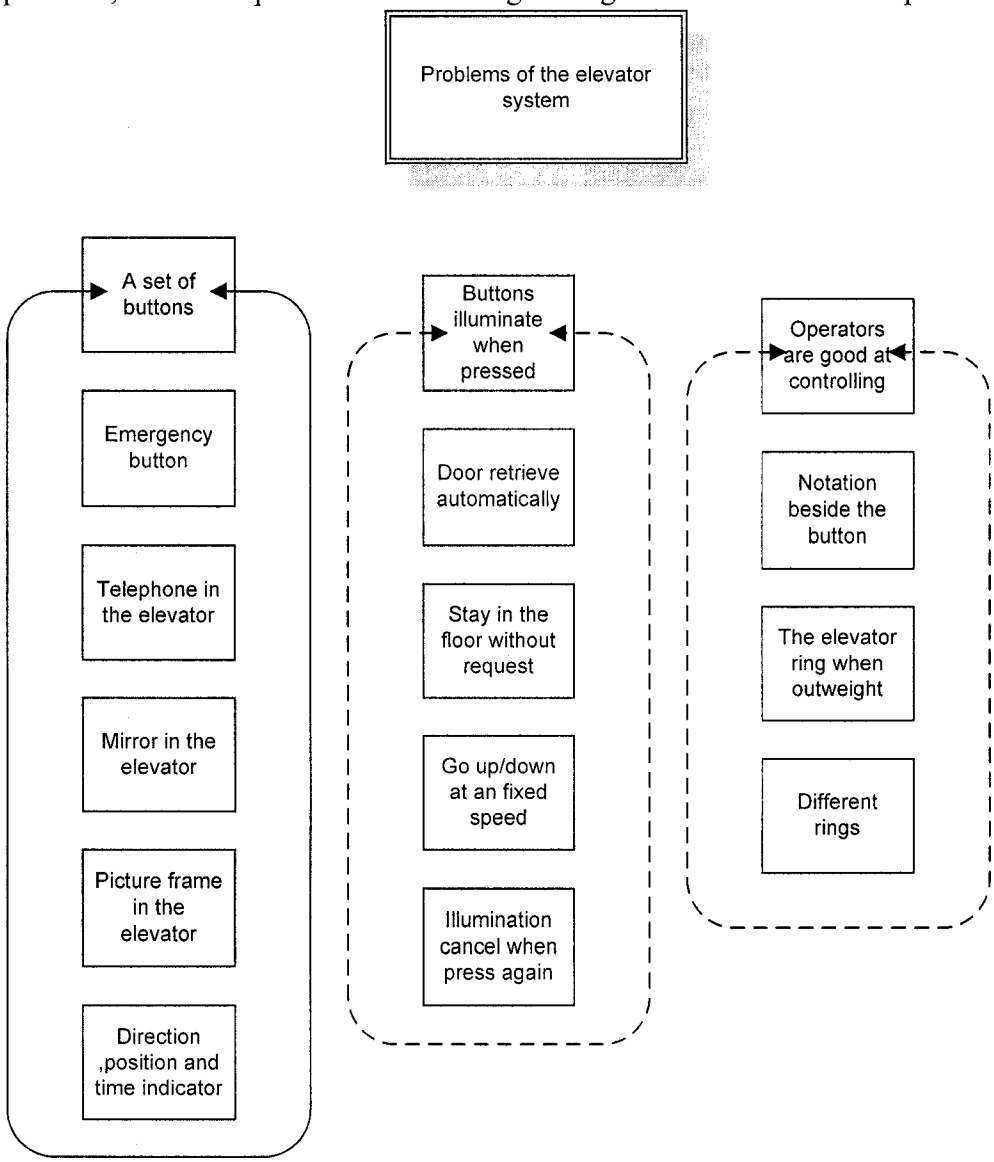


Figure 8.1 Affinity diagramming

In the following part, we list some categories related to the elevator system.

A. Each elevator has a set of m buttons, one for each floor. Each floor, except the first floor and top floor has two buttons, one to request and up-elevator and one to request a down-elevator.

B. These buttons illuminate when pressed and cause the elevator to visit the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

corresponding floor. The illumination is canceled when an elevator visits the floor.

C. The door will retrieve automatically when it senses that people enter.

D. The elevator has the button for emergency

E. Each elevator has a telephone.

F. When an elevator has no requests, it remains at its current floor with its doors closed

G. Each elevator has a mirror

H. Each elevator has a picture frame.

I. The operator must be proficient in controlling elevator
J. The elevator goes up/down at an appropriate speed.
K. The elevator has the direction indicators, position indicators and time indicators in each floor.
L. These buttons illuminate when pressed, and then the illumination will be cancelled if pressed again
M. There are some notations beside the button to explain the usage of the button.
N. If the total weight on the elevator car exceeds the maximum capacity, it displays a request message asking a passenger to step out.
O. The elevator informs the user by different rings if errors occur.

8.2.1.4 Prioritize the Categories

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

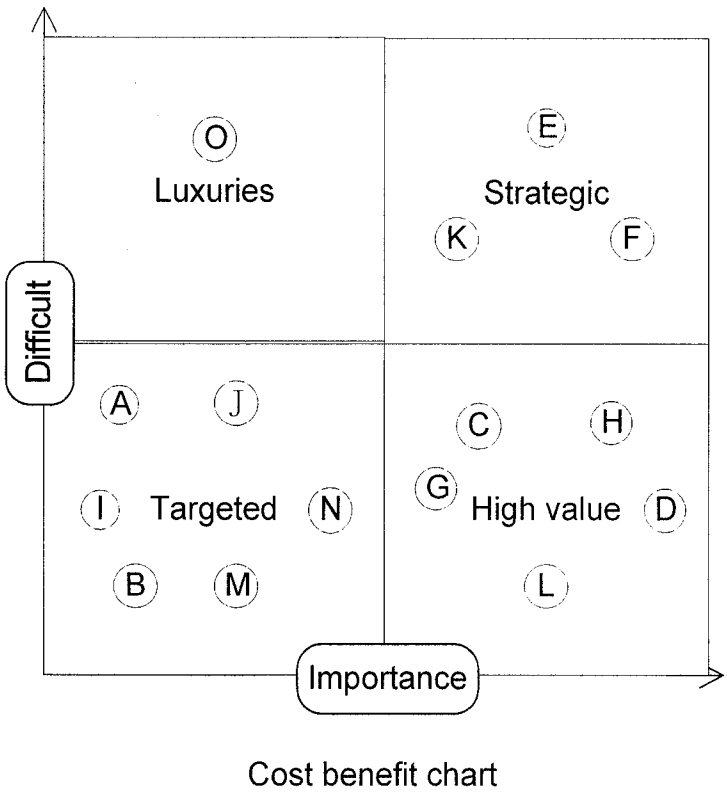


Figure 8.2 Cost Benefit Chart

We put the customer's needs into different quadrants based on the importance and difficult associated with them. According to this chart, the emphases should be in "high value" quadrants and strategic. The needs in the "targeted" quadrants are the basic requirements that we need to meet; however, the ones in the "high value" quadrants will add more value features into the design. On the other hand, the needs in the "strategic" quadrants should be considered carefully if designers want to implement it.

Feature C, H, G, L and D can add more value into our elevator system for example, Feature L (These buttons illuminate when pressed, and then the illumination will be cancel if pressed again) provide a mechanism to allow customers to change the request that they make by mistake.

Feature A, B, I , J, M and N are the most basic requirements that must be implemented in our system for example, the elevator can't work very well if there is no feature B (These buttons illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when an elevator visits the floor) in the elevator system.

O is the feature that doesn't have to be implemented because they are too difficult to implement and bring less return from the investment.

Feature E, F and K are optional features that really depends on the designer and market.

This method helps professionals to make a decision correctly and communicate more efficiently. In the final step, we want to use the UML extension mechanisms: tag

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

value, constraint and stereotype to represent these kinds of information within a UML diagram.

We use the constraint extension mechanism of UML to insert two customers' need derived from the above method into the new use case diagram to increase the understandability of the elevator system.

8.2.2 UORE (usage oriented requirement engineering)

When we deal with a complex system, too rapid formalization of requirements may have negative consequences. So it is better to implement a task that encourages a complete understanding of what the customer and the end users require from the system and how they intend to use it in practice(Regnell).

Use Case Driven Analysis (UCDA), as a key contribution of the objectory method, is helpful for designers in this respect. However, UCDA still have some disadvantages for example the lack of synthesis. Therefore, we will try to employ usage oriented requirement engineering, an improvement to UCDA, to achieve an improved requirements engineering process. In our case study, we will discuss the method based on the elevator system.

8.2.2.1 Analysis phase

The analysis phase of UORE consists of two interrelated activities: identification of use cases and actors and unification of the terminology. Moreover, these two activities can be performed interactively.

In fact, the analysis of UORE resembles the objectory version of UCDA. However, there are still some aspects that make the method different (James 1999):

- Changed semantics of actors and use cases
- Identification of use case contexts
- Strict application of the single-actor view
- Explicit unification of terminology
- Structured description of uses cases.

In UORE, use cases describe the system's behavior seen by only one actor - this single-actor-view approach makes the use case concept simpler.

The following Figure 8.3 shows such use case descriptions. It describes two use cases by passengers in details.

There are nine use cases based on the previously described requirement documentation of the elevator system in our system:

- Control elevator: These scenarios include lots of details, for example open/close door, button operation....
- Request elevator: This scenario mainly makes a request from passenger after passengers select the floor button.
- Call for help: Passengers will use phone or emergence button to call for help

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

when problems exist.

•Fix elevator: The technician will fix the problems of elevator.

•Activate elevator: The technician will restart elevator system when it is down.

•Clear elevator: The cleaner will clean the elevator regularly.

•Open/Close the Doors: The elevator should be able to open and close the doors for the passengers to get in and out of the car. The functional areas of this use case should also enable the passengers to make door reversals when the doors are closing and the passenger wants to get in (or out of) the car.

•Go up/down: The elevator will often go up / down to load passengers.

•Move/Stop the Car: The main function of an elevator, detailed action will include the changing of driving speed, how to make the decision to stop, and driving directions of the car.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Actors:
Elevator operator: Operate the Elevator panel to control elevator.
Passenger: ride the elevator to specific floor.
Technician: fix the elevator' s problem
Cleaner: keep the elevator clean.
Motor: control elevator physical movement.
Door: load people.

1. Control elevator, normal case
Actor:" Elevator operator "
1. IC Invocation Conditions:
1. IC. 1 the operator has started the elevator.
1. FC Flow Conditions:
1. FC. 1 the elevator Is In good condition
1. FE Flow of Events:
1. FE. 1 The operator receive the request from passenger.
1. FE. 2 The operator press the button.
1. FE. 3 The elevator go and stop In the floor that passengers are In.
1. FE. 4 The door open and passengers enter In.
1. FE. 5 The door close.
1. FE. 6 The operator press the button.
1. FE. 7 The elevator go and stop In the floor that passengers request.
1. FE. 8 The door open and passengers get out.
1. FE. 9 The door close.
1. TC Termination conditions:
1. TC. 1 The operator wait for request from passengers.

2. Request Elevator, normal case
Actor:" Passenger "
2. IC Invocation Conditions:
2. IC. 1 The Elevator Is ready to load The Passenger.
2. FC Flow Conditions:
2. FC. 1 The Elevator Is In good condition.
2. FC. 2 The Passenger have pressed the button to make a decision.
2. FE Flow of Events:
2. FE. 1 The Elevator Is In operation
2. FE. 2 The Passenger press the hallpositionbutton to request
2. FE. 3 The Elevator stop In the floor that passengers are In.
2. FE. 4 The Door open
2. FE. 5 The Passenger enter and choose the floor to process
2. FE. 6 The Door close and The Elevator process the request and reach the request floor.
2. FE. 7 The Door open and the Passenger get out of elevator
2. TC Termination conditions:
2. TC. 1 The Elevator Is ready to process the request

3. Call for help, normal case
Actor:" Passenger"
3. IC Invocation Conditions:
3. IC. 1 The elevator Is ready to process request.
3. FC Flow Conditions:
3. FC. 1 The communication Instruction of elevator work well.
3. FC. 2 The elevator got some problems.
3. FE Flow of Events:
3. FE. 1 The elevator got some problems.
3. FE. 2 The Passenger press the emergency button or make a phone call to call for help.
3. TC Termination conditions:
3. TC. 1 The Passenger got response.

4. Fix elevator, normal case
Actor:" Technician "
4. IC Invocation Conditions:
4. IC. 1 The elevator Is not In use.
4. FC Flow Conditions:
4. FC. 1 The technician has figured out what Is problem.
4. FE Flow of Events:
4. FE. 1 The technician enter elevator.
4. FE. 2 The technician solve the problem.
4. FE. 3 The technician restart the elevator.
4. TC Termination conditions:
4. TC. 1 The elevator Is In use.

Figure8.3: Use case description

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Actors:
Elevator operator: Operate the Elevator panel to control elevator.
Passenger: ride the elevator to specific floor.
Technician: fix the elevator' s problem
Cleaner: keep the elevator clean.
Motor: control elevator physical movement.
Door: load people.

5. Activate elevator, normal case
Actor:" Technician "
5. IC Invocation Conditions:
5. IC. 1 The elevator Is down
5. FC Flow Conditions:
5. FC. 1 The technician has solved the problem.
5. FE Flow of Events:
5. FE. 1 The technician open the door mamually.
5. FE. 2 The technician activiate the elevator by key or other tools.
5. FE. 3 The technican close the door.
5. TC Termination conditions:
5. TC. 1 The elevator Is In use.

6. Clean elevator, normal case
Actor:" Cleaner "
6. IC Invocation Conditions:
6. IC. 1 The elevator Is not In use.
6. FC Flow Conditions:
6. FC. 1 The cleaner collect enough tools to clean the elevator.
6. FE Flow of Events:
6. FE. 1 The cleaner open the door.
6. FE. 2 The cleaner do the cleaning.
6. FE. 3 The cleaner close the door.
6. TC Termination conditions:
6. TC. 1 The elevator Is In use.

7. Open/Close Door, normal case
Actor:" Door "
7. IC Invocation Conditions:
7. IC. 1 The elevator Is In operation.
7. FC Flow Conditions:
7. FC. 1 The elevator Is In good condition.
7. FE Flow of Events:
7. FE. 1 The passenger make a request.
7. FE. 2 The elevator stop at the floor requested .
7. FE. 3 The door open.
7. FE. 4 The passenger enter In.
7. FE. 5 the door close.
7. TC Termination conditions:
7. TC. 1 The Elevator Is ready to process the request

8. Go up/down, normal case
Actor:" Motor "
8. IC Invocation Conditions:
8. IC. 1 The elevator Is In good condition.
8. FC Flow Conditions:
8. FC. 1 The operator Is on duty.
8. FE Flow of Events:
8. FE. 1 The passenger make a request.
8. FE. 2 The operator process the request
8. FE. 3 The elevator (motor) go up/down
8. FE. 4 The elevator stop at the floor request and load the passengers
8. TC Termination conditions:
8. TC. 1The elevator Is ready to process request.

9. Stop elevator, normal case
Actor:" Motor "
9. IC Invocation Conditions:
9. IC. 1 The elevator Is In good condition.
9. FC Flow Conditions:
9. EC. 1 The operator Is on duty.
9. FE Flow of Events:
9. FE. 1 The operator press button or Insert the key to request a stop
9. FE. 2 the elevator (motor) stop.
9. TC Termination conditions:
9. TC. 1 The elevator Is ready to process request.

Figure 8.4: Use case description

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Now we have gotten the actors and use case description, so we will begin the synthesis phase in the following parts.

8.2.2.2 Synthesis phase

This method in the synthesis phase consists of 3 activities: formalization of use cases, integration of use cases and verification (Regnell).

1.formalization of use cases

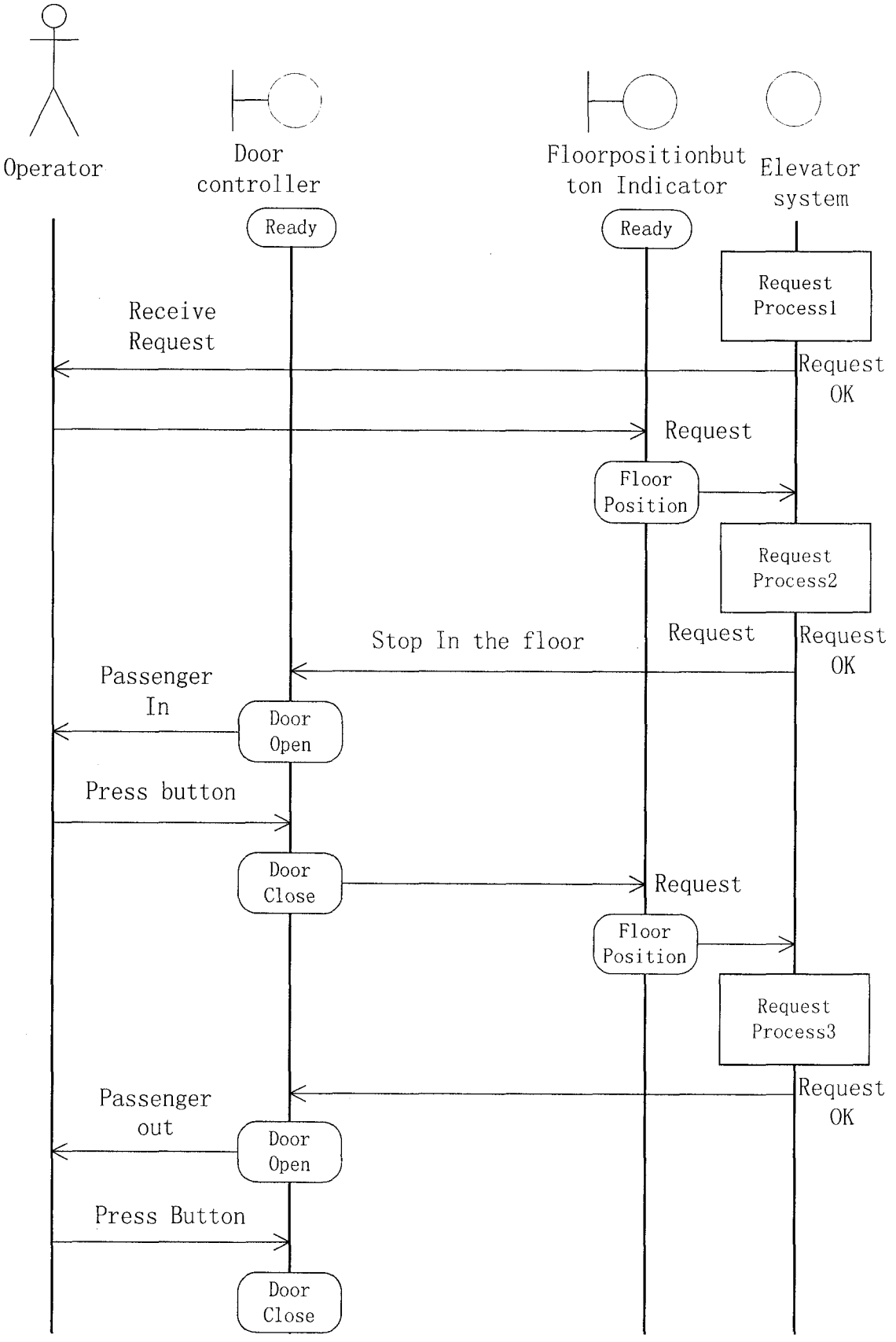
The purpose of the formalization of use cases is to produce a formal use case specification (UCS) that represents in a formal, graphic language, using message sequence charts, for each use case identified in the analysis phase. After identifying all abstract data and atomic operations, we transform the flow of events of every use case into an UCS that models the relations between stimuli/responses/states and atomic operations.

In the following diagram, we illustrate the notation of UCS for two use case specifications: request elevator and call for help by our elevator system (See Figure 8.5 and Figure 8.6).

The left-most time axis of the following diagram shows the specified actor. The right-most time axis represents the system. We have the different AIO's (the entities that form the nature of user-system communication) involved in this use case between the actor and system. The AIO states are drawn as diamonds on the AIO time axis, and the atomic operations are drawn as boxes on the system's time axis.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Control elevator - normal case



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 8.5: Use Case: Control elevator Specification

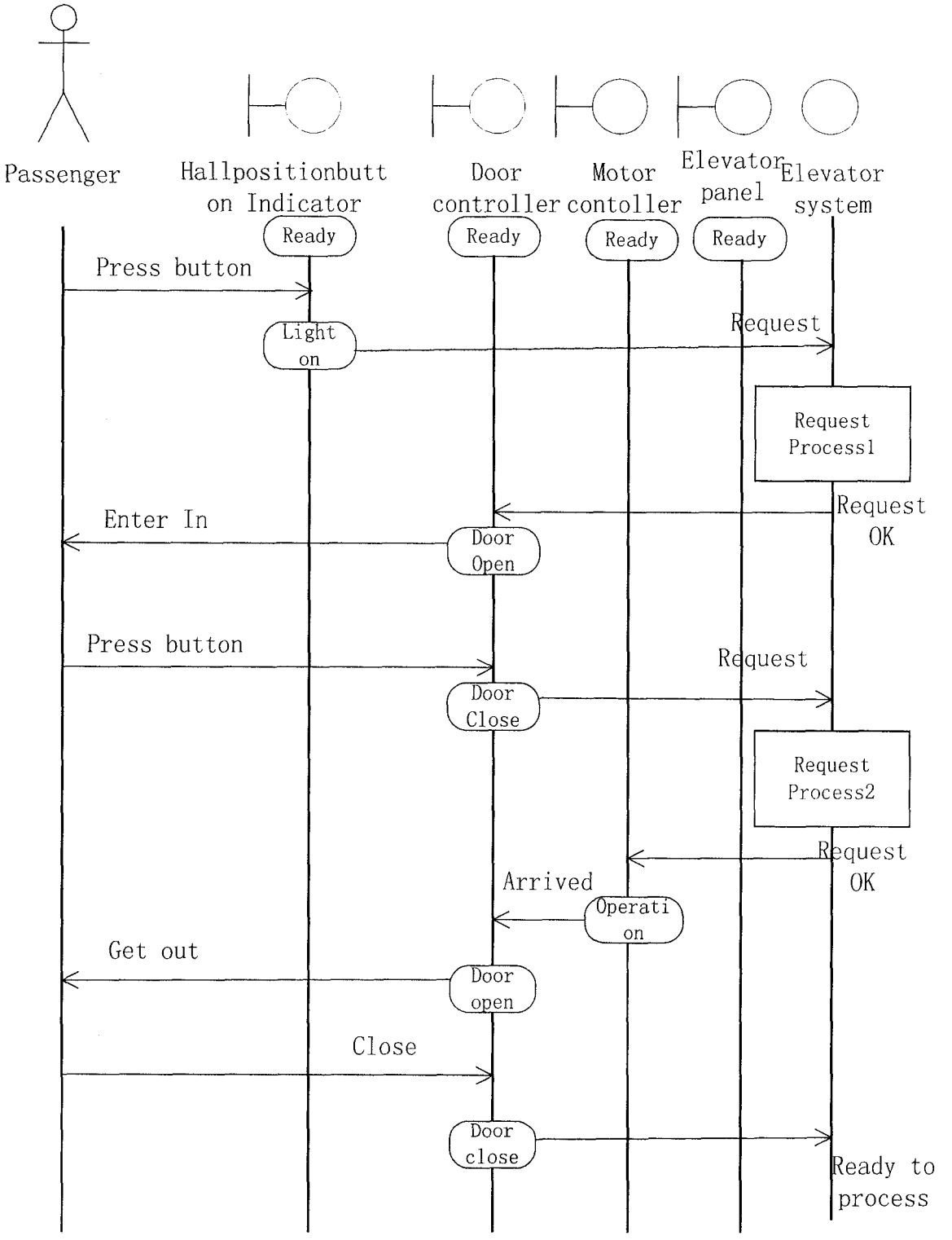


Figure 8.6: Use Case: Request elevator Specification

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Call for help: normal case

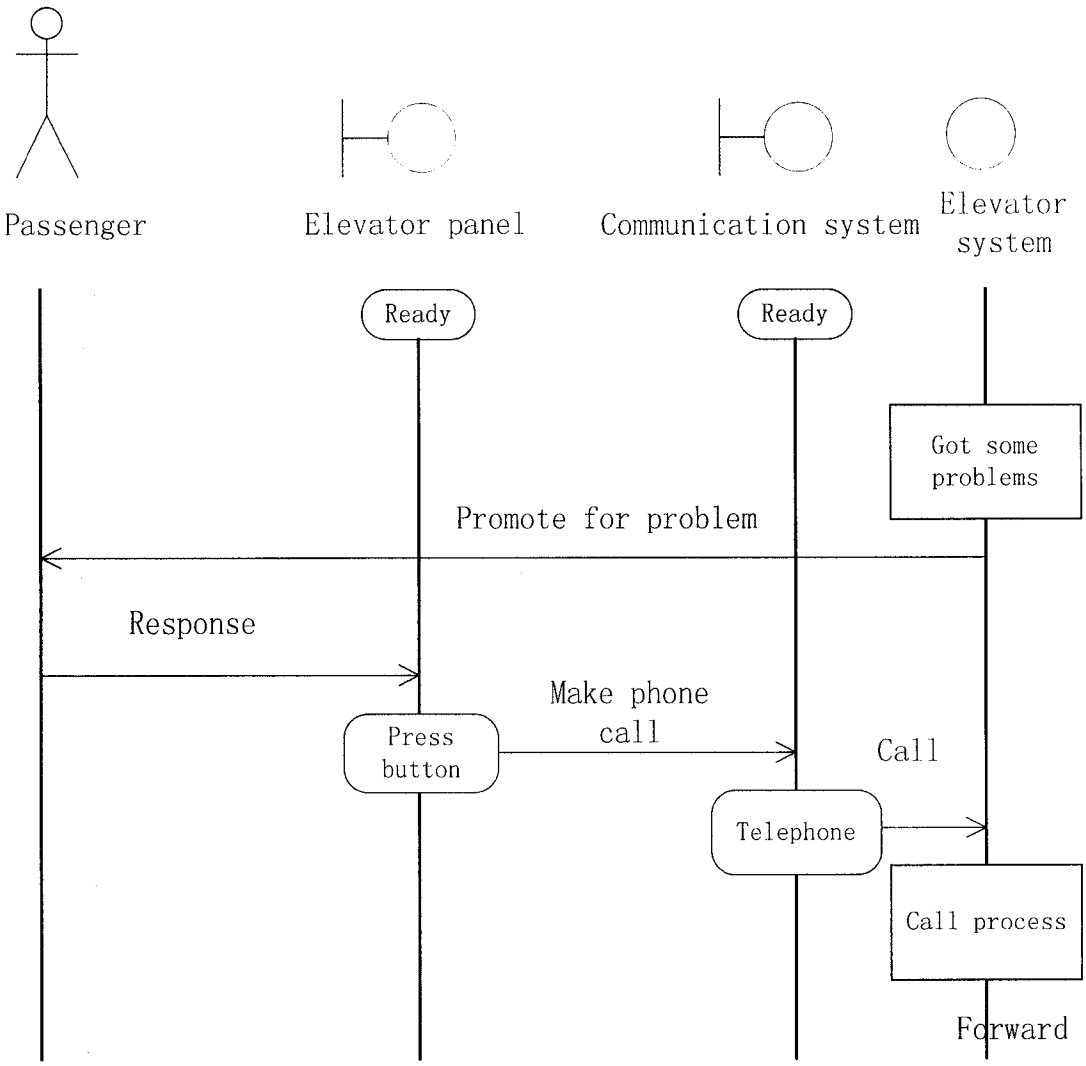
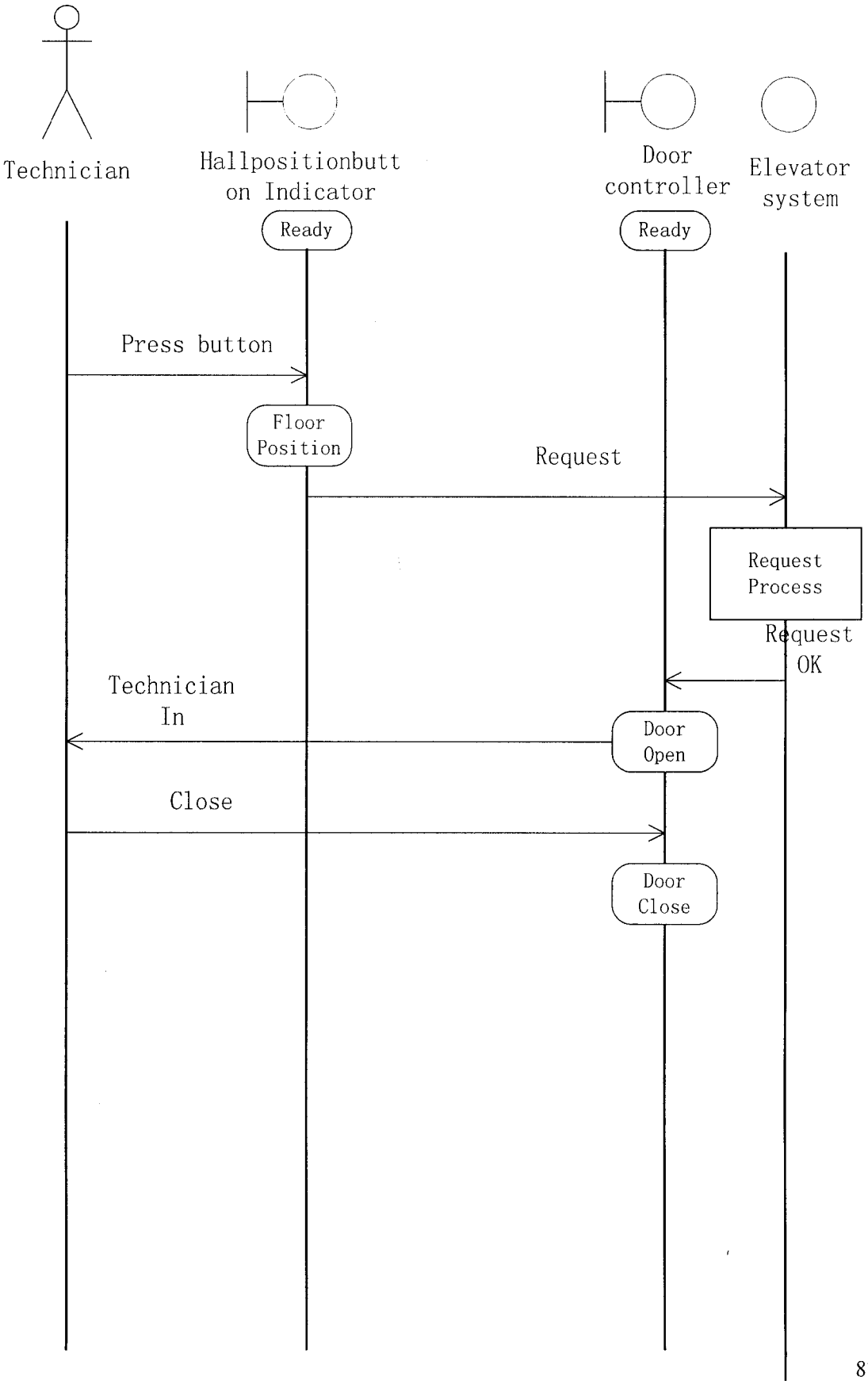


Figure 8.7: Use case: call for help specification

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fix elevator - normal case



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 8.8: Use Case: Fix Elevator specification

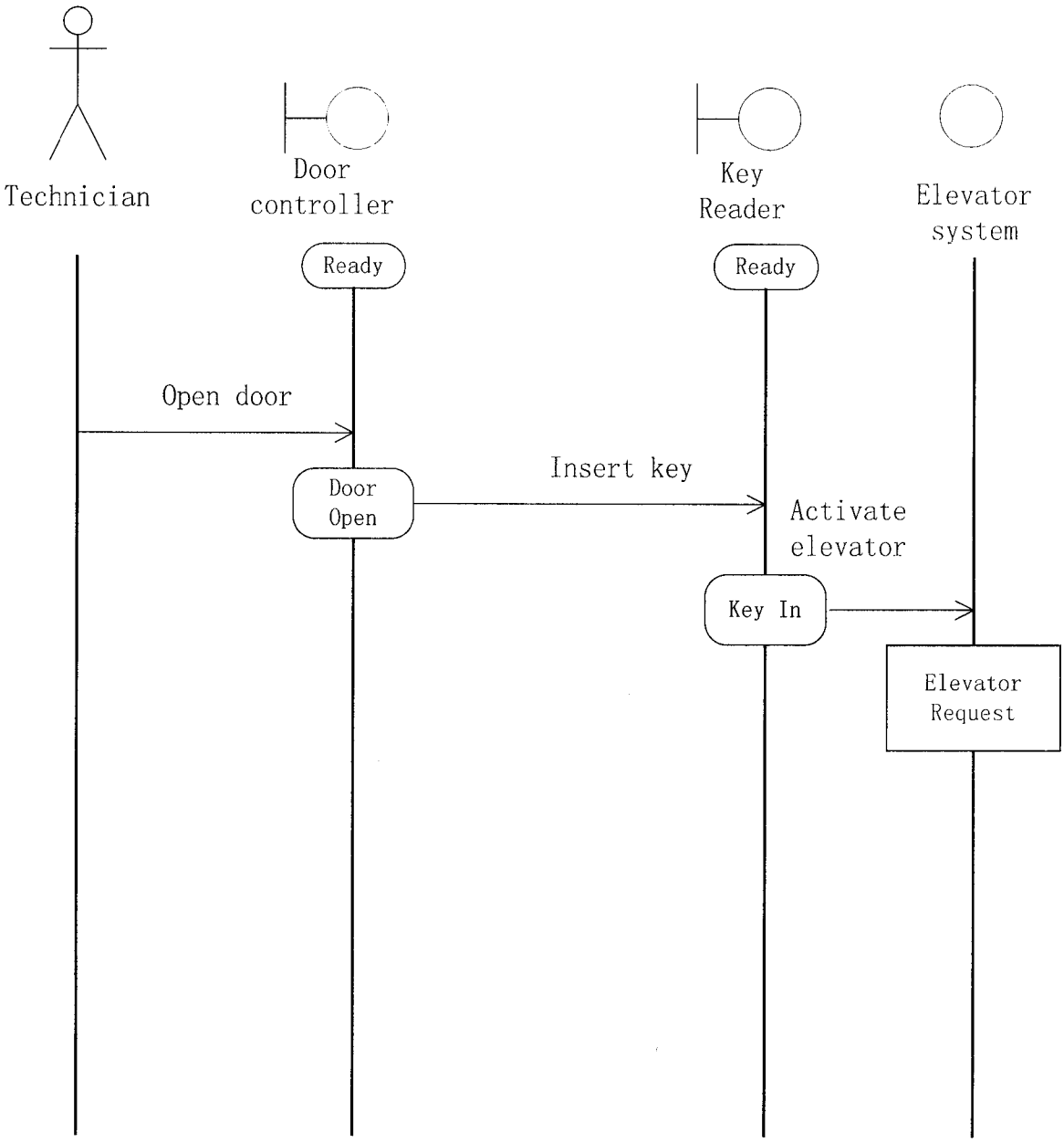


Figure 8.9: Use Case: Activate Elevator specification

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

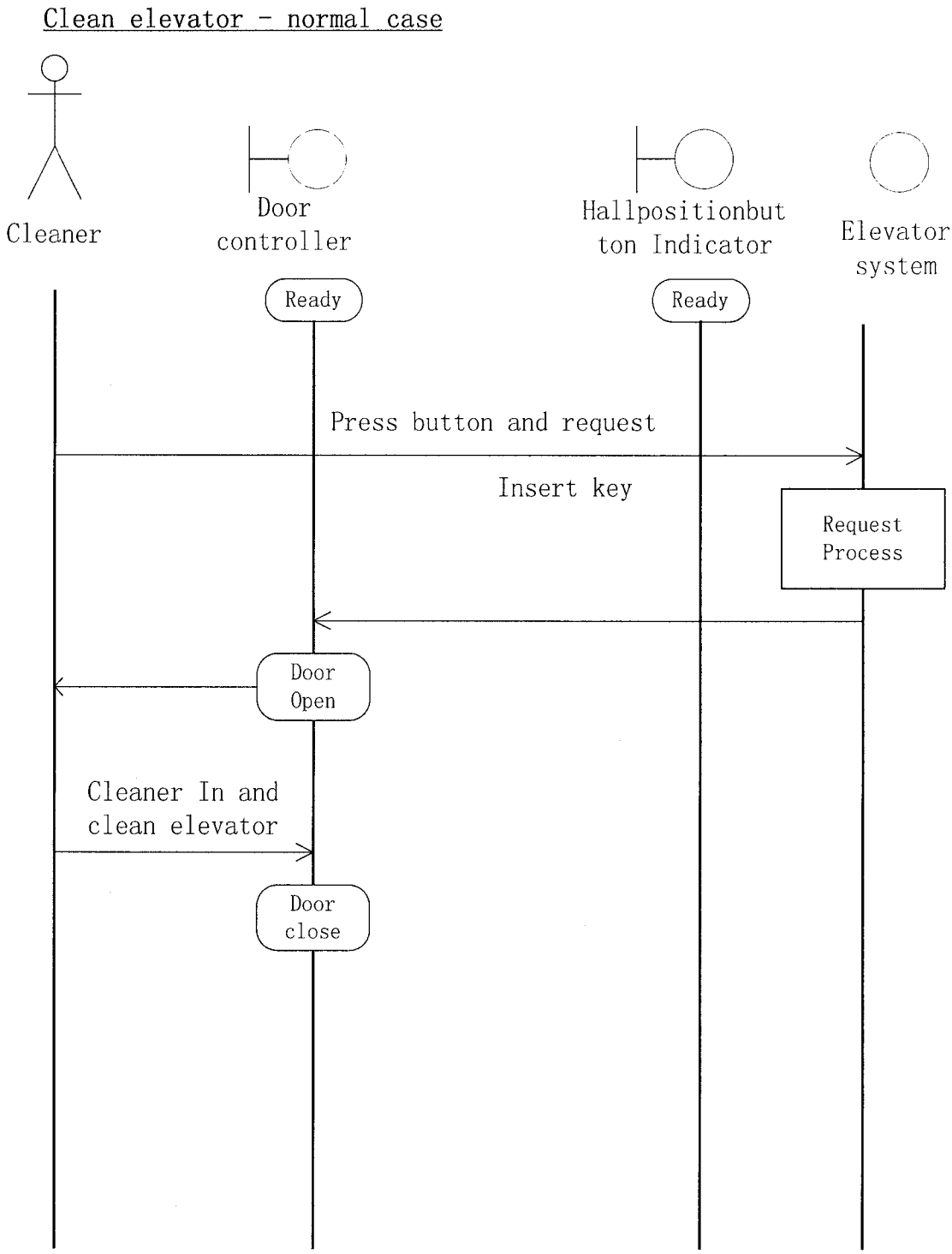


Figure 8.10: Use Case: Clean Elevator specification

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

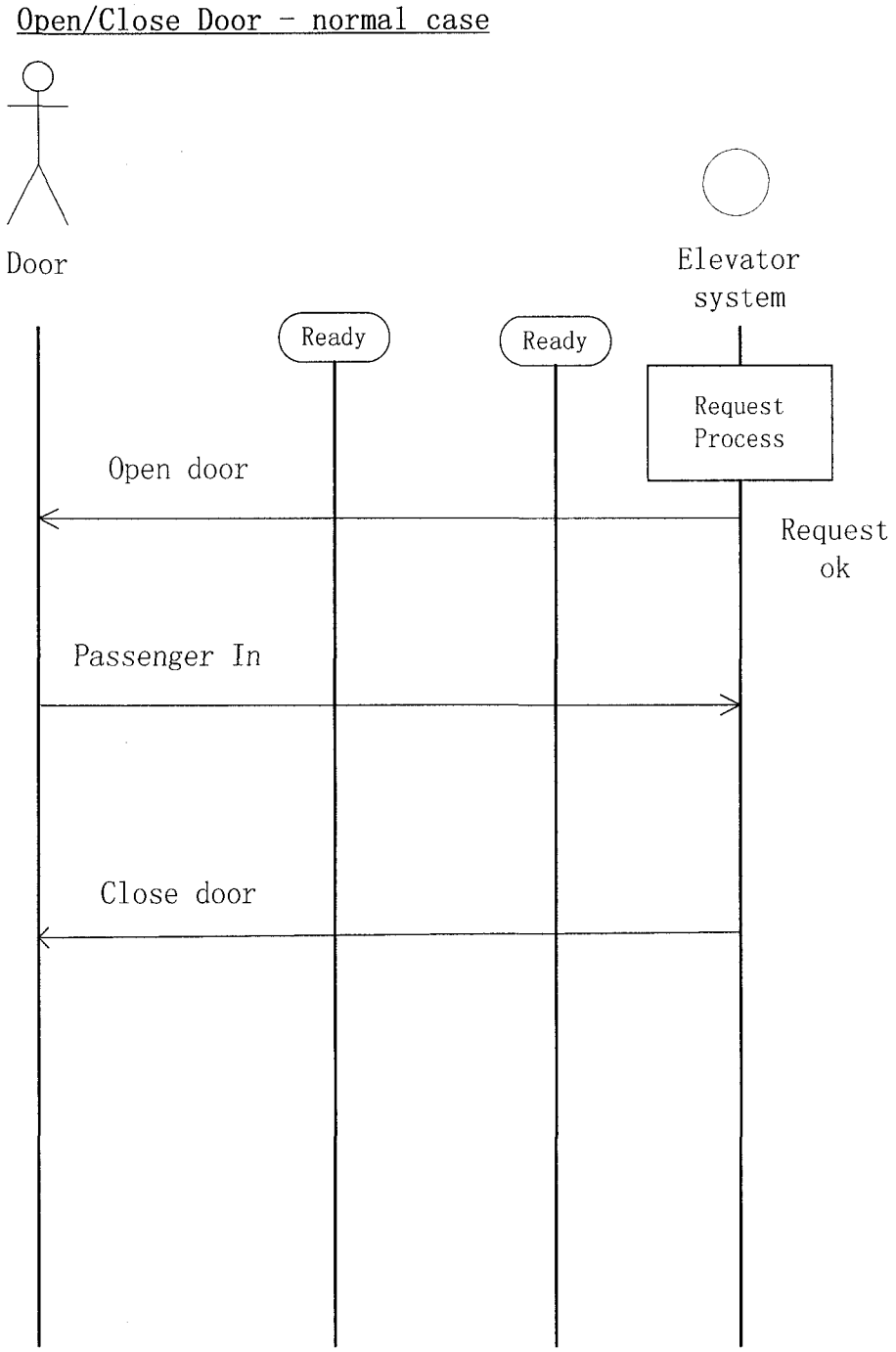


Figure 8.11: Use Case: Open/Close door specification

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

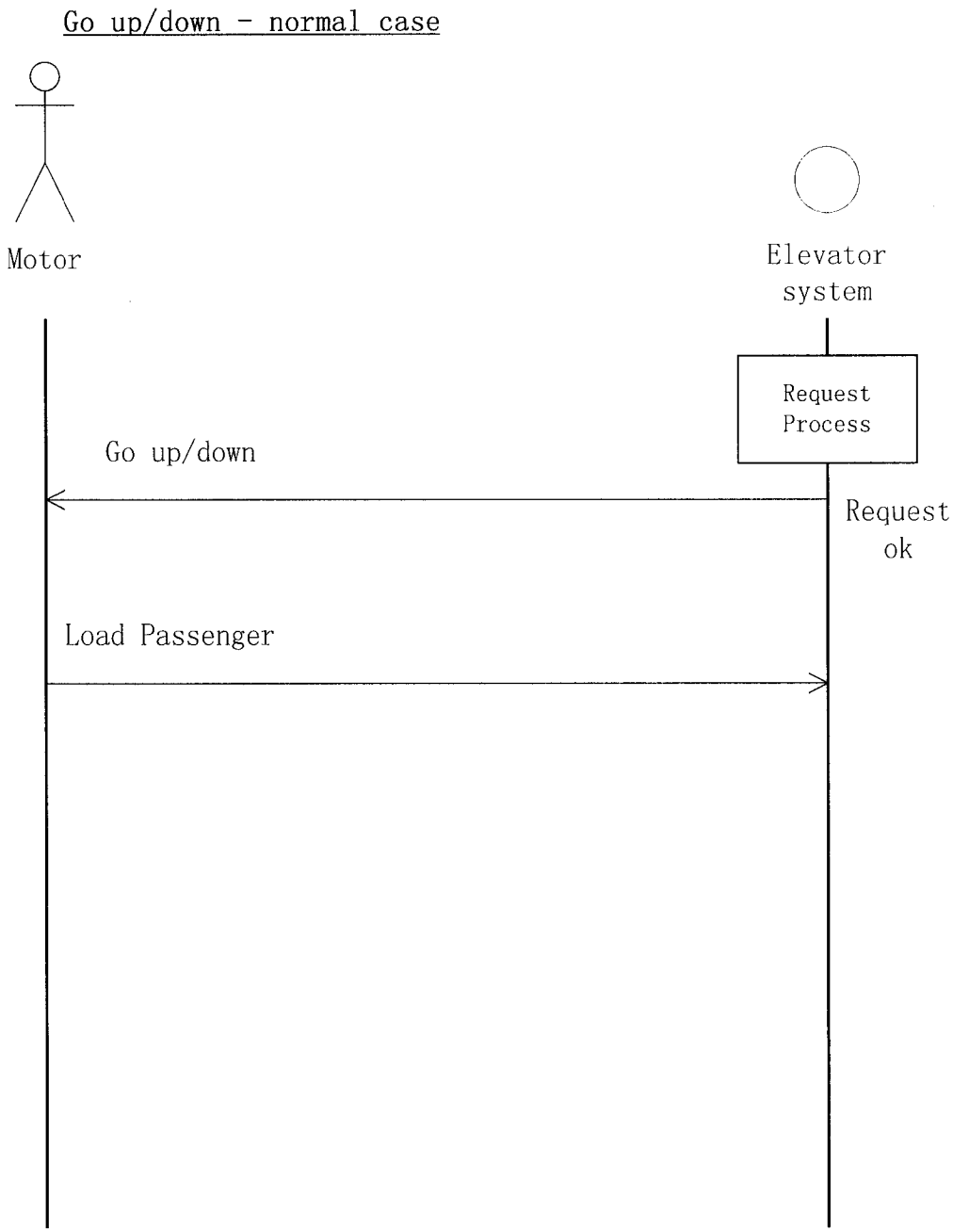


Figure 8.12: Use Case: Go up/down specification

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

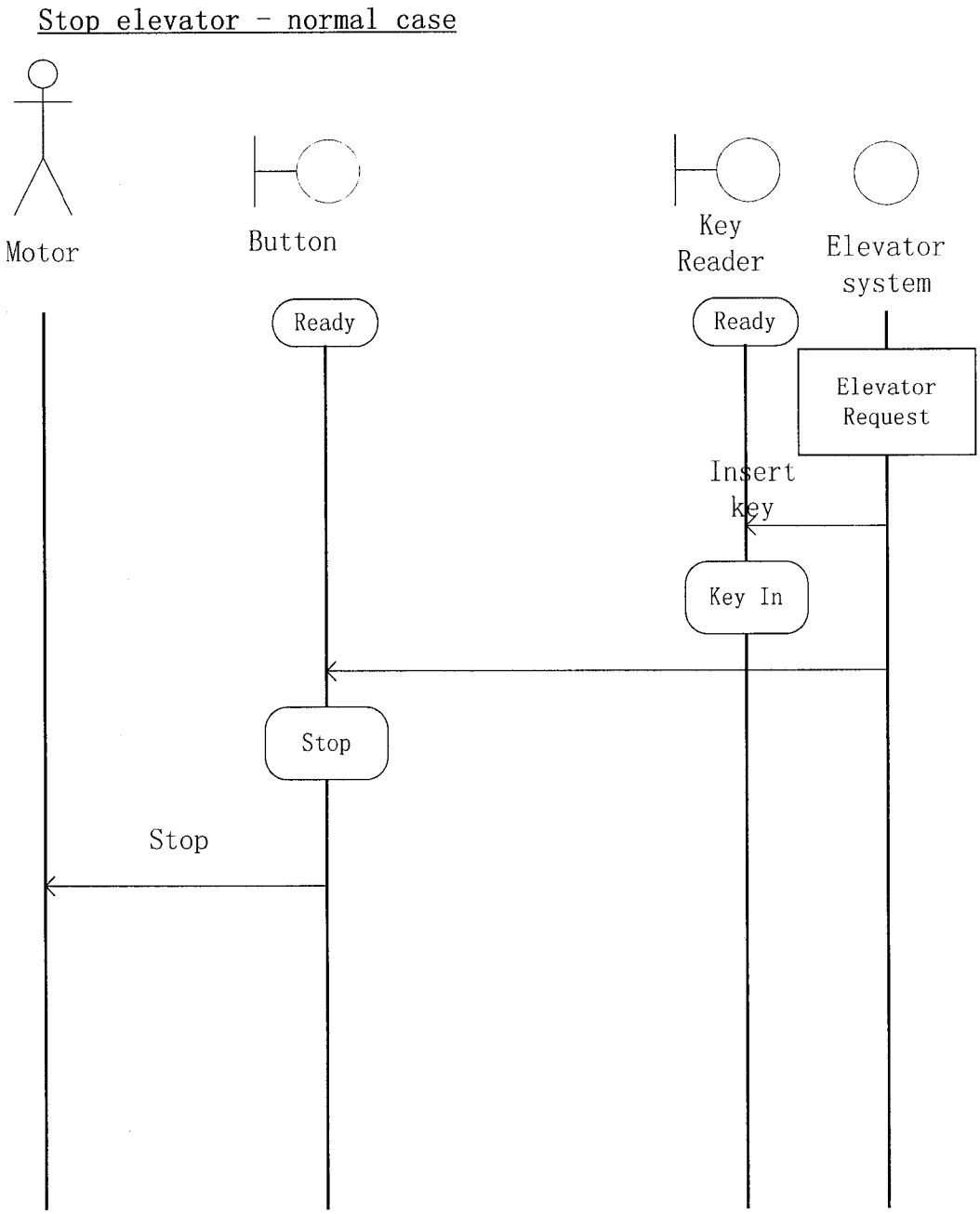


Figure 8.13: Use Case: Stop elevator specification

2. Integration of use cases

These step aims at merging different use case specifications and producing a Synthesized Usage Model. Firstly, we create abstract usage scenarios after identifying the user and system actions. We accomplish this by transforming every UCS into an Abstract Usage Scenario (AUS), drawn as a sequence of user actions (bubbles) and system actions (boxes) interconnected with transitions (arrows) that represent the resulting messages of each action.

The main purpose of creating AUS's is to make the synthesis feasible.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 8.14 describes the AUS for use case: control elevator by the elevator operator.
Figure 8.15 and 8.16 separately describe the AUS for use cases: request elevator and call for help by passengers.
Figure 8.17 and 8.18 separately describe the AUS for use cases: fix elevator and activate elevator by technician.
Figure 8.19 describes the AUS for use case: clean elevator by cleaner.
Figure 8.20 describes the AUS for use case: open/close door by door.
Figure 8.21 and 8.22 separately describe the AUS for use cases: go up/down and stop elevator by motor.
So in the next step, we will synthesize the usage model.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

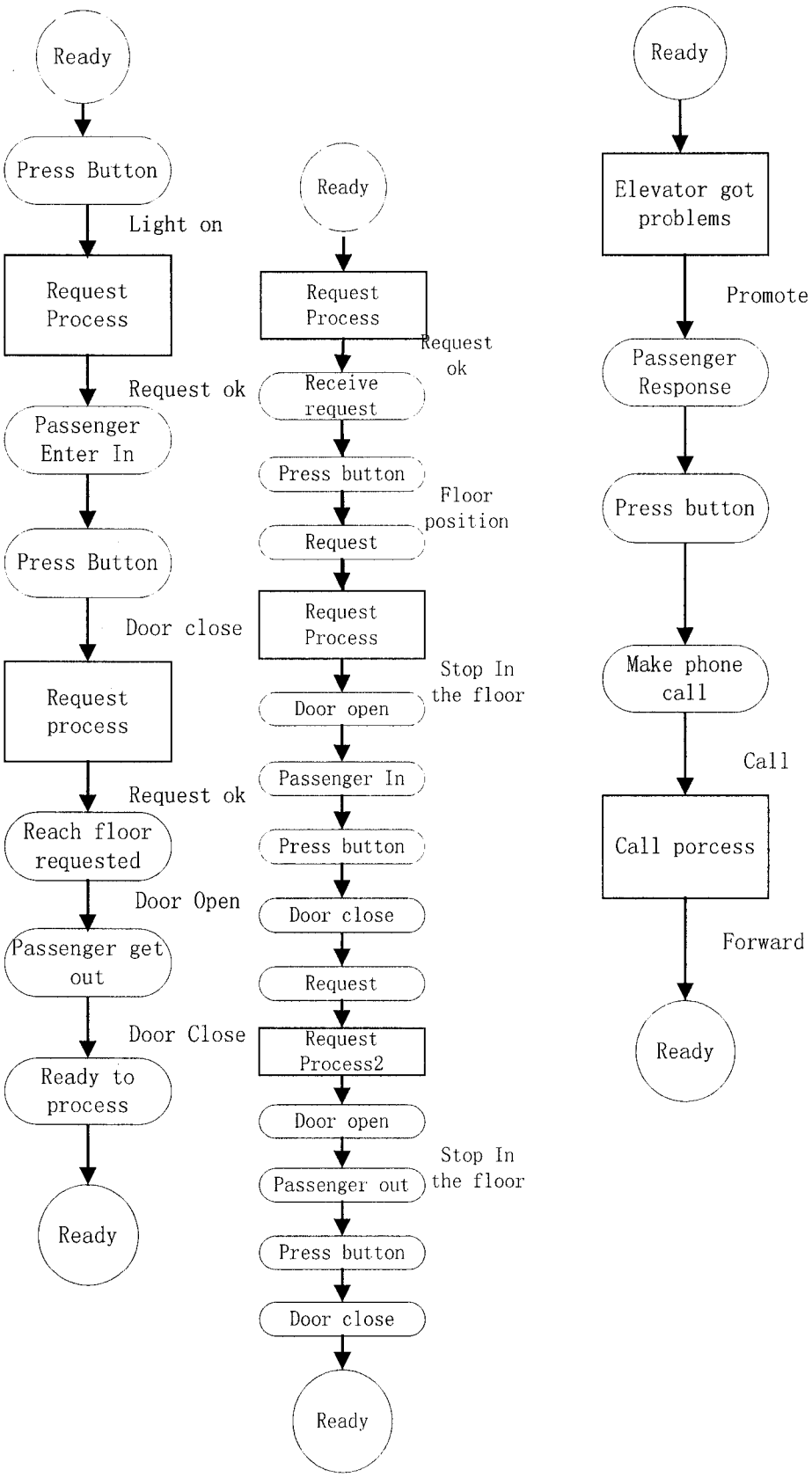
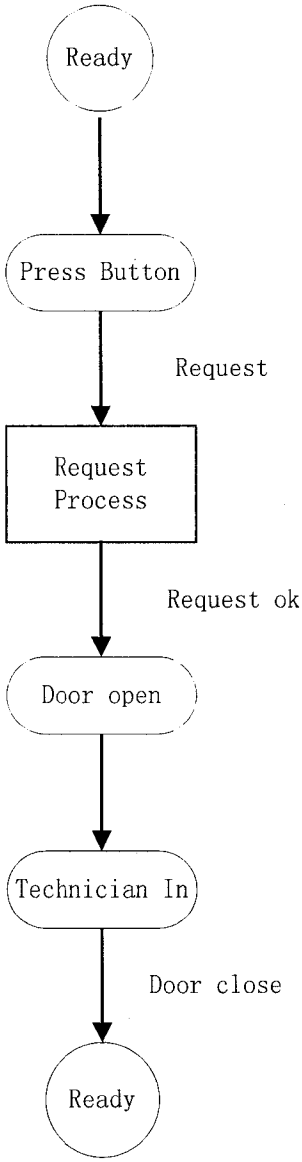


Figure 8.14: Abstract Usage Scenario: Control elevator

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 8.15: Abstract Usage Scenario: Request elevator

Figure 8.16: Abstract Usage Scenario: Call for help



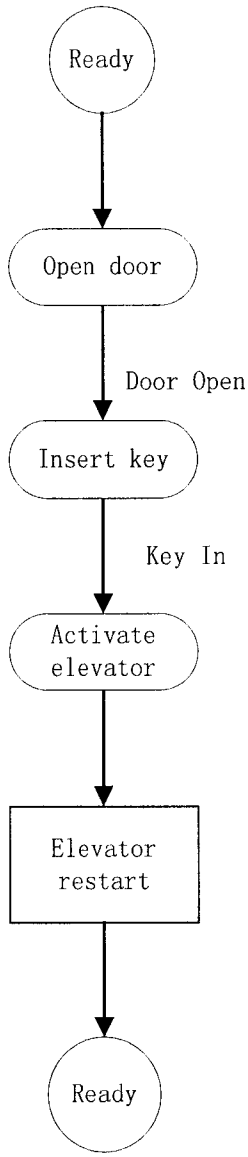


Figure 8.17: Abstract Usage Scenario: Fix elevator

Figure 8.18: Abstract Usage Scenario: Activate elevator

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

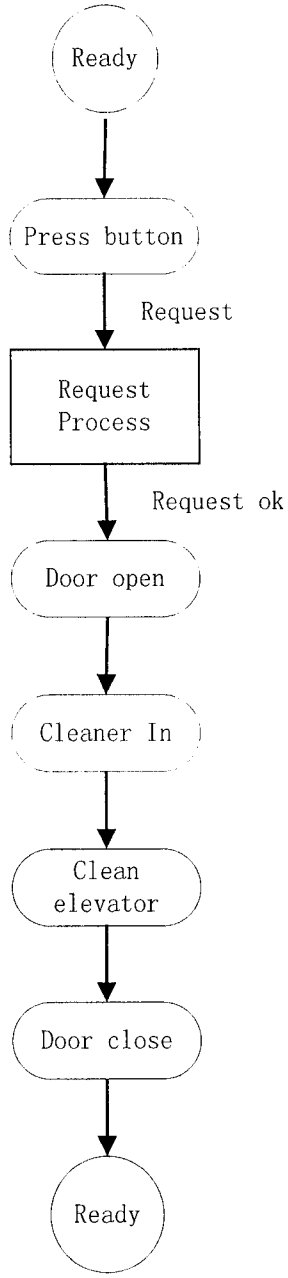


Figure 8.19: Abstract Usage Scenario: Clean elevator

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

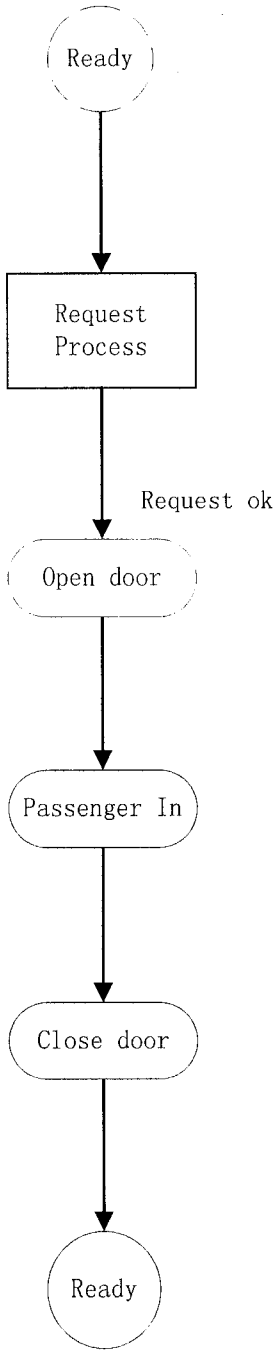
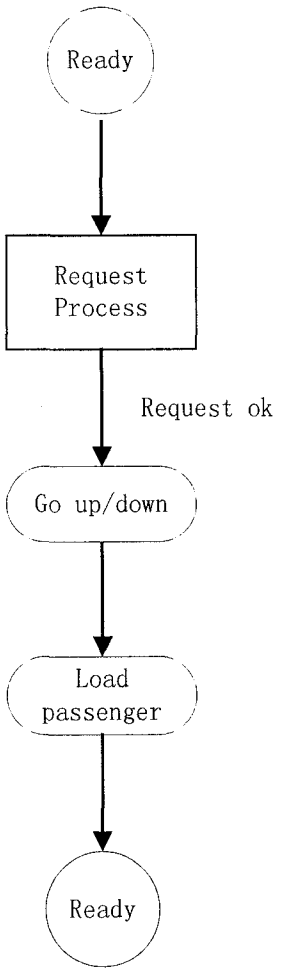


Figure 8.20: Abstract Usage Scenario: Open/Close door

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



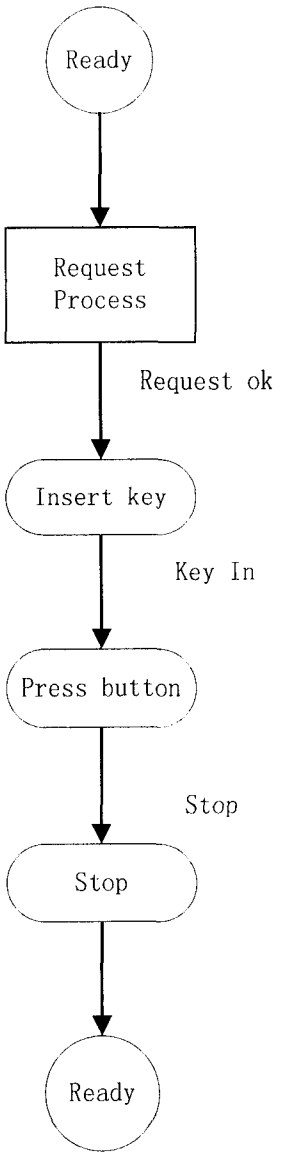


Figure 8.21: Abstract Usage Scenario: Go up/down

Figure 8.22: Abstract Usage Scenario: Stop elevator

The Synthesized Usage Model (SUM) consists of one usage view per actor. A usage view is synthesized from all Abstract Usage Scenarios produced for one specific actor. A usage view is created by finding similar parts of Abstract Usage Scenarios and merging them. The result is a directed graph with three types of nodes: user actions, system actions, and labels. These nodes have the same meaning as in the Abstract Usage Scenarios. Labels are used to maintain tractability between usage views and AUS's (James 1999).

So Figure 8.23 is the usage view for "Passenger", it is synthesized from the two Abstract Usage Scenarios produced for the specific actor: "Passenger".

Figure 8.24 is the usage view for "Operator".

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 8.25 is the usage view for "Technician"
Figure 8.26 is the usage view for "Cleaner"
Figure 8.27 is the usage view for "Motor"
Figure 8.28 is the usage view for "Door"
3. Verification

The final step of this method is to verify the activity in order to obtain a consistent and complete SUM. Firstly, what we should is to check if the UCS is a correct transformation of the informal use case description. Then, we must make sure that the SUM completely covers every UCS(Regnell).
We just take two use cases of the elevator system as an example to explain the use of the UORE. The Synthesized Usage Model created by UORE could be used as a reference model for the whole system development process. Therefore, the method also can be used to an improvement to replace UCDA in the Rational Unified Process.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

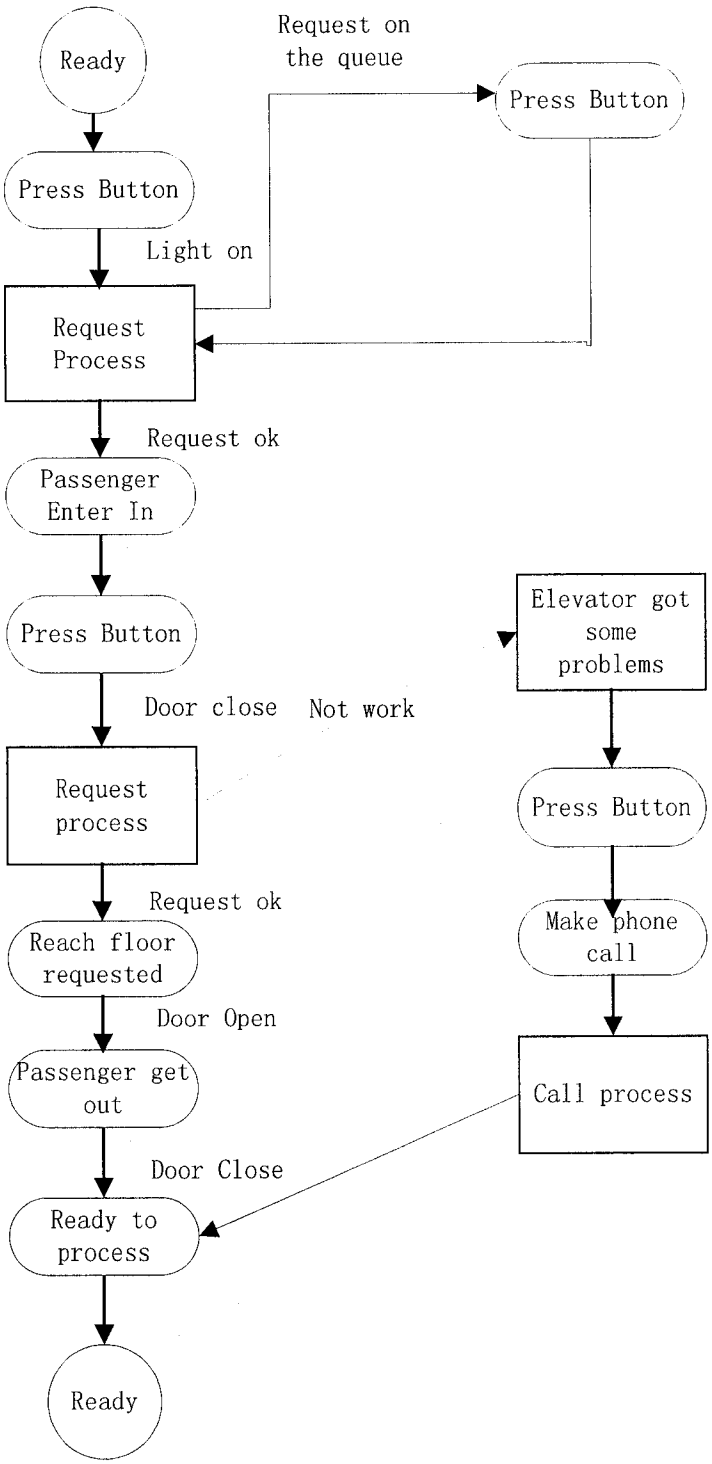


Figure 8.23: The usage view for "passenger"

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

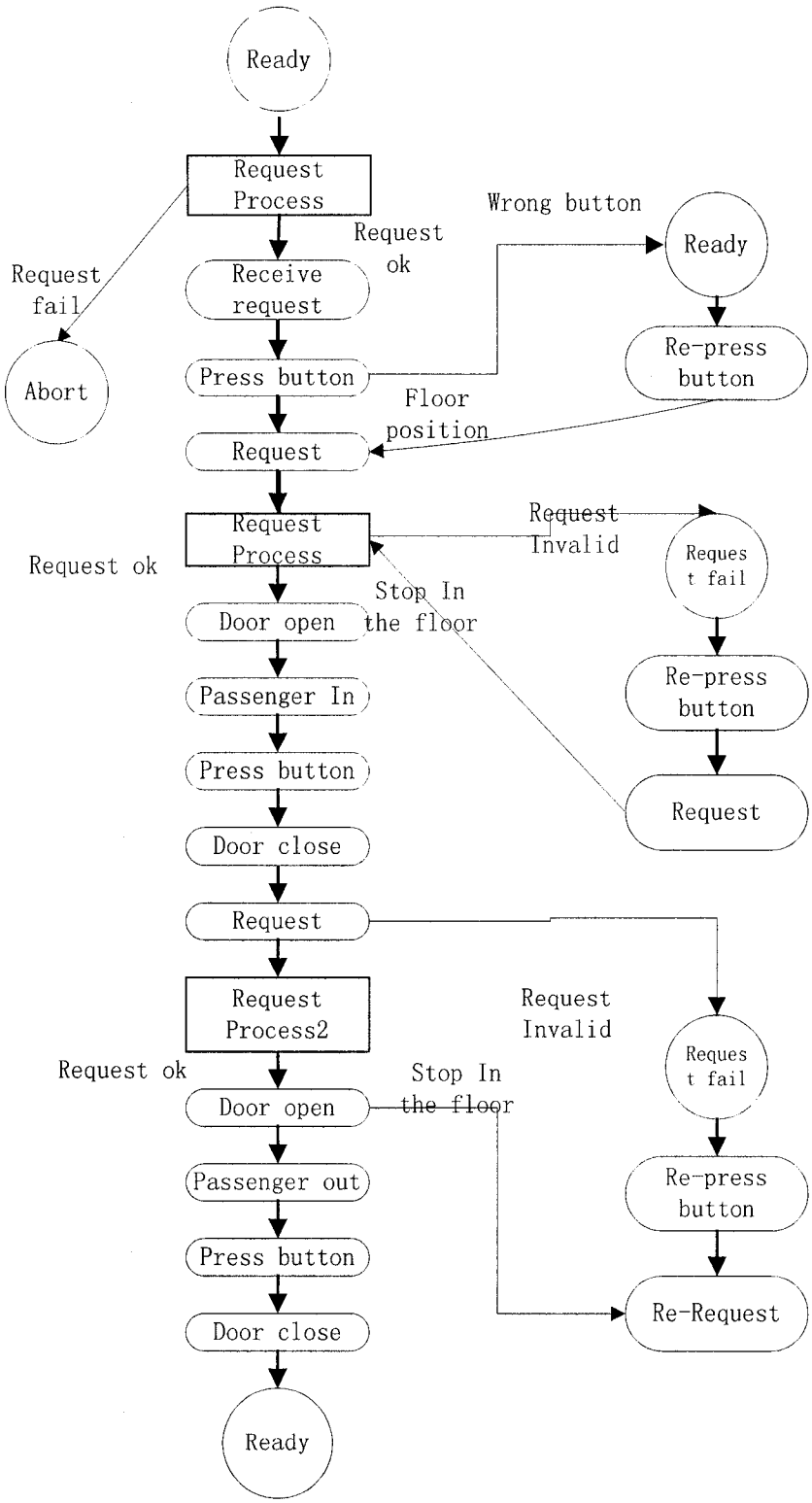


Figure 8.24: The usage view for "Operator"

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

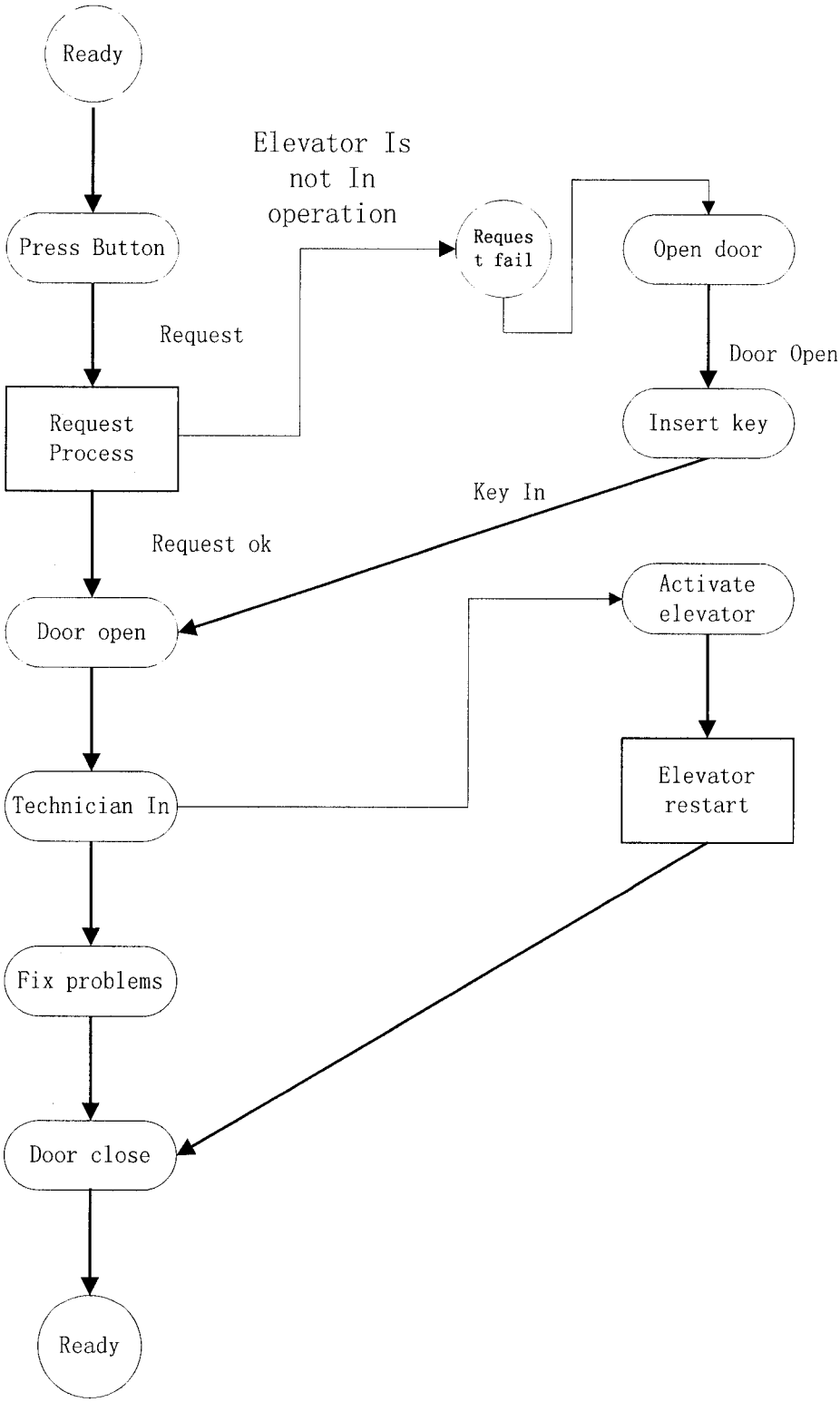


Figure 8.25: The usage view for "Technician"

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

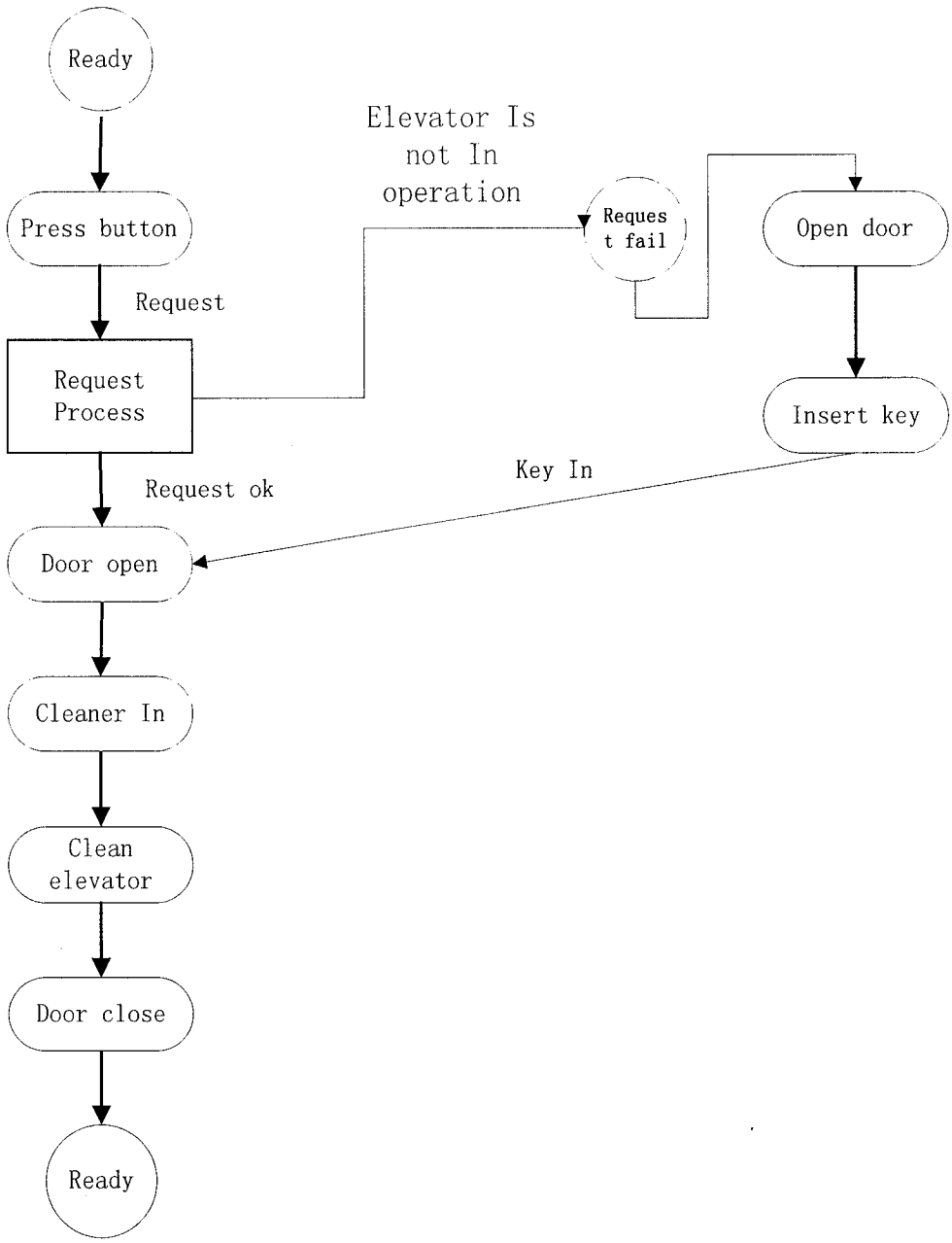


Figure 8.26: The usage view for "Cleaner"

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

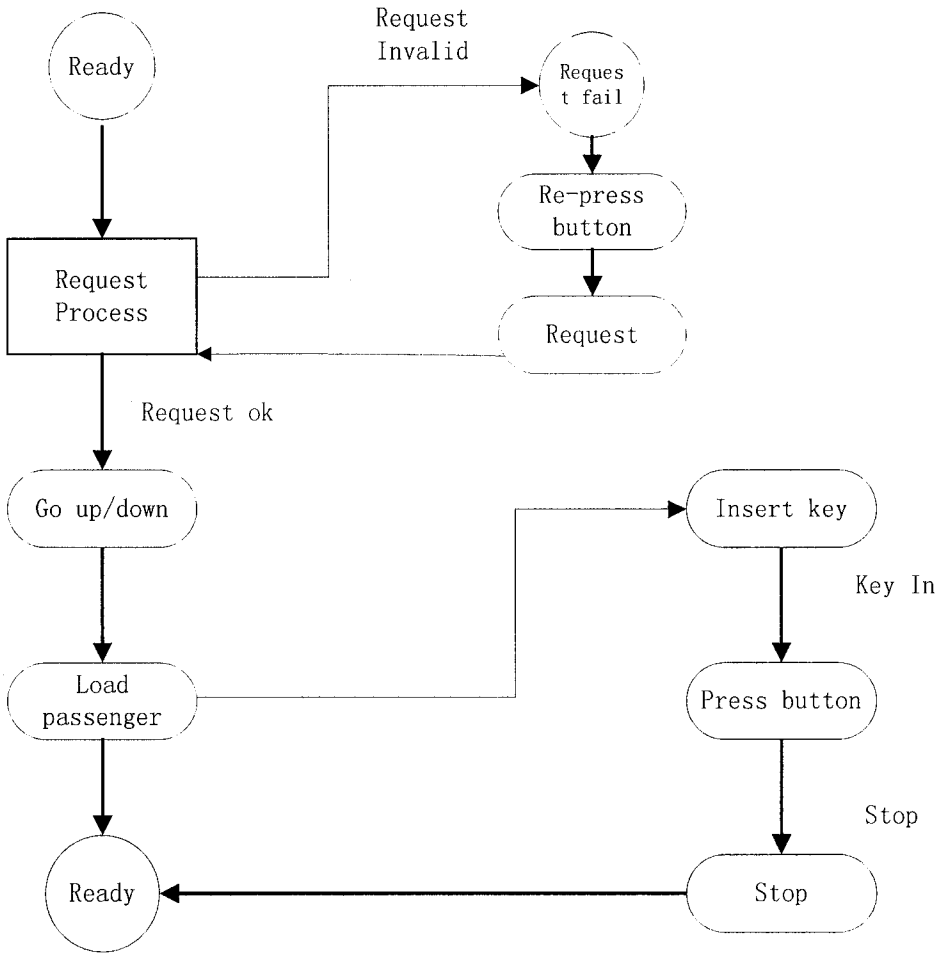


Figure 8.27: The usage view for "Motor"

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

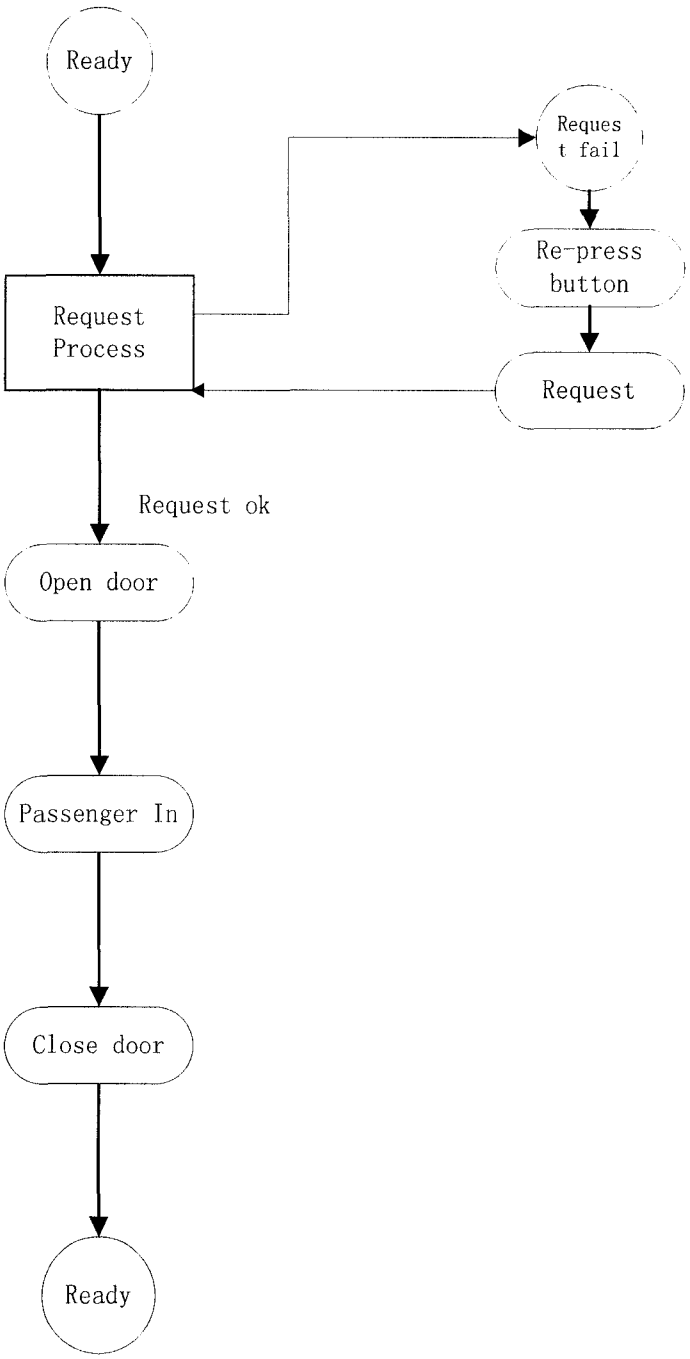


Figure 8.28: The usage view for "Door"

8.2.2.3 UORE application

All the models (Use Case Description, Use Case Specification, Abstract Usage Scenario, Synthesized Usage Model) produced by the method UORE (Usage oriented requirement engineer) are necessary parts to implement this method. However, the SUM (Synthesized Usage Model) is the most important output of this method because it captures both function requirements and system usage aspects in a comprehensive

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

manner. It consists of one usage view per actor. A usage view is synthesized from all Abstract Usage Scenarios produced for one specific actor. A usage view is created by finding similar parts of abstract Usage Scenarios and merging them.
The UORE can be applied in the following aspects related to UML:

1.The main drawback of the UCDA (use case driven analysis) is the lack of synthesis. The Use Case Model that we get from UCDA is just a loose collection of use cases. However, the method UORE can solve this problem.
The two views complement each other nicely: use cases provide the informal map of interactions between the system and actors, whereas UORE precisely describe a particular atomic system action, called a system operation. So the UORE could be supplied with the use case model. Furthermore, we can use UML's Object Constraint Language to apply this information from UORE to a UML class model. In our new use case model, we can capture more information of use cases on usage views such as use case may overlap, occur simultaneously and influence each other.

a. We add the SUM for each actor into UML directly so that SUM might be a supply to use cases. Figures 23-28 are SUMs of actors and they can be a supply to use cases.

b. Represent information into UML diagram by constraint.

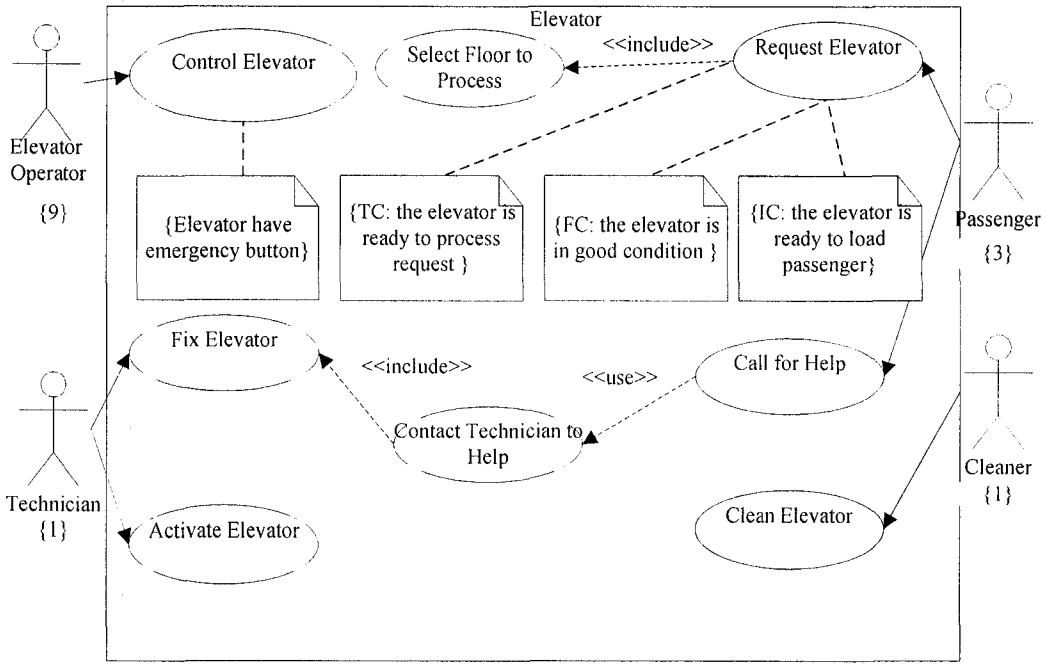


Figure 8.29 New use case diagram

In the new use case diagram above, we represent the invocation conditions, flow conditions and termination conditions of passengers into the diagram in order to describe the use case in a concise way. The same conditions for other actors can also be added into the appropriate diagram.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.Although use cases are perfect materials for creating test cases, the UCM resulting from UCDA cannot be used for automatic generation of test cases. This limits its applicability as a reference model for validation and verification. However, the possibility of the automatic generation of test cases is one of the most important properties of the SUM. So we can adopt UCM to implement this function. We won't analyze this application in this thesis because we are concentrating on the phase of requirement analysis.

3.In UCDA, one physical use can appear as different actors in a single use case so that it causes a lot of confusion. However, UORE adopts a one-actor-view method in order to solve this problem. So in this thesis, all the UML diagrams and notations that we have produced are based on the specifications and graphs of UORE.

4.A specific use case cannot occur in every situation. What we need for each use case is a specification of the context in which it can be triggered and successfully accomplished. This issue is not addressed by UCDA. UORE can provide this information such as invocation and termination contexts so that we can integrate them with UML diagrams.
The following diagram is a class diagram in which the invocation and termination have been integrated. The "IC" in the diagram means invocation conditions; the "TC" means termination condition.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

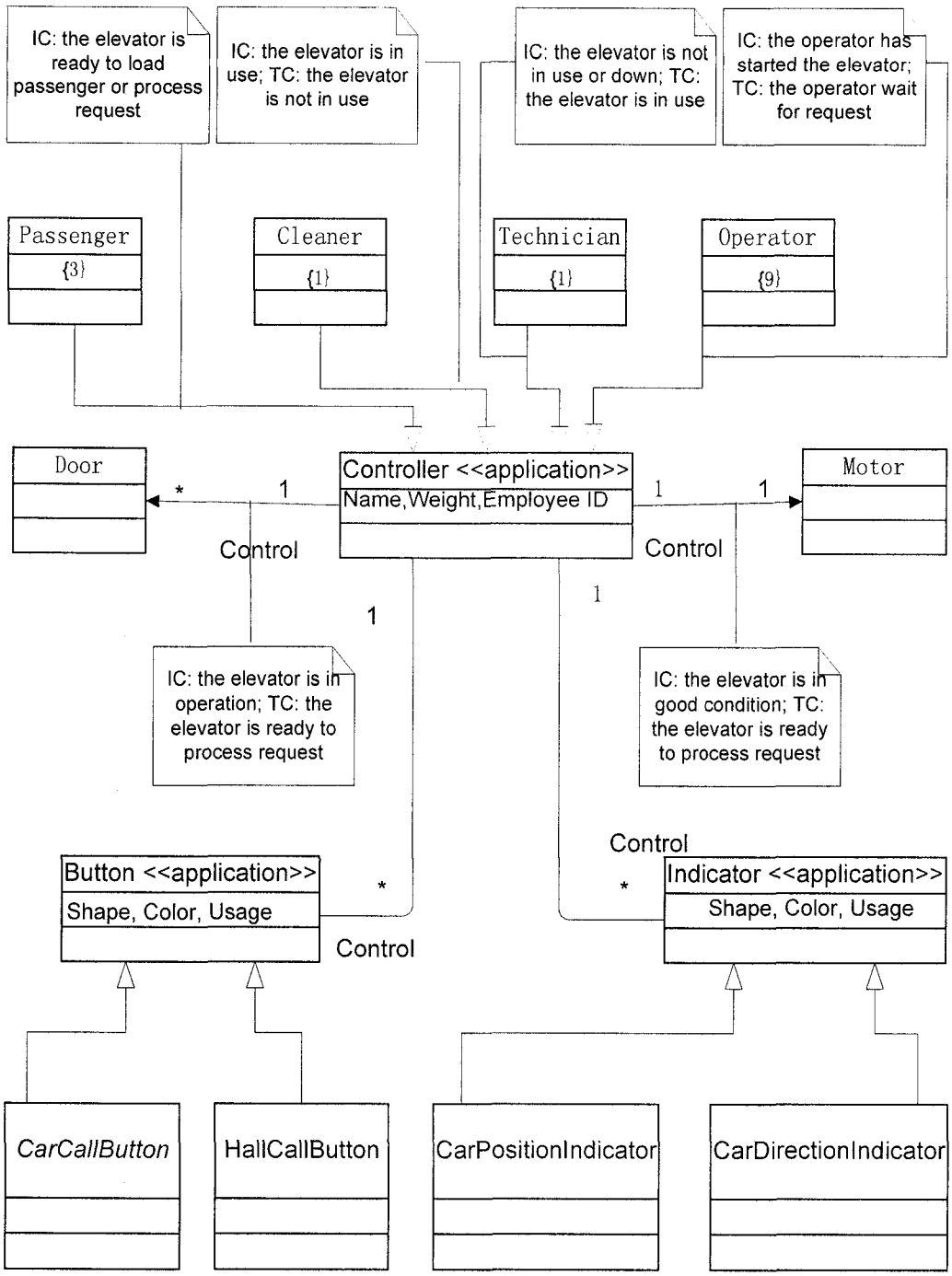


Figure 8.30 New Class diagram

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

8.3 Business modeling workflow

8.3.1Context model

The context diagram is a high level diagram used to describe the system functions and relationships to external entities.

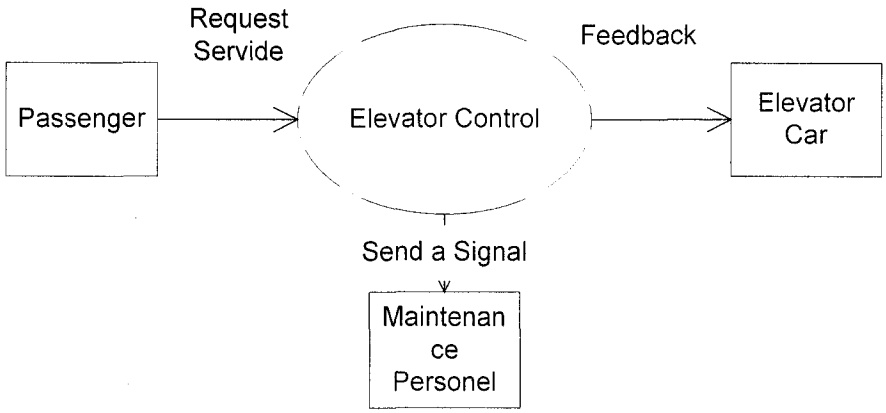


Figure 8.31 Context Diagram

Context document

That diagram is a context diagram of an elevator system. It focuses on describing passengers, elevator cars, maintenance personnel and their relationships. Meanwhile, it represents an overall understanding of the domain.

8.3.2 High-level requirement model (use case model)

Use Case Diagram: According to the requirement document, Users X Actor Role and Actor Role X Use Case Matrix mention above, we can extend the use case diagram of elevator systems showed in Figure 8.32:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

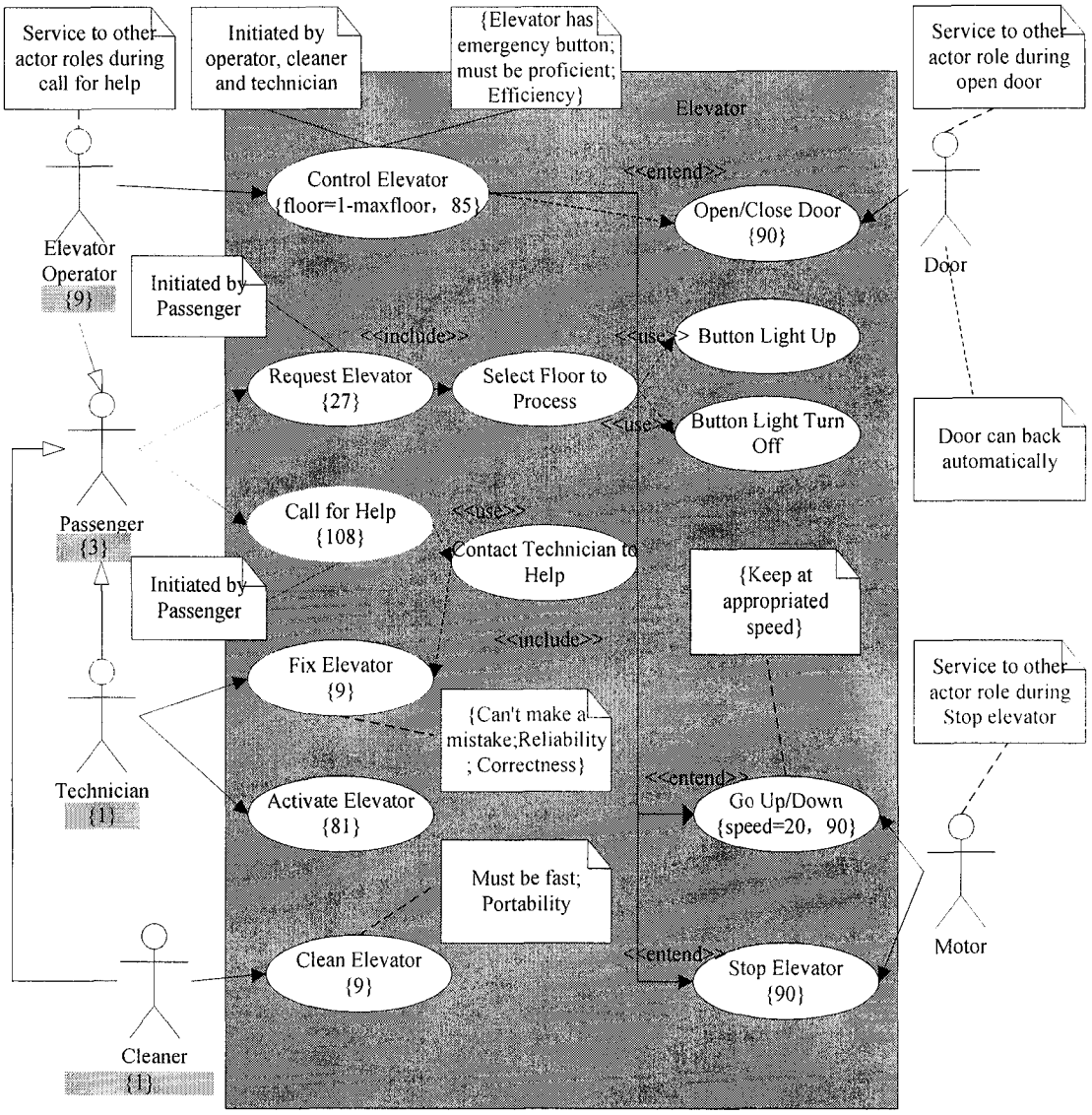


Figure 8.32: New User Case Diagram of Elevator System

Figure 8.32 user case diagram describes what the system does and how it interacts with the user.

(1) "Users X Actor Role" and "Actor role X Use Case" illustrate relative importance of the role by value, so we might use the value of the "Role wt" extracted from "Users x Actor Role" and "Actor role X Use Case" in the use case diagram to indicate the actor's prioritization. We adopt "Tagged value" (one of UML extension mechanisms) to insert that information into a use case diagram. The benefit of this improvement is to add more details about the role in the use case diagram. So, those use case diagrams can represent more information and are easier to understand. Figure 8.32 describes in detail.

(2) We use the arrow to indicate the relationship among the actor roles shown on

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the left of the diagram.
(3) We employ constraints (a UML extension mechanism) to represent the information derived from "Use Demanded Quality X Use Case" so that each use case can be easier to understand.
(4) The system requirement extracted from cost-benefit analysis chart might be used to explain the use case or actors in the diagram by a constrained method.
(5) Use Cases X Data Attributes illustrates the data attribute that is carried out by different use cases. Here we can use this kind of information to describe the use case by tagged value or constraints methods.
(6) IEEE quality factor extracted from "Use Case X IEEE Quality Factor" also can be added into the use case diagram by constraints.
(7) "User Demand Quality X Use Case" matrix depicts the user demanded quality for each use case. Customer requirements are extremely important in the beginning stage of software design. Normally, customers will bring forward lots of requirements, but not all of them are valuable. So we use a need-opportunity matrix to prioritize user demands, and then combine the more valuable customer requirements with the "User Demand Quality X Use Case " matrix. Finally, we represent this information in use case diagram by the "constraint" extension mechanism.
(8) "Actor role X Use Case" matrix describes the function weight of each use case. We also adopt the same method in (1) to represent information in the use case diagram. In addition, we can represent the symbols, such as "I" and "S" of the "Actor role X Use Case" into the use case diagram to indicate which use cases are initiated by actor roles and which provide services to other use cases. For example, the role of Door initiates the use case control elevator, and then provides a service to the Elevator Operator during Open/Close Door.
(9) We represent some non-functional requirements categorized by cost benefit analysis into use case diagram. We highlight the description by red to indicate it is in the " high value " quadrants of this method and by blue to indicate it is in the " targeted" quadrants of the method. This representation approach is sub-optional and it will be enhanced in future work.

So elevator basic scenarios that can be extracted from Use Case Diagram are:

- Elevator Operator control Elevator
- Passenger request elevator and call for help
- Technician fix Elevator and Activate Elevator
- Cleaners clean Elevator
- Open /Close Door
- Motor stop Elevator or make it go up/down

There are nine main use cases based on the requirement documentation of the elevator system, as shown in Figure 8.32:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Elevator operator controls Elevator: Controller open/close Door.
- Request Elevator and call for help: Passengers push the Floor button, then elevator receives hall calls from the passengers, turns on or turns off the light of hall call buttons, updates the record of hall calls in system controlling parts; If there are some emergency, passenger contact technician to help.
- Technicians fix Elevator and Activate Elevator: Technician will take action to fix elevator when got call from passenger. After uncovering the system, they will activate the elevator.
- Cleaners clean Elevator.
- Move/Stop the Car: The main function of an elevator, detailed action will include the changing of driving speed, how to make the decision of stop, and driving directions of the car.
- Open/Close the Doors: The elevator should be able to open and close the doors for the passengers to get in and out of the car. The functional areas of this use case should also enable the passengers to make door reversals when the doors are closing and the passenger wants to get in the car.

Elevator operator is the role that humans play when interacting with the system. The passenger interacts with the Elevator system by making car and hall calls. A passenger also makes decisions whether to enter/leave the car or not by observing the indication of moving direction and car position. Therefore the use case diagram shows that the actor has relationships with three use cases of the system: control Elevator, Open/ Close Door, Go up/down and Stop Elevator.

8.3.3 Domain model (class diagram)

Class diagrams show the static structure of each class, their internal structure, and their relationships. From the use cases of the Elevator system and the requirements of the system, we can derive a class diagram as shown in Figure 8.32.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

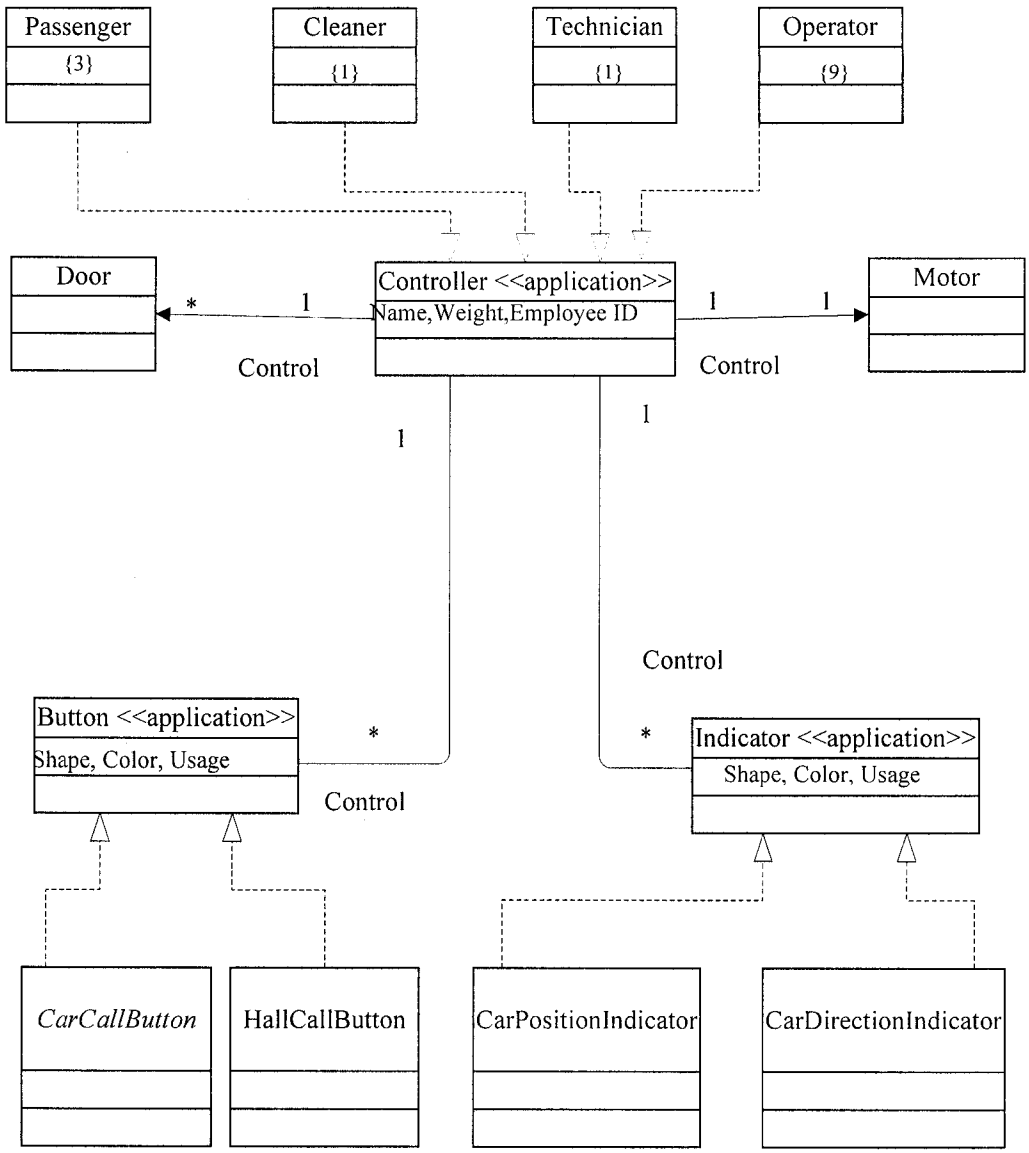


Figure 8.33: New Class Diagram of Elevator System

(1) Based on the matrix: Object x Data Attributes, we add 3 new superclasses in Figure 8.33 by the stereotype method. In the "Classes X Date Attributes" matrix, we can find that Passenger, Cleaner, Technician and Operator classes share some same attribute, but their work category is different. So we can create a new superclass <<controller>> that can cover all these attributes of those four classes. Moreover, we also can create another two new superclasses <<button>> and <<indicator>> that cover carcallbutton, hallcallbutton, carpositionindicator and cardirectionindicator.

(2) For some classes, we create new-tagged values that are used to hold participant names associated with this given class and keep these values in the second compartment of some classes. In this way, related information is treated as first-class members in the same way as attributes and operations of a class. We can get that information from the matrix "Classes X Data attributes" and add it

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

into class diagram by the tagged value mechanism.
(3) In additional, we highlighted the class that we changed in order to indicate that it is an application class.
(4) The "Use Cases X Actor Role" matrix describes the relative importance of each role and also can be represented in the class diagram by the "tagged value" extension mechanism.
(5) "Classes X Superclasses" matrices show the relationship between classes and superclasses, so all the classes can be represented as a new class by the stereotype extension mechanism. Furthermore, the "Classes X Data Attribute" matrix can provide attributes for each class, so we can add them into the class diagram. The "Classes X Classes" matrix will describe the relationship among classes and can be shown in the class diagram.

- **Elevator Controller:** The main controller in the elevator system. Elevator Operator communicates and controls all other objects in the system.
- **Door:** There are two doors in the system, the "god" class - the ElevatorControl – commands the doors to open and close, according to the situation stated in the use case.
- **Motor:** The car is being controlled to move up or down (at different speeds), making stops at floors when necessary.
- **Button:** The ElevatorController class also controls the button class, which further generalizes two subclasses CarCallButton and HallCallButton. The control object communicates with the Button objects, retrieving information on whether the button is pressed and controls the illumination of the Button lights.
- **Indicator:** There are two kinds of indicators in the system, the CarPositionIndicator and the CarDirectionIndicator. The indicators are controlled to show the information about the current position and moving direction of the car.

8.3.4 Business process model (activity diagram)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Activity diagrams represent the business and operational workflow of a system. An Activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state. The following diagram describes the activity of passenger.

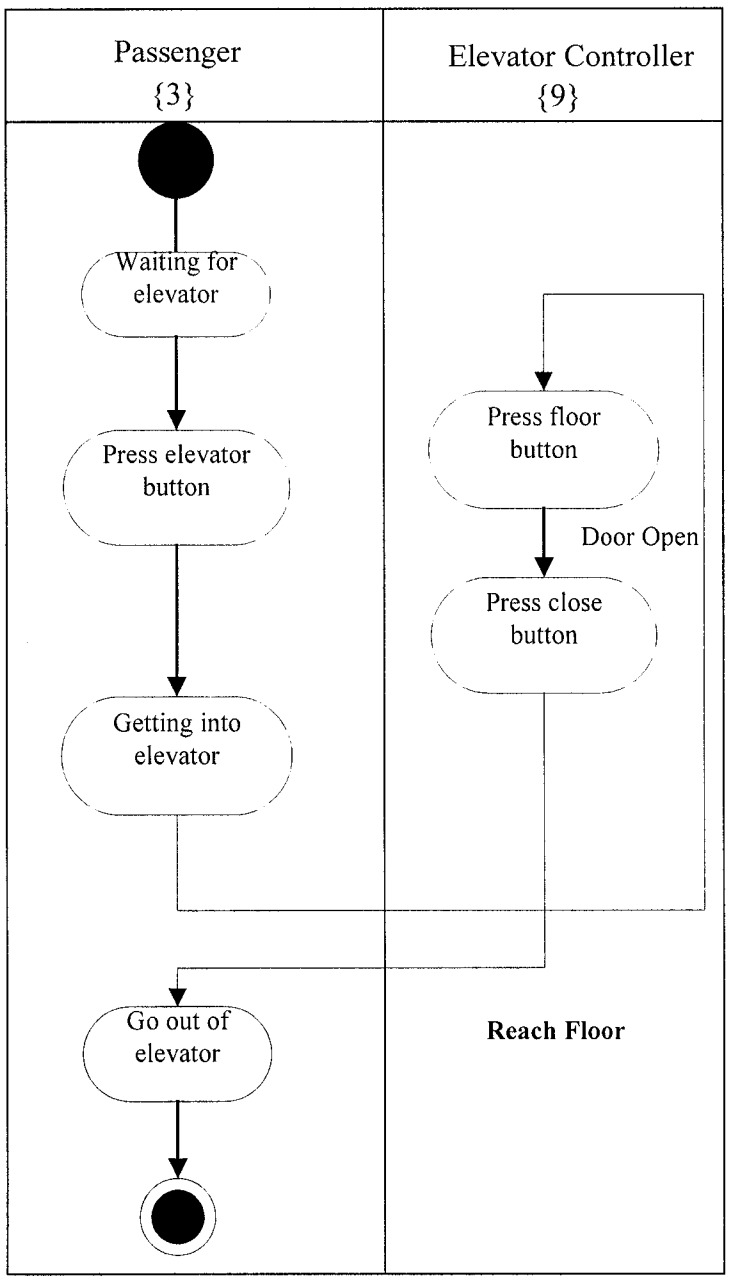


Figure 8.34: New Activity Diagram of Elevator System

(1) The biggest disadvantage of traditional activity diagrams is that they do not make explicit which classes execute with which. Therefore, the labelling of each activity with the responsible class will be useful for designers to understand the overall process more exactly. According to the Use Case x class matrix, we make modifications within Figure 8.34 and add a class to each activity.

(2) There is no constraint imposed on the nature of a role nor the consistency

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

amongst the roles defined (e.g. all objects, all organizational units, etc.) So this deficiency restricts the application of the concept of responsibility to activity diagrams." User X Actor Role" matrix and "Actor Role X Use Case" matrix can supply information to set constraints on the objects in the activity diagram, we can use the constraint extension mechanism to represent this information. Moreover, the " class X Class" matrix shows the relationship between classes and superclasses, which can also be added.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

9.0Conclusion

Even though UML and RUP are very popular methods that have been used in a wide variety of projects and organizations, they still have some limitations. In this thesis, the rational unified process is employed to integrate Quality Function Deployment, Cost Benefit Analysis, and Usage oriented requirement engineering into UML in order to improve the UML diagrams and notations, and further promote superior designs in software system.

This thesis proposes a new software design process and notation. In the initial stage of software design process, the QFD-style matrix is employed to capture, organize and analyze customer non-functional requirements in order to represent them into UML diagram and notations. Then, cost benefit analysis is applied to categorize the customer needs into different levels of importance. Finally, the UORE is integrated into this software design process to improve the UML diagram and enhance the quality of system design. Although there are some others methods that can also be used to overcome the limitations of UML and RUP, these three methods appear to be the most effective tools for designers in requirement analysis.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

BIBLIOGRAPHY

Alhir, Sinan Si .*Understanding the Unified Modeling Language (UML)* O'Reilly and Associates, Inc., 1998

Amatya, Ananda. "*Introduction to UML*", Department of Computer Science, University of Warwick, October 11, 1999.

Ambler, Scott W. *Enhancing the Unified Process:Software Process for the Post-2000 (P2K) World A Ronin International White Paper.*Ronin International Inc. Copyright 1999-2000

Barrett, Randy. *Chasing the BPR Tool Market Page*. Enterprise Reengineering, March 1996. <www.dtic.mil/c3i/bprcd/5316.htm>

Benjamin Musial Timothy Jacobs. *Application of Focus + Context to UML.* Department of Electrical and Computer Engineering,Air Force Institute of Technology,Dayton, Ohio, USA 45430

Berner, Stefan., Glinz ,Martin., and Joos ,Stefan. *A Classification of Stereotypes for Object-Oriented Modeling Languages*, University of Zurich (Germany): 2000.

Bergner,Klaus., Rausch, Andreas., Sihling Marc. *Using UML for Modeling a Distributed Java Application.* T E C H N I S C H E U N I E R S I T T M N C H E N 1997

Better,GK Khalsa. Metadate Management through Better Metamodels, Objectrad,Temecula,CA.

Booch, Grady. Object-Oriented Analysis and Design with Applications Second Edition. Addison-Wesley. Santa Clara, CA. 1998.

Booch, Grady., Rumbaugh, James., and Jacobson ,Ivar .The Unified Modeling Language User Guide, September, 1998

Boushi ,M. "Towards better object-oriented software with QFD,", Transaction of 6th Symposium on QFD.

Buede, Dennis. *Elevator System Case Study.* Department of Systems Engineering and Engineering Management. Jr.School of Engineering, Stevens Institute of Technology. Hoboken,NJ.

Buede, Dennis *SYSTEM DESCRIPTION DOCUMENT FOR Elevator Control Prepared For: Up and Down Elevator Co.* Sunday, March 24, 2002

Cesare ,Sergio de. Mark Lycett BUSINESS MODELLING WITH UML: DISTILLING DIRECTIONS FOR FUTURE RESEARCH, Department of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

information Systems and Computing, Brunel University.

Chulani, Sunita., Santhanam ,P., Moore ,Darrell., Leszkowicz ,Bob., and Davidson, Gary. *Deriving A Software Quality View from Customer Satisfaction and Service Date*, 1992

Clarke, Siobhán *Extending UML Metamodel for Design Composition.*School of Computer Applications,Dublin City University,Dublin 9,Republic of Ireland.

Cox, Karl and Phalp, Keith Thomas *Use Case Authoring: Replicating the CREWS1 Guidelines Experiment.* Empirical Software Engineering Research Group,School of Design, Engineering and Computing, Bournemouth University,Talbot Campus, Fern Barrow, Poole, Dorset, BH12 5BB

Davis, Guy.,Zannier, Carmen.,Geras, Adam. *Quality Function Deployment QFD for Software Requirements Management*

Damelio, Robert. *The Basics of Processing Mapping.* Quality Resources. New York, 1996.

Dean ,Edwin B. *Parametric Cost Deployment.* Multidisciplinary Design Optimization Branch, NASA Langley Research Center. Hampton, VA 23681.

Dewalt ,Craig. "*Business Process Modeling With UML*", Hopkins University, 7 December 1999.

Disegno, Analisis. Var Jacobsen Interview.
<http://www.analisis-disegno.com/uml/JacobsenInterview.html>, 1999.

Dong, Jing *UML Extensions for Design Pattern Compositions.* Department of Computer Science, University of Texas at Dallas, Texas, USA

Drusinsky, Doron. *How To Make Statecharts work for you*

Dünser,Thomas., Deplazes, Romeo.,Meier, Markus. *A New Elevator System And Its Implementation* .Center of Product Development, ETH Zurich, Switzerland

Engels,Gregor.,Heckel,Reiko.,and Sauer ,Stefan. *UML - A Universal Modeling Language?*
University of Paderborn, Dept. of Computer Science, D 33095 Paderborn, Germany

Eshuis ,Hendrik. *SEMANTICS AND VERIFICATION OF UML ACTIVITY DIAGRAMS FOR WORKFLOW MODELLING.* geboren op 17 september 1975

Evans, Andy S. *Reasoning with UML Class Diagrams.* Department of Computer Science,University of York, Heslington York, UK

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fontoura, Marcus and Lucena, Carlos *Extending UML to Improve the Representation of Design Patterns.*Software Engineering Laboratory (LES)Computer Science Department, Pontifical Catholic University of Rio de Janeiro Rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, Brazil

François, Pinet and Ahmed, Lbath *Semantics of Stereotypes for Type Specification in*

Freydenson ,Elan. *Bringing Your Personas to Life in Real Life.* <http://www.boxesandarrows.com/archives/002343.php>.Marca, David A. and

Fuentes, José M., Quintana, Víctor *Errors in the UML Metamodel?* Research Department,dTinf S.L. Leganés- Madrid, Spain ,Juan Llorens, Gonzalo Génova ,Rubén Prieto-Díaz

Giacobazzi ,Roberto. *Completeness in abstract interpretation: A domain perspective.* Univ. di Pisa, Corso Italia 40, 56125 Pisa (Italy)

Gogolla, Martin. *Using OCL for Defining Precise . Domain-Specific UML Stereotypes* .University of Bremen, Computer Science Department.Postfach 330440, D-28334 Bremen, Germany.

Graham I. 1991. *Object Oriented Methods*. Reading, MA (USA): Addison-Wesley.

Grunske, Lars. *Roland Neumann Quality Improvement by Integrating Non-Functional Properties in Software Architecture Specification,*

Harrington, James., Esseling ,Erik K. C., and Nimwegen ,Harm Van. McGraw-Hill.Business Process Improvement Workbook, New York, 1997.

Huemer, Christian. *Defining Electronic Data Interchange Transactions with UML.*

Hucka, Michael., Doyle, John and Kitano, Hiroaki. *SCHUCS: A UML-Based Approach for Describing Data Representations Intended for XML Encoding.*Systems Biology Workbench Development Group,ERATO Kitano Systems Biology Project Control and Dynamical Systems, California Institute of Technology, Pasadena, CA 11 December 2000

Jacobson, Ivar *Applying UML in The Unified Process Rational Software*

Jacobson, Ivar. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Reading, MA (USA): Addison-Wesley, 1992.

Jazayeri, Mehdi and Podnar, Ivana. *A Business and Domain Model for Information Commerce*. Information Systems InstituteDistributed Systems Group May 24, 2000.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Kandé, Mohamed Mancona., Mazaher ,Shahrzade., Prnjat ,Ognjen., Sacks, Lionel., and Wittig, Marcus. 1 *Applying UML to Design an Inter-Domain Service Management Application.* GMD-Fokus, Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany

Kaufman, Morgan *A UML Documentation for an Elevator System Distributed Embedded Systems*, Fall 2000

Koopman, Philip *UML-Based Design Process(with an emphasis on course project survival)Distributed Embedded Systems.* September 11, 2002

Kulik ,P., MacDonald, D., "Bridging Gaps in Customer Requirements",1998.

Lamia ,Walter M.. "Integrating QFD with OO software Design Methodologies," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890

Lauesen, Soren. *Task Descriptions as Functional Requirements,* IT-University of Coperagen, 2003

Lions, Jean Marie., Simoneau, Didier. Pitette, Gilles. Moussa, Imed. *Extending OpenTool/UML Using Metamodeling:An Aspect Oriented Programming Case Study.* TNI-Valiosys, France,Technopole Brest-Iroise BP 70801

Madsen, Søren Langkilde. *Use case modeling*

Martin, Mark V. , Kmenta, Steven., Ishii, Kos *QFD AND THE DESIGNER:LESSONS FROM 200+ HOUSES OF QUALITY.*Mechanical Engineering Design Division,Stanford University, Stanford, CA 94305-4022 USA

McGowan ,Clement L.. *IDEF0/SADT Business Process and Enterprise Modeling.* Eclectic Solutions Corporation San Diego CA. 1988.

McQuaid, Heather L., Bishop ,David. *Communicating the Priority of Problems Discovered During a Product Evaluation.* MAYA Design, Inc. USA

Monarchi, D.E. and Puhr, G.I. "*A Research Typology for Object-Oriented Analysis and Design,*" *Communications of the ACM 35*, 9 (Sept 1992): 35-47.

Moraes ,Ian. *Introduction to the Unified Modeling Language (UML),* Glenayre Electronics, September 1998

Naumenko, Andrey., Wegmann, Alain. *A metamodel for the Unified Modeling Language: critical analysis and solution.*Systemic Modeling Laboratory,Swiss Federal Institute of Technology – Lausanne.EPFL-IC-LAMS, CH-1015 Lausanne, Switzerland

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Pallinane, Madhavi V.,Uppalapati, Neelima.,Kesani, Niranjan.,Gottipati, Sampath *SIMULATION OF AN ELEVATOR SYSTEM.* Computer Science,Mississippi State University,07/31/2001

Paltor, Ivan Porres *Digital Sound Recorder: A case study on designing embedded systems using the UML notation.*Åbo Akademi University, Department of Computer Science, Lemminkäisenkatu 14, FIN-20520 Turku, Finland

Pinet ,Francois. and Lbath ,Ahmed. *Semantics of Stereotypes for Type Specification in UML: Theory and Practice.* Laboratory of Information System Engineering, INSAyon: France.

Pollice, Gary, *Rational Software Using the Rational Unified Process for Small Projects :Expanding Upon Extreme Programmiing*

Quatrani, Terry. Introduction to the Unified Modeling Language, June 2003

Regnell ,Bjorn., Kimbler ,Kristofer., and Wesslen ,Anders. *Improving the Use Case Driven Approach to Requirements Engineering.* Dept. Communication Systems, Lund University, Sweden, 1995.

Reznik, Julia., Born, Marc.,Fokus, GMD *UML Profile for DCL* Rosenberg ,Doug . *Use Case Driven Object Modeling -- A 99% Fat-Free Approach.* ICONIX Software Engineering, Inc.

Regnell, Björn., Kimbler ,Kristofer., Wesslén ,Anders. *Improving the Use Case Driven Approach to Requirements Engineering.* Dept. of Communication Systems, Lund University, Sweden

Rua Marquês de São Vicente *Extending UML to Improve the Representation of Design Patterns Marcus Fontoura and Carlos Lucena.* Software Engineering Laboratory (LES),Computer Science Department, Pontifical Catholic University of Rio de Janeiro, Brazil

Rumbaugh ,James., Jacobsen, Ivar., and Booch, Grady. Addison-Wesley. The Unified Modeling Language Reference Manual Reading, Massachusetts 1999.

Schleicher, Ansgar., Westfechtel, Bernhard. *Beyond Stereotypin : Metamodeling Approaches for the UML.* Department of Computer Science III, Aachen University of Technology,Ahornstr. 55, 52074 Aachen

Sendall, Shane and Strohmeier, Alfred *Requirements Analysis withUse Cases Theory (9 Lessons).*Swiss Federal Institute of Technology in Lausanne Software Engineering Lab

Shlaer, S. and Mellor, S.J. *Object Lifecycles: Modeling the World in States.* Englewood Cliffs, NJ (USA): Yourdon Press, 1992.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Song, Il-Yeol *Developing Sequence Diagrams in UML.*College of Information Science and Technology,Drexel UniversityPhiladelphia, PA 19104

Terrasse ,Marie-No¨elle., Savonnet ,Marinette., Becker ,George., and Leclercq, Eric. *A UML-based metamodeling architecturewith example frameworks* .Laboratoire LE2I, Universit´e de Bourgogne B.P. 47870, 21078 Dijon Cedex, France

Tyndale-Biscoe, Sandy.Sims, Oliver.,Wood, Bryan.,Sluman, Chris. *Business Modelling for Component Systems with UML Open-IT Limited*

Vasconcelo, Alexandre Marcos Lins de s. *Extending the Rational Unified Process for the Development of Web Applications*. September 2002.

Villiers, DJ de.Empulsys. Rational Home Page. Rational Software Corporation. <http://www.rational.com> 23 June 2003.

Warmer, Jos., Objecten ,Klasse. *The future of UML*

Wesley,Addison. Literate Modeling Papers Page. Literate Modeling. http://www.literatemodeling.com/papers.htm 2002.

Williams, Clay E. *Software Testing and the UML* Center for Software Engineering, IBM T. J. Watson Research Center

Xie,Zhiwu., Liu, Kecheng. *Improving Business Modelling with Organisational Semiotics*. Department of Computer Science .The University of Reading, UK

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.