Design and Implementation of End-User Programming Tools for Web Mashups

by

Yumeng Gao

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

Master of Science

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering

University of Alberta

Abstract. Web mashups are defined as lightweight web applications that are composed of content, functionality and presentation from diverse web sources. The majority of web users are end-users with limited or no programming experience. In order to address varying requirements of end-users to better support their business and personal activities, end-user programming has become the most common form of programming in recent years. An emerging research focuses on the design and implementation of end-user programming tools for web mashups. This paper presents design guidelines and key factors of building a successful mashup tool from the aspects of usability and human-computer interaction (HCI). Usability is analyzed from five dimensions: usefulness, ease of use, ease of learning, error tolerance and user satisfaction. Several cognitive dimensions, cross-platform UX, liveness scales, automation degree and collaboration community are considered as significant HCI factors. To achieve these success factors, a collection of techniques and methodologies are provided including Model-View-ViewModel design pattern, responsive web design, progressive enhancement process, drag-and-drop and WYSIWYG (What You See Is What You Get). This paper also introduces a practice of the mashup tool. This mashup tool provides a user-centered, rich-featured and high-interactive composition approach that enables non-programmers to create their own applications in a graphical environment. A user-oriented, within-subject design experiment was conducted to evaluate the mashup tool. The evaluation results were promising and indicated that the proposed mashup tool has a high level of usability for end-users, and has adequately integrated HCI factors into the user experience design.

Table of Contents

1. Introduction 1
2. Related Work 2
2.1 End-User Programming 3
2.2 Visual Programming 5
2.3 Web Mashups 7
2.3.1 Mashup Components 8
2.3.2 Consumer Mashups and Enterprise Mashups 11
3. Theories 12
3.1 Viewpoint of End-users 13
3.1.1 Usefulness 14
3.1.2 Ease of Use 14
3.1.3 Ease of Learning 15
3.1.4 Error Tolerance 16
3.1.5 Satisfaction 16
3.2 Viewpoint of Developers 16
3.2.1 Cross-platform 18
3.2.2 Levels of Liveness 18
3.2.3 Automation Degree 19
3.2.4 Modularity 19
3.2.5 Collaboration Community 20
3.2.6 Viscosity 20
3.2.7 Closeness of Mapping 21
3.2.8 Consistency 21
3.2.9 Diffuseness 21
3.2.10 Responsiveness 21
4. Techniques 22
4.1 Model-View-ViewModel (MVVM) 22
4.1.1 The Evolution of MVVM 22
4.1.2 Basic Concepts 23
4.1.3 Benefits of MVVM for Mashup Tools 24
4.2 Responsive Web Design 27
4.2.1 Fluid Grid 28
4.2.2 Fluid Image 29

4.2.3 Media Queries 30
4.3 Progressive Enhancement 30
4.3.1 Testing Browser Capabilities 31
4.3.2 Benefits of Progressive Enhancement for Mashup Tools 32
4.4 Drag-and-Drop 33
4.5 WYSIWYG 34
5. Implementation 34
5.1 Vue.js Framework 34
5.1.1 Single File Components 35
5.1.2 Supporting Libraries 35
5.1.3 MVVM in Vue.js 36
5.1.4 Performance 37
5.2 Grid System 39
5.3 Navigation 40
5.4 Component Panel 41
5.5 Responsive Components 41
5.6 Custom Panel and Toolbars 42
5.7 WYSIWYG Text Editor 42
5.8 Autosave 43
5.9 Modernizr 44
6. Case Study 44
Step 1: Create a new project and a new web page 44
Step 2: Design the web page layout 46
Step 3: Drag-and-drop mashup components into the layout 47
Step 4: Customize CSS properties of the mashup components 49
Step 5: Link pages together 52
7. Evaluation 52
7.1 Participants 52
7.2 Evaluation Design 53
7.3 Evaluation Tasks 54
7.4 Evaluation Scale 55
7.5 Interpretation of Results 58
8. Conclusion and future work 61
References 63

1. Introduction

Web applications have been developing rapidly in today's world. Users can easily access the application using any device connected to the Internet via a uniform environment - the web browser. Web-based approach also makes the application easier to develop, install and maintenance. Different users have different functionality requirements based on their personal and business activities, which facilitate the development of web mashups. Web mashups are lightweight web applications that are composed of multiple content, presentation and functionality from disparate web sources. End user programming tools for web mashups (i.e. mashup tools) can be designed to allow end users who have little or no programming knowledge to combine different mashup components in a visual programming environment, and create their own web applications according to their specific needs.

This paper proposes a collection of design guidelines and key factors to build a successful mashup tool from the viewpoint of both end users and developers. To design and evaluate the mashup tool from the viewpoint of end users, this paper presents a five-dimensions usability principle: usefulness, ease of use, ease of learning, error tolerance and user satisfaction. To improve the performance of human-computer interaction in the development process of the mashup tool, this paper references and extends the definition of cognitive dimensions including modularity, viscosity, closeness of mapping, consistency, diffuseness and responsiveness, as well as the concepts of cross-platform UX, liveness scales, automation degree and collaboration community.

To achieve these key factors, this paper provides several techniques and methodologies and introduces the concepts and benefits of them. Model-View-ViewModel (MVVM) design pattern improves modularity, maintainability, efficiency and testability of the mashup tool. Adopting responsive web design and progressive enhancement process ensures better usability and cross-platform UX. Drag-and-drop and WYSIWYG (What You See Is What You Get) are also widely used in the development of end user programming tools. Drag-and-drop makes the mashup tool easier to learn and use for non-programmer users. WYSIWYG allows the tool to provide immediate visual feedback to users.

This paper presents in detail the implementation of an innovative, rich-featured, responsive and highly-interactive mashup tool. This tool is developed using Vue.js framework. Vue framework supports massive libraries and has high performance. Mashup components are defined in

separate loosely-coupled Vue instances, which represent the ViewModel in MVVM design pattern. In addition, this paper introduces four sections of the mashup tool: navigation, component panel, mashup canvas and custom panel/toolbars. End users can build the layout of their web applications by dragging Bootstrap grid system into the mashup canvas. Component panel provides a variety of web services and Bootstrap responsive UI components. Custom panel and toolbars allow users to customize CSS properties of all mashup components. Also, this paper presents a case study and illustrates the process of creating a University of Alberta website using the mashup tool.

To evaluate the usability and interactivity of this mashup tool, a within-subjects design experiment was conducted. Every participant was asked to perform one task with the mashup tool, and the other task without the mashup tool, then complete a questionnaire with defined evaluation criteria. Usability was evaluated using System Usability Scale (SUS), and the SUS score is measured using Sauro-Lewis curved grading scale. The evaluation criteria of HCI factors were measured using five-grades Likert Scale. The evaluation results show significant evidence of the superiority of this tool in terms of system usability. These results also strongly support the applicability and effectiveness of suggested design guidelines and techniques to achieve high human-computer interaction.

The paper is organized as follows. Section 2 introduces the definition, main features and related research of end-user programming, visual programming and web mashups. Section 3 proposes a series of design guidelines to build a successful mashup tool. Section 4 provides a collection of techniques and methodologies to implement the key factors. Section 5 presents the implementation details of the mashup tool. And section 6 presents a case study of how to build a website using the tool. Section 7 shows the evaluation design, process, criteria, scale and interpretation of evaluation results. Finally, conclusion and future work are drawn in Section 8.

2. Related Work

Web mashup is a web-based composition approach that enables end users to develop new applications using a visual programming language. In this section, I introduced the definition, main features, purposes, related research and applications of end user programming, visual programming and web mashups.

2.1 End-User Programming

As the number of personal computers and personal workstations increases, along with the sharp reduction of computer operation systems and software costs, the population of computer user community has expanded dramatically. Most people simply use canned application software available for their computers. Software programmers create application software with specific functions for a majority of users, coupled with different packages for multiple additional functions. However, they cannot satisfy every single user with all capabilities he or she needs, also the software cannot be rewritten or modified by users. Those who have extra functionality requirements beyond what the software is capable of have found out that they have to program. Therefore, research has been moving in the direction of the field of end-user programming.

An end user is simply any computer user. Ko defines the phrase end-user programming as "programming to achieve the result of a program primarily for personal, rather public user" [2]. End-user programmers might be system administrators, artists, teachers, students, managers, scientists, engineers or anyone else who writes any kind of program themselves, varying needs to better work or to support their hobbies. For example, an email user may write email rules to manage, sort and filter e-mail; a photographer might write a Photoshop script to apply the same filters to hundreds of photos; a geologist might write a MATLAB script to simulate seismic wave propagation; or an accountant may use spreadsheets at home to tabulate and keep track of family financial loans. All the programs listed above are primarily intended for personal use in their own domains of expertise but not for a broader group of users.

This definition is also applicable for experienced and skilled software developers. It is really important to not conflate end-user programming with inexperience. Professional developers who have many years of research or work experience in software programming may still engage in end-user programming by writing code to support their own task. For example, a programmer may write a script to automatically retrieve and store data with thousands of inputs to save time; or write a code segment to visualize a data structure to help diagnose a bug. In this case, the program's first priority is to support the programmer's own task but not for a large number of use cases.

End-user programming is a construction process of programs, focusing initially on exploring and developing technologies and tools to make programming simpler. For instance, visual programming, introduced in Section 2.2, has been verified to be a recognizable approach. Users program with a set of interaction techniques and graphical representations instead of traditional,

text-based languages. End-user programming has multiple dimensions [3]: software engineering, human-computer interaction and organizational use.

From a software engineering perspective, end-user programming brings the benefits of rigorous software engineering methodologies to end users. A wide variety of methodology frameworks such as prototyping, waterfall model and spiral model are involved in the development of generic applications. Generic applications include multifunctional applications that can be configured to different user needs, domain independent tools, or application generators that support users in creating new applications. Therefore, generic applications can be characterized as development tools which are supportive of end-user programming.

From the view of human-computer interaction, end-user programming is user-oriented techniques that turn easy-to-use information and communication technology into easy-to-develop and easy-to-maintain systems. The main goal of end-user programming is to improve and evaluate programming usability. More specifically, usability can generally be slotted into following factors: usefulness, learnability, effectiveness, low error rate and satisfaction. Usefulness ensures that all user requirements are met. Learnability means easy to learn so users can get started rapidly. Effectiveness guarantees that users can use it effectively with high productivity. Low error rate ensures that users could make few errors and satisfaction means less frustrations for users.

From the perspective of an organizational user, it is common to launch a generic, multipurpose application system among the whole organization that allows all employees to access to the necessary information rapidly. Employees can be novice or experienced computer users, ranging from the young to the mature, with different abilities and disabilities, and different cultural, educational, training and employment backgrounds [4]. Therefore, the support of end-user programming is fundamental in order to resolve the difference between a generic system and user-specific needs. For example, an accounting firm in Scandinavia adopts Visma Business [4], a comprehensive financial and accounting system, for their employees to process the business. Depending on different employee groups with different access rights, also depending on the size of business and clients that accountants will interact with, the system can be configured to different organizational needs. The support of end-user programming achieves an efficient and diverse usability of a complex application system.

Researchers began to work on ways to provide formal software engineering methodologies to end-user programmers. They started to invent new kinds of technologies and tools that

collaborate with end-user programming to improve software quality and most of them are related to testing and debugging. In the contrast of professional software engineering, people who are engaging in end-user programming are not likely to know about systematic and disciplined software engineering activities [5]. Also they rarely have the time or interest in learning relative knowledge about quality control mechanisms and test adequacy criteria. Their first priority is to create programs to support their own domains of work and hobbies. Segal analyzed the problems of end-user development in [6] after drawing on data from field studies of different groups of end users and indicated that testing was not being taken seriously. It was partly due to pressures of time and exacerbated by the iterative evolutionary development rather than having separate phases of development and maintenance, causing later parts of the development process such as testing and debugging got squeezed.

Given these differences, the major challenge of end-user programming is to provide software quality assurance mechanisms without requiring users to transfer their priority (run mathematical algorithms, analyze massive amounts of data, modeling and simulation, etc.) This can be done by developing techniques or methods that can incorporate testing and debugging activities into the current pattern of software development. For example, providing testing tools to validate links and corresponding pages when developing web applications; or tools that can validate data inputs and provide instant visual feedback to end-users when developing spreadsheet applications.

2.2 Visual Programming

Conventional programming languages such as C, C++, Java, JavaScript, Python, etc. are difficult to learn for people who have no or little experience. The processes of programming, testing and maintaining are also complex, time-consuming and labor-intensive. The lack of experience may lead to “analysis paralysis”. Developers spend a long period of time on project planning, software requirements analysis, framework design and data modelling. They focus excessively on achieving perfection and completeness during the analysis phase. However, the extra time devoted to information gathering and decision making no longer add any significant value to the project. So the real challenge turns into the ease of programming. To solve this challenge, recently researchers have been directed towards an increase in adoption of high-level programming languages, which makes the process of application development simpler and more understandable than lower-level languages.

Visual programming describes any system that allows users to create and edit a program using two (or more)-dimensional representations, in contrast, linear programming language is processed by compiler as a one dimensional stream composed by textual characters. The primary driving forces of this research is (a) to support the development of complex and large-scale software applications; (b) to aid end-users to build different software modules and assemble them graphically into sophisticated applications; and (c) to increase productivity by allowing the programmers to abstract the complex programming statements to relatively straightforward visual level.

Visual programming language is a language that consists of visual syntax. The program syntax is represented with at least two dimensional elements, characterized by multiple visual expressions, such as location, shape and color. According to the different types and scopes of visual syntax being used, visual programming languages may be classified into programming paradigms and visual representations [7]. Programming paradigms contains concurrent languages, constraint-based languages, dataflow languages, form-based/spreadsheet-based languages, functional languages, imperative languages, logic languages, multi-paradigm languages, object-oriented languages, programming by demonstration languages and rule-based languages. Visual representations contain diagrammatic languages, iconic languages, languages based on static pictorial sequences and sound-based or speech-based languages.

Dataflow programming languages are the core to most visual programming languages. It is a programming paradigm that models a program as a directed graph consist of arcs and boxes. More specifically, programs are represented as a set of nodes, which could be either sources, sinks, processing blocks, or more complex structures such as subprograms. Each node is an executable block. It retrieves input data, performs computation and transformations over it and passes output to the next block. Each node can be connected to zero or more terminals, and it is not necessary to name the nodes or the connection arcs.

The three main features of dataflow programming languages are: iteration, procedural abstraction, and a large and diverse library of predefined functions that can be used as building blocks in construction new programs [8]. Dataflow programming languages implement a variety of iteration constructs. For example: cycles in the dataflow graph; control flow constructs such as FOR and WHILE loops; parallel iteration that runs a series of operations on all elements independently; and sequential iteration that recycle a single changing value through repeated executions of the iteration box. Procedural abstraction is the process of creating procedures or

subroutines that encapsulate subtask details [9]. An entire graph is considered as a procedure and condensed into a single node. Input and output arcs connected to this node are considered as arguments to this procedure. Procedural abstraction allows programmers to view the program at a higher level and separate details of how to carry out one procedure from another procedure that uses this procedure. Also it saves screen space and makes a visual program more compact.

Visual programming is widely used to support application construction, system design and simulation, image processing, computer graphics, data visualization, and user-interface generation. For example:

- VisSim [10], developed by Visual Solutions in Westford, Massachusetts, is a visual programming language for modeling, simulating and controlling complex large-scale dynamic systems. It uses a block diagram approach. Users can simply drag and drop blocks listed in the workspace, connect them with wires and run the simulation. VisSim automatically turns block diagrams into executable well-tested C code, then executes the simulation with no compilation delay and generates instantaneous and accurate response. It resolves the requirements of experienced programmers and reduces time cost and human error.

- GeoVISTA Studio [11], developed by the Pennsylvania State University, GeoVISTA Center and Department of Geography, is designed to support visual and computational exploration, evaluate hypotheses, analyze and visualize sophisticated geoscientific data. It applies component-based development. Prototypes are built by assembling a series of independent components together via a visual user interface. The user interface is a real-time design window. GeoVISTA Studio analyzes and displays the results instantaneously when any changes made to the prototypes. It also provides a library that contains lots of well-developed reusable functionality, which increases productivity and accuracy.

2.3 Web Mashups

With the popularization of computing and the Internet, more and more people can access to millions of websites in their daily life, web application development has evolved considerably. The ubiquity of web browsers makes it easy to maintain and update web applications without distributing and installing software on thousands of client computers. Also web applications have inherent support for cross-platform compatibility. The development of web applications

started from contents and functionality in the form of open APIs or reusable services that are available online, such as webmail, wikis, online retail sales and instant messaging services. However, the required functionality may vary from person to person according to different needs and activities. This has facilitated the current trend towards an emerging practice called web mashups.

A web mashup is a web application that combines multiple content and functionality from disparate web sources to create new applications or services. Various contents such as XML or HTML formats, RSS or Atom feeds and ShockWave Flash (SWF) are typically displayed on a graphical interface. Functionality elements are typically provided by open APIs. Different from traditional component-based application development, the natural environment of mashup development is the web. Mashups are composed of web techniques and tools to serve user-specific needs. Nowadays, most of the Internet leading companies including Google, Yahoo, IBM, Intel, etc., have invested into mashups, which indicates that the development of mashups is becoming a real hot spot.

2.3.1 Mashup Components

According to Daniel & Matera "a mashup component is any piece of data, application logic, and/or user interface that can be reused and that is accessible either locally or remotely." [12]. This definition is very broad because mashups, as composite applications, are not limited to any specific component technology. The technological openness and heterogeneity makes mashup intricate and complex, but also makes it appealing to practitioners and researchers. All existing mashup components can be grouped into three types depending on the purpose they serve: data components, logic components and user interface components. Mashups can integrate multiple types of mashup components and span across multiple application layers.

2.3.1.1 Data Components

Data components provide access to different data sources. Data sources can be either static such as RSS and Atom feeds, CSV, XML, JSON or other similar data file formats, web scraping, linked data, etc., or they can be dynamic, that is, use one or more variables in a query rather than hard-coded values, such as REST Data Services.

- RSS (Really Simple Syndication) is a type of web feed. It is a standardized content distribution method for website authors to push notifications of frequently updated

information on their website, such as blog posts, press releases and podcasts. RSS is a XML-formatted plain text file that contains a set of elements to describe the updated information. Title, link and description are required. Optional elements include language, copyright, publication date, category, image, etc.

- Atom Syndication Format is another content syndication format similar to the RSS. An Atom feed consists of some metadata such as id, title, updated date and time, author and links, followed by a series of entries such as headlines, full-text articles, excerpts and summaries.

- CSV, XML, JSON or other similar data file formats store data in standardized, human-readable and machine-readable plain text. It is the simplest way to distribute data on the web.

- Web scraping or web data extraction is used for extracting data from websites. Current web scraping solutions range from ad-hoc, which requires human effort, programming skills and knowledge about HTML web page structures, to fully automated web data extraction tools. These tools allow users to interactively select data from any web pages using a point-and-click method, derive regular expressions of the extracted data and re-publish this to their own mashups.

- Linked data is a method to create a network of interlinked data items. Data items are connected to each other via links, and can be easily navigated and found through semantic queries.

2.3.1.2 Logic Components

Logic components provide access to the business logic layer. A logic component serves as an intermediary for data exchange between data access and presentation layers. It requests information from the data access layer, manipulates that information as required, and return the ultimate results to the presentation layer for formatting. For example, SOAP and RESTful web services, JavaScript APIs, device APIs, etc.

- SOAP (Simple Object Access Protocol) is a XML-based communication protocol for exchanging structured information when implementing web services. It is a stateless, one-way message exchange paradigm between a web service and its clients. It enables client applications to connect to remote services and invoke remote methods. It relies on

other application-layer protocols like HTTP or SMTP for the actual message transmission.

- RESTful web services are REST (Representational State Transfer) architecture based web services. In the REST architectural style, resources including data and functionality are accessed via uniform resource identifiers. The representation of resources can be XML files, HTML pages, plain text, PDF, JSON and other document formats. The resources are manipulated using a set of well-defined stateless operations: GET, POST, PUT and DELETE. GET retrieves the current state of a resource in some representation; POST or PUT updates an existing resource or creates a new resource, given a representation enclosed in the request by the client; DELETE deletes the resource identified by the request URI. For example, Twitter provides lots of REST APIs to give access to read and write Twitter data, including posting a new tweet, reading the author's profile and follower data, searching tweets, reading direct messages, etc.

- JavaScript APIs are JavaScript programming interfaces provided by the client browser. They are local to the client, i.e. the web browser, and may require installation. For example, web applications can store data locally within the user's browser using HTML5 Web Storage API; File API are used by the browser to provide secure access to the file system.

- Device APIs are JavaScript APIs that enable web applications to interact with device services on either desktops, laptops, tablets or mobile devices. They provide access to device capabilities that run outside the web browser on the local device, such as calendar, contacts, messaging, camera, geolocation, accelerometer, etc. [12] For example, developers can access to device camera using Media Capture and Streams API, or access to light sensor on the device with Ambient Light API.

2.3.1.3 User Interface Components

A user interface component provide a predefined user interface fetched from a third-party web application. Users can interact with the web application using this user interface. For example, HTML code snippets, JavaScript APIs with built-in UI elements, portlets, widgets, etc. Developers only need to identify a placeholder inside the layout of mashups to use the UI components.

- HTML code snippets are the simplest user interface components. A snippet can be a single HTML element, or an entire web page that can be integrated into another view.

- JavaScript APIs with built-in UI elements provide user interfaces that can be rendered into other applications. For example, Google Maps JavaScript API contains UI elements to allow user interactions with the map. These UI elements are known as controls such as Zoom control, Street View control and Rotate control, and developers can include variations of these controls in their applications.

2.3.2 Consumer Mashups and Enterprise Mashups

Consumer mashups tend to use license-free technologies to help consumers create personalized websites. The key feature distinguishes mashups from other development paradigms (component-oriented and service-oriented paradigms) is the integration of open web services. The most widely used open web services are public programmable APIs such as Facebook, Google Maps, Twitter, YouTube, etc., or RSS/Atom feeds such as BBC News feeds and IEEE Spectrum feeds.

Nowadays, companies use enterprise software systems such as Enterprise Resource Planning system to manage and perform business procedures on a daily basis. All companies have a continuous need of adapting their enterprise systems to new environments, information requirements and business processes. However, business users, as the end users of enterprise systems, are the domain experts of business industry but not professional programmers. Delegating programming tasks to IT professionals is expensive, time-consuming and error-prone caused by potential misinterpretation when communicating about the requirements. So the development of mashups has extended from consumer web to enterprise web.

Enterprise mashups are implemented on enterprise intranets. An enterprise mashup encapsulates enterprise resources, web resources and open web services in a lightweight mashup design paradigm. It enables business users to create small and interactive enterprise widgets based on their personal information assets needs, and deploy these applications directly to their machines without learning any programming knowledge. Enterprise mashups focus more on assembling over development. Employees can assemble a wide range of information from both enterprise internal system and other government agencies using public APIs. It allows them to customize their own dashboards and manage financials, grants, recipients, awards, projects, etc. within a single user interface.

3. Theories

To assist with the mashup development process, and to enable end-users to mash up their own applications, the focus of ongoing research is to develop mashup tools. All existing literature related to mashups can be classified into two categories:

(a) Articles that propose a new mashup architecture, a mashup composition model or a mashup design approach. For example, Liu et al. [13] introduce a mashup architecture, propose a mashup component model and a runtime mechanism to help developers to create their own mashup tools. Similarly, Belleau et al. [14] present a three-step approach that can be used to build a data mashup system in the bioinformatics domain.

(b) Articles that present new mashup tools, describe specific features of the tools, and illustrate how these features can enhance the ability of end-users to build sophisticated mashups by giving a few examples of usage scenarios. For example, Aghaee et al. [15] present a mashup tool called "NaturalMash" and describe some features such as inline search, auto completion, semi-structured text editing, drag-and-drop and error highlighting.

One common drawback of previous literature is the lack of theoretical analysis about the following problems: (a) what are the key factors that drive the success of mashup architectures or tools? And (b) what factors have positive effects on user acceptance and user satisfaction? This paper is mainly devoted to providing a theoretical analysis of all these key factors to help developers build a successful mashup tool.

The success factors are proposed from the viewpoint of end-users who use the mashup tools, as well as the viewpoint of developers who develop the mashup tools. The viewpoint of end-users is about human factors that can be easily understood, directly experienced and evaluated by them. Therefore, research on end-user programming is referenced to support this viewpoint. The viewpoint of developers, on the other hand, discusses human-computer interaction (HCI) factors that focus on the ability of human brain and sensory-perception, in order to help developers, analyze the quality of mashup tools. Therefore, research on cognitive science is referenced to support this viewpoint. The multifaceted view of success factors allows developers to better understand user experience requirements and evaluate the success of mashup tools.

3.1 Viewpoint of End-users

Two major considerations when developing web mashup tools are: (a) how to make end-users use a mashup tool? And (b) how to make them enjoy using the tool? The success of a web mashup tool is achieved by applying a user-centered principle to the whole development process. The user-centered principle focuses on the importance of human characteristics and human behavior, as well as the consideration of what factors will enhance user experience the most.

This paper references and extends the definition of "Usability" to the development of mashup tools. Usability is an important principle for specifying and evaluating a mashup tool from the aspects of end-user performance and satisfaction.

Usability is not a single, one-dimension principle but has multiple dimensions. A standard definition of usability can be found in ISO 9241-11: "The extent to which a product can be used by specified users to achieve specific goals with effectiveness, efficiency, and satisfaction in a specified context of use." Nielsen [16] expands the characteristics of usability in ISO 9241-11 into five dimensions: learnability, efficiency, memorability, errors, and satisfaction. Quesenbery [17] also creates a five-dimension usability model and summarizes it to five E's: effective, efficient, engaging, error tolerant and easy to learn. In addition, Igbaria et al. [18] report that both perceived usefulness and perceived ease of use have direct effects on system usability and personal acceptance. They also show that technical support and training both indirectly affect acceptance.

Some usability dimensions in the above research can be referenced by the definition of usability of web mashup tools, while other dimensions may have less or negligible effects on the success of mashup tools. In general, the usability of web mashup tools is evaluated by five factors: usefulness, ease of use, ease of learning, error tolerance, and satisfaction.

Usefulness and ease of use have the most powerful and direct effects on the usability of mashup tools. It is crucial to enhance these two dimensions of the mashup tools for a better user experience and acceptance. Ease of learning is also an important factor because of the nature of end-users who have no or little experience in software development. Error tolerance of the tools needs to be considered as well because end-users are likely to make errors during the development process. Satisfaction is another factor to be considered. A mashup tool provides a visual programming environment for end-users, so it will ensure a certain level of satisfaction.

3.1.1 Usefulness

Usefulness refers to how completely and accurately a mashup tool can help end-users meet their specific software development requirements. It is measured by the degree to which end-users believe that the mashup tool can enhance their job performance. In other words, end-users should be able to mash up their own application with the mashup tool to achieve all their requirements, desired functionality and features.

It is necessary for a mashup tool to integrate different web resources and web services as much as possible. Because a mashup tool is designed for people from different industries with different requirements. And the requirements can vary dramatically according to the job needs, for example:

- An accountant would like to develop a web application that helps him run a set of algorithms, calculate data and generate PDF files. The mashup tool needs to provide UI components, such as file system, inputs, buttons and tables. Function modules like data calculation, data visualization and file export should be included as well.

- A traveler would like to develop a web application that can show all the hotels and restaurants around him on Google Maps and generates GPS directions. The mashup tool needs to provide UI components like forms, search boxes and tables, function modules like fetching data, filtering and sorting, as well as web services like Google Maps.

3.1.2 Ease of Use

According to ISO 9241, efficiency is defined as “resources spent by user in order to ensure accurate and complete achievement of the goals”. It is measured by the total number of individual actions a user must take or the total time a user must spend to meet a goal. The basic requirement of a high-efficiency mashup tool is ease of use. Davis [19] defines ease of use as “the degree to which a person believes that using a particular system would be free of effort”.

Ease of use of a mashup tool is mainly reflected in three aspects: terminology, visual representation and interface design. Terminology refers to words, phrases and abbreviations used by the tool. Visual representation refers to graphics, images, icons and other visual expressions used in the visual programming process. Proper terminologies and visual representations can help end-users easily understand the functions of each provided components and modules, so that they can perform and develop more efficiently. Interface

design also plays an important role in enhancing the ease of use. A user-friendly interface should provide an explicit navigation design and a well-defined categorization of content. It will reduce a lot of time and efforts that end-users need to spend on making action decisions. The interface should also allow end-users to easily and freely move around.

Furthermore, ease of use is affected by individual differences among all end-users [20]. Individual differences, that is, personality traits, determine the way end-users think and behave under different situations. For example, keyboard shortcuts can be easy to use and improve efficiency for some users who prefer keyboard operations. On the other hand, they can slow down some users who are unfamiliar with the keyboard and prefer to use a mouse to move the cursor. So it is important to think through and understand the individual differences when designing the mashup tool.

3.1.3 Ease of Learning

Ease of learning is the most fundamental usability dimension for mashup tools. The first thing most end-users would do with a new tool is to learn how to use it. The first impression most end-users would have will be the ease of learning of this tool. Ease of learning is defined as how well a mashup tool supports initial orientations, and how well it assists end-users in deepening their understanding of its capabilities [17].

Ease of learning can be enhanced by providing built-in training elements, such as tutorials, instructions, prompts, examples, tips and hints. They are helpful for beginners who need to spend some time on getting familiar with the tool and learning mashup technologies. An easy-to-learn mashup tool should also be predictable. Predictability means that the tool is able to infer the following operations automatically based on some predefined rules. It can place predicted information or controls in where users expect them to be, and act in ways users expect. For example, the mashup tool will highlight all the possible destinations, while users are linking together multiple modules.

However, it may cause a negative effect on usability if a mashup tool places too much weight on the ease of learning. The same problem can be easily handled by end-users when it happens repeatedly. So some training elements can become tedious once the users are familiar with the mashup tool. For example, opening a tutorial window is really helpful for the first use, but it becomes annoying when it pops up every time when the tool is opened.

3.1.4 Error Tolerance

Normally, end-users don't take time to learn the full functionality and operations before starting using the tool. Instead, they tend to jump right in and start using the tool as soon as they have learned part of it. It can lead to a high possibility of making errors. Also, human accidents, misinterpretations and negligence always happen, such as wrong clicks and misspellings. So it is important to enhance the error tolerance of the mashup tools.

Quesenbery [17] defines error tolerance as "How well the design prevents errors, or helps with recovery from those that do occur?" It means that a mashup tool should prevent end-users from making errors during the mashup process, and allow them to easily restore the application when programming error happens. All the actions should be reversible. Catastrophic errors must not occur.

Error tolerance can be enhanced by applying automatic testing techniques, popping up alert messages before the execution of risky commands, showing easy-to-understand error messages, and providing solutions to fix the errors.

3.1.5 Satisfaction

Satisfaction is the most subjective dimension among all five dimensions of usability. It is defined by the degree to which the interface and the interactions make a mashup tool pleasant and satisfying to use. Individual use experience during the whole mashup process is different for everyone. But in general, a mashup tool which is attractive, respectful and helpful is more enjoyable than the one which is ugly, rude and aggravating.

The most direct and effective way to enhance satisfaction of a mashup tool is to create a user-friendly interface and a clean visual layout. Especially for the homepage, front panel and navigation menu, because the first impression carries a lot of weight. Besides, there are some other ways which can make the tool more enjoyable. For example, after end-users finishing mashing up a function module, popping up a congratulation message can create a sense of achievement and satisfaction.

3.2 Viewpoint of Developers

The viewpoint of developers focuses on human-computer interaction (HCI), which is linked to, but somewhat distinct from human factors. Human factors is seen as lacking in a theoretical

motivation of cognitive strategy, while HCI factors comprises a better cognitive coupling between human and computer [21]. It is mainly about how software developers present the mashup tools effectively to end-users in a highly interactive way.

This paper references and extends the definition of "Cognitive Dimensions" into the design of mashup tools. Green [22] defines a cognitive dimension as "a characteristic of the way that information is structured and represented". He introduces five cognitive dimensions including hidden dependencies, viscosity, premature commitment, role-expressiveness and hard mental operations. The list of cognitive dimensions has been gradually expanded. Blackwell et al. [23] describe cognitive dimensions as "providing a vocabulary that can be used by designers when investigating the cognitive implications of their design decisions." They summarize fourteen dimensions including viscosity, visibility, premature commitment, hidden dependencies, role-expressiveness, error-proneness, abstraction, etc. In addition, they publish a few new candidate dimensions such as creative ambiguity, indexing, etc.

Some cognitive dimensions in the above research are important to the design of mashup tools, however, others may have less or negligible effects on the success of mashup tools. For example:

- Premature commitment is not crucial for mashup tools. Because when end-users mash up their web applications with mashup tools, they can design any part of it at any time.

- Role-expressiveness is not important for the mashup tools. Because different components are represented with different graphical expressions. Each component looks very much distinct from each other. The meanings and functionality of them can be easily inferred.

- Visibility is not considered as well. All mashup tools support visual programming and all mashup components are represented graphically.

Besides all the proposed cognitive dimensions in previous research, the scale of "Liveness" with four levels for visual programming system proposed by Tanimoto [24], the concept of "Automation Degree" proposed by Aghaee [25], as well as the concept of "Collaboration" proposed by Fox et al. [26] are all significant HCI factors for the design of mashup tools.

In conclusion, this paper proposes ten HCI factors that developers need to consider in order to design a successful mashup tool. All the ten factors are equally important. They are important to all stages of development process to amplify the interactivity of the mashup tools. Developers

can take these HCI factors as design principles to design and evaluate their mashup tools, to anticipate how a design change will affect the mashup tools, and to analyze the tradeoff between different designs solutions.

3.2.1 Cross-platform

A successful mashup tool should be able to run on a variety of hardware and software platforms, including desktops, laptops, tablets and mobile phones. The cross-platform capability can be represented in a few aspects, for example:

- The interface of a mashup tool is responsive on different browsers.
- The developed mashup applications are responsive on different browsers.
- On mobile devices, a mashup tool can access to other internal components, such as photo library and calendar.
- On touch-based devices, a mashup tool implements touch recognition and tracking. It also supports multi-touch gestures, such as zooming in and out.

3.2.2 Levels of Liveness

A scale of liveness for visual programming systems is proposed in [24], which can be referenced by the mashup tools design. That is, mashup tools can be classified into 4 levels based on the degree to which they present live feedback to the end-users.

At level 1, the mashup tool is used for creating prototype mashups, which means end-users can only design a web interface. The developed application has no actual functionality. It is not directly connected to any kind of runtime system, so it can't be executed.

At level 2, the mashup tool can produce an executable function flowchart which contains enough details so that the application is completely specified. However, the function flowchart needs to be manually fed to another tool, in order to be transformed into an executable mashup application [25].

At level 3, the function flowchart can be deployed into an executable mashup application within the mashup tool. A design change can lead to immediate visual feedback. The problem is that, at this level, it is uncertain to see if the mashup design and runtime environment are in sync with

each other [25]. In other words, the execution needs to be triggered by an explicit action by end-users, such as pressing the “run” button. And whenever the end-users change anything, they need to manually re-execute the mashup application by re-pressing the “run” button.

At level 4, the mashup tool supports stream-driven updates, that is, live updates of the mashup application. The application will be continually activated after it has been executed once. So there is no need for end-users to re-execute the application every time after making any changes. All the changes are instantly observed, and the mashup tool will provide immediate visual feedback to end-users.

3.2.3 Automation Degree

Aghaee [25] defines automation degree as “How much of the development process can be undertaken by the tool on behalf of its end-users?” He classifies automation degree into three levels: fully automated, semi-automated and manual.

Full automation is not suitable for mashup tools, because the mashup tools need direct involvement of end-users in the whole development process. Also, full automation will increase the complexity of the tools and the burden of learning.

Semi automation is the best choice for mashup tools. It can minimize the workload of end-users, reduce the incidence of human errors and highly enhance the efficiency. Semi-automated mashup tools can automatically perform some simple programming tasks. They can simplify the mashup process by providing guidance and assistance to the end-users. For example, a semi-automated mashup tool can analyze the current application design, infer and highlight all the possible options for end-users.

3.2.4 Modularity

Modularity is the degree to which a system is composed of relatively independent modules which can be combined. Baldwin and Clark define a module as “a unit whose structural elements are powerfully connected among themselves and relatively weakly connected to elements in other units. [27]” In other words, modules are units that form a complex system. They are interacted with but structurally independent of one another. The design of mashups is based on principles of modularity. Each mashup components or the combination of multiple components serve as modules that can be combined into mashups and reused in different applications.

The concept of modularity is used primarily to reduce complexity by decomposing an application into self-contained but interconnected modules. The internal complexity of each module is hidden by defining a separate abstraction with a well-defined interface. A well-defined interface separates the specification of a module which indicates how the module interacts with the larger system from its implementation details. This allows the implementation details to be changed as necessary without affecting the external use of the module.

3.2.5 Collaboration Community

Collaboration community is significant to the success of mashup tools. It can be either a private or a public community, such as a central repository or a forum. The community members can be professional developers, end-users, groups or companies. Every community member can upload and share their own mashup applications, modules, functions and templates into the community. All the shared mashups can be reused by other members by copying to their workspaces. Members can reuse the mashups in these ways: fully reuse of the complete mashups, partially reuse of the complete mashups with parametrization, or partially reuse parts of the mashups.

A collaboration community may have members from different industries who have different skills, techniques, expectations and requirements. So it is necessary for a collaboration community to provide some techniques to help members find the best mashups for their requirements. There are several categorization techniques which can be used by the community to help members find their needs when searching for a specific mashup, such as tagging and labeling. In addition, rating is a useful feature used by the community that allows members to evaluate the usability and quality of the shared mashups. It can be represented in different ways, including star-rating, favorites, download counts, etc. Visual previews and comments are also very helpful for community members to reference.

3.2.6 Viscosity

Viscosity refers to the resistance to change. It is measured by how much effort is required to make a single change. For example, spreadsheets have high viscosity, because users may need to change the whole layout in order to insert a new formula cell into it. High viscosity means more dependencies. And the problems may occur when the mashup tools contain more dependencies between modules. A small change can lead to a sequence of adjustments, or even break the application. Mashup tools that implement separation of concerns principle can effectively

reduce viscosity.

3.2.7 Closeness of Mapping

Closeness of mapping indicates the closeness of a representation to a component. It is measured by how well a visual expression of a component represents the meaning of that component, and how closely they match to each other.

3.2.8 Consistency

Consistency means that similar semantics should be expressed in similar ways. Higher consistency increases the usability of the mashup tools. For example, when all the web services are grouped together and represented with similar visual expressions, end-users can easily infer that a component belongs to a web service but not a UI component. On the contrary, the usability of the mashup tools can be compromised if the similar semantics are represented in different ways.

3.2.9 Diffuseness

Diffuseness of mashup tools mainly indicates the verbosity of the user interface. For example, long paragraphs and big icons are verbose because they occupy too much available space within the user interface. All the meaningless, long-winded symbols and graphical entities should be removed to reduce the diffuseness of the mashup tools.

3.2.10 Responsiveness

Responsiveness is defined as “the specific ability of a system or functional unit to complete assigned tasks within a given time [28]”. Long delays have a negative effect on user experience, and can decrease the user's satisfaction. Users may believe that the mashup tool is not functioning, or an action has been ignored. Therefore, responsiveness is considered as a significant HCI factor. Higher level of liveness brings out better responsiveness. It means that the mashup tools can deliver an immediate visual feedback to end-users during the mashup process.

4. Techniques

After analyzing the key success factors of mashup tools development, this section introduces some techniques that can be used during the implementation phase. I present the concepts, definition and benefits of Model-View-ViewModel design pattern. Responsive web design and progressive enhancement process take the above-mentioned usability dimensions and HCI factors into the design of mashup tools. Also, drag-and-drop and WYSIWYG techniques are widely used in modern web applications, and can be benefited in developing mashup tools.

4.1 Model-View-ViewModel (MVVM)

4.1.1 The Evolution of MVVM

At early stage of graphical user interfaces development, Model-View-Controller (MVC) became one of the first design patterns to help developers build their software systems. It is a way of structuring a software system into different components to enhance the system's interactivity and improve user experience. In the MVC paradigm, a View manages the graphical or textual output of data and displays it to the users. Each View has an associated controller, which interprets keyboard and mouse input from the users, and passes the command to the Model. A Model contains both the information presented by the View, and the logic to change this information in response to the command from the controller.

Model-View-Presenter (MVP) is a variation on the MVC pattern. It replaces the controller with a Presenter. The main difference between MVC and MVP is that the Presenter is responsible for not only passing commands to the Model, but also updating the View (Figure 4.1). There is no dependency between the View and the Model. The View and the Presenter have a one-to-one mapping relationship.

Model-View-ViewModel (MVVM) is another variation of MVC. It replaces the controller with a ViewModel. Unlike the MVP, it has a one-to-many mapping relationship between the ViewModel and the Views (Figure 4.1). Multiple Views can hold multiple reference to one ViewModel, but the ViewModel has no information about the Views. The communication between a View and a ViewModel depends on a two-way data binding. If the data in the ViewModel changes, the related data value in the View will change asynchronously, and vice versa.

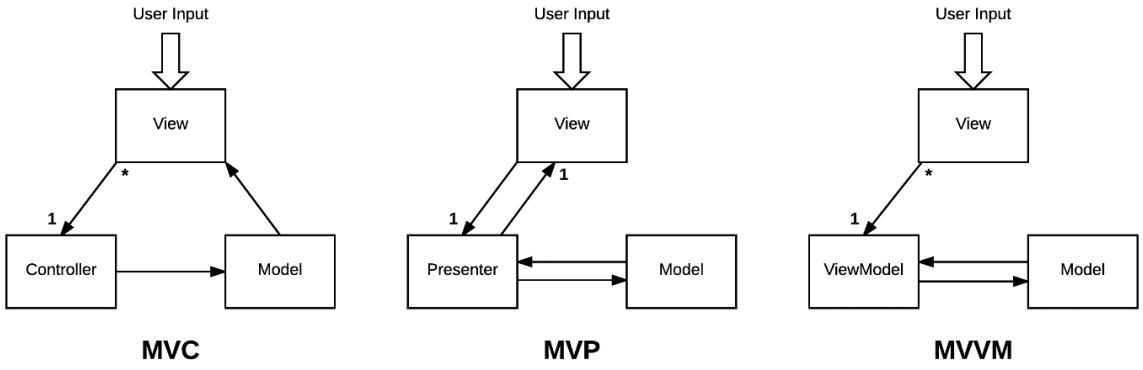

Figure 4.1 Differences between MVC, MVP and MVVM

4.1.2 Basic Concepts

The MVVM pattern belongs to a class of patterns named Separated Presentation. It isolates the user interface from the underlying business logic. It improves the independence and testability of the mashup tools by separating data access, data presentation and data-processing logic into three different components: Model, View and ViewModel.

4.1.2.1 Model

The Model in MVVM is an implementation of the application domain model which represents the business logic. It holds the actual data or information, and encodes real-world business rules into the program. It determines how the data can be created, stored and changed. For example, a model can be a contact record entity containing names, phone numbers and addresses. Given a certain format of an address, a database table can be created which has different columns (street number, city, province and postcode), corresponding exactly to the fields specified in the business logic. However, the Model doesn't contain any behaviors or services that manipulate the information. It is not responsible for formatting contents to look good on the screen, or pulling data from a remote server.

4.1.2.2 View

The View in MVVM is responsible for displaying the information and data to the user and generates events in response to user interactions. Views have a nested structure; a view can also be a sub-component of a parent view. It defines interfaces, structures and layouts of what the

users see on the screen. It consists of different visual elements, such as buttons, forms, graphics, etc.

4.1.2.3 ViewModel

The ViewModel means “Model provided for the View”. It acts as an intermediary between the View and the Model. It contains data-transformers that convert data from Model types into View types. More specifically, the ViewModel invoke methods in the Model classes to retrieve data from the Model. Then it reformats the data through the data-transformers to make the data available to the View.

The ViewModel also provides a set of implementations of commands available to the View and can be triggered by the View. For example, the click event of a button is bound to a command in the ViewModel, when a user clicks the button in the View, the command will be invoked. The View has no knowledge of how this event is handled, or what impact the event will have on the Model. The commands that the ViewModel provides define the functionality to be triggered by the UI controls, but the View determines how that functionality is rendered.

The interaction between the ViewModel and View is achieved through a two-way data binding, whereby changing one side of the binding concomitantly updates the other side. The data binding system also supports input validation, which provides a standardized way of transmitting validation errors from the ViewModel to the View [29].

In addition, the ViewModel is responsible for determining if a user action changes the data in the Model, and updating the data on the Model if needed. For example, if a user inputs a new phone number in a contact information field, and then clicks the update button, the View only inform the ViewModel that this click event occurred. The ViewModel retrieves the new phone number data from the View and then updates the Model.

4.1.3 Benefits of MVVM for Mashup Tools

4.1.3.1 Model View Separation

To achieve separation of concerns principle and reduce viscosity of the mashup tools, it is important to reduce dependencies in the code. The dependency exists as long as one unit of code hold the reference to another. It can be classified into three types: class dependency, interface dependency and method/field dependency. The interdependent relationship in the code makes

it hard to change and maintain, because a small local change in one module can spread into other modules and has unwanted global effects.

Circular dependency is the most problematic situation where two or more modules are directly or indirectly depend on each other to function properly. It is impossible to separate and reuse a single module because of the tight coupling among the mutually dependent modules. The MVVM pattern solves the problem of circular dependency by ensuring that dependencies are properly directed, and the code at higher levels are dependent on the code at lower levels. More specifically, the dependencies between View, ViewModel and Model are unidirectional, and the View and Model are separated by the ViewModel. Figure 4.2 illustrates the dependency direction between three components of MVVM. The dependency in the opposite direction should be completely forbidden.

The View should not call methods directly or bind to data on the Model. All the interaction is achieved through the ViewModel. It gets all the data it needs from the ViewModel. In turn, the ViewModel gets all the data it needs from the Model. It encapsulates the data and methods into interfaces so that the View can implement and use them. Because of the separation between View and Model, MVVM provides a powerful mechanism for UI changing and data changing.

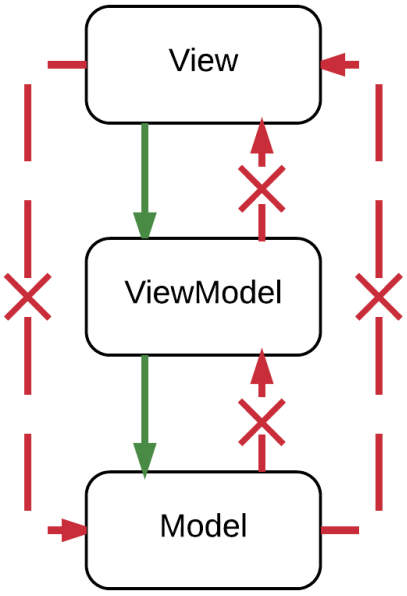

Figure 4.2 Dependency Direction in MVVM

4.1.3.2 Change and Maintenance

As an end-user programming application, the mashup tool tends to change more frequently to improve the user experience. Developers have to redesign the UI of the tool constantly to enhance the ease of use based on user feedback. For example, the visual presentation of an UI element needs to be more appropriate, or a new tutorial window needs to be added. It is also necessary to integrate more UI elements, UI controls, modules and web services into the mashup tool to make it more useful. In addition, user interface programming is device-dependent, that is, the code is different between a browser-based application, an iOS web application and an Android web application. If developers want to provide the capability of cross-platform support to the mashup tool, such as multi-touch recognition and personal digital assistants, they need to change most parts of the UI code.

In this case, tightly coupled and change-resistant code will cause all kinds of maintenance problems that ultimately result in poor user satisfaction. If the presentation class and the business logic class are dependent on each other, developers have to change the code which contains the business logic whenever they make a minor change to the user interface. This may cause errors and require regression testing to all the business logic classes. The MVVM pattern separates the Model and View, which minimizes the risk of causing errors in the business logic, and makes the mashup tool much easier to update and maintain.

4.1.3.3 Efficiency and Productivity

Designing HTML pages and developing complex business logic require different programming knowledge and skill sets. It is more efficient for a team to separate the development efforts of these two parts, by assigning them to designers and developers correspondingly. The MVVM pattern enables a designer-developer independent workflow. The layers of the mashup tools can thus be developed in multiple work streams for higher productivity. During the development process, designers and developers can work concurrently and independently on their components. More specifically, designers can concentrate on the View and create visually appealing UI elements and user-friendly interfaces, while developers can work on creating the robust ViewModel and Model components. The achievements of both teams can be integrated simply by adding references of the ViewModel into the View and adding the correct data bindings.

4.1.3.4 Testability

The MVVM pattern increases the testability of the mashup tools. The two-way data bindings between the View and the ViewModel allow developers to separate the presentation logic from the actual presentation, so they can create unit tests for the ViewModel and Model without using the View. Views and unit tests are two different types of consumers of the ViewModel. The unit tests for the ViewModel can achieve exactly the same functionality as used by the View. In addition, the testability of ViewModel classes can assist in the process of programming. Developers can design the architecture of applications and decide whether something should be in the View or ViewModel by considering the unit test cases. They should be able to write the unit tests for the ViewModel without creating any UI objects, because the ViewModel class should never have any dependencies on any View classes. Also, having a suite of tests for the ViewModel classes allows developers to run the regression testing much easier and faster, which helps reduce the cost of changing and maintaining the mashup tools.

4.2 Responsive Web Design

The interface of the mashup tool may change depends on platforms and browsers. It can be affected by lots of different factors, such as window widths, screen resolutions, user preferences and the system default fonts. For example, when the window size is shrunk, the navigation text can wrap in an unseemly manner, and the gaps between images can become too compact. Rather than targeting a specific version of a specific browser, it is important to consider all these browsing factors when designing the mashup tool.

In the next few years, mobile browsing is expected to outpace desktop browser-based access. New devices with new screen sizes and operating systems are being developed every day. Rather than tailoring interface designs to each mobile device, the mashup tools should also support iOS web browsing and Android web browsing on different sizes of different mobile devices. Ideally, developers should be able to meet the growing demand for a high quality mobile experience without changing a line of code.

To solve the above problems, mashup tool development should implement the concept of Responsive Web Design. It suggests that design and development should respond to users' preferences and behaviors, and also environments such as platform, device, screen size and orientation. Responsive web design makes no assumptions about the width of the browser window. It adapts perfectly to devices that have portrait and landscape modes. When the user

switches from Google Chrome on a laptop to Safari on an iPhone device, the web application should change automatically to accommodate for resolution, layout, image size, etc.

Three key technical features in responsive web design are:

- Fluid grid-based framework that uses relative sizing
- Fluid images and media
- Media types and media queries

By implementing responsive web designs for mashup tools, developers can create a more user-friendly and flexible interface which can adapt to all resolution range. It is also possible to define specific changes when the page is reshaped. The key point is to consider the users' requirements and device capabilities. When designing the tool for mobile devices, developers should not only focus on adapting content to the screen size, but also consider mobile users needs and preferences, and then design the content accordingly. The tool interface may need to change to respond better to a small touch-based environment. For example:

- Collapse navigation content behind a button in mobile screens. Navigation toggle-buttons are left-aligned by default. As the available viewport width increases, navigation bar will also grow to occupy as much horizontal space as possible to keep content securely aligned.
- Select to hide or show some UI elements on mobile environments when the size is shrunk or expanded, to provide a more focused content, simpler navigation, lists or rows instead of multiple columns.
- Increase the target area on buttons and links for smaller screens.
- Gradually expand the size and line space of text to improve users reading experience.

4.2.1 Fluid Grid

Fluid layout, also known as liquid layout, changes its width based on the size of the device screen. A page layout grid is composed of a series of intersecting vertical and horizontal grid lines. It serves as a framework on which developers can organize text content, margins, padding and spacing in the page. By applying fluid grid, page elements can be sized to relative units like

percentages which refers to the portion of the viewport it takes up, rather than absolute units like points or pixels. When a fluid grid changes in size, all the content wrapped within it needs to shift around on the page. This flexibility prevents horizontal scrollbar from showing and makes full use of the screen available on different devices [30]. For devices with smaller viewports, fluid grid reduces the possibility that users may miss some content hidden by the horizontal scrollbar. For devices with large viewports, it decreases the needs of vertical scrolling. Users can view more content on the page at once.

4.2.2 Fluid Image

Another challenge for a responsive mashup tool is about the UI elements and their visual presentations provided by the tool, such as images, thumbnails, graphics, icons, videos and other embedded media. If the image or media is much wider than its container, it will overflow its container and overlap with other components in the page. The key solution is to apply constraints to all images and media to make them responsive. Write rules to prevent them from exceeding the width of their containers.

The most basic rule is the “max-width” constraint, i.e. “max-width: 100%”. This rule provides a straightforward constraint for every image and media in the page. If the width of an image is narrower than its container element, it will display its actual size. If it is wider than its container element, then this constraint will force the width of the image to match the width of its container. Also, when a user shrinks or expands the container, the image will resize proportionally, and its aspect ratio remains unchanged. The “max-width” is not supported in Internet Explorer, but “width: 100%” would solve the same problem in an IE-specific CSS.

Another primary consideration for mobile web browsing are image resolution and download times. If the original image size is designed for large devices, it can significantly slow download time and take up unnecessary space. Filament Group [31] has developed a technique to solve this issue, which is fully supported in modern browsers including IE, Safari, Chrome and Opera. It uses HTML to reference both higher and lower resolution images, and writes the logic in JavaScript to adjust images based on the screen width. When the page loads, it will automatically load the high resolution or low resolution image as necessary. Especially for mashup tools with a lot of images, this technique is great for reducing bandwidth and loading time.

4.2.3 Media Queries

One of the most important features of a stylesheet is that it can specify how to be presented on different media. Certain CSS properties are only designed for certain media. The CSS specification provides a list of CSS media types, each media type targets a specific class of web device. For example, “all” for all devices, “print” for paged material and for documents viewed on screen in print preview mode, “screen” for color computer screens, “handheld” for handheld devices with small screens and limited bandwidth, etc. [32]

However, most browsers and mobile devices only implement the media types partially, or in different ways, or completely ignore this features. W3C developed CSS media queries to improve upon the media types. It is currently implemented by the recent versions of Safari, Opera, IEMobile, etc. “A media query consists of a media type and zero or more expressions that check for the conditions of particular media features.” [33] Media features are similar to CSS properties and are used to describe requirements of the output devices. Developers can combine multiple queries by using semantic operators such as AND or NOT. Each media feature has a name and a value, for example:

- Width: It describes the width of targeted display area of the output device. It refers to the width of the viewport.
- Device-width: It describes the width of a rendering surface of the output device. It refers to the width of the screen.
- Orientation: It describes the orientation mode of the output device. It can either be portrait or landscape.
- Aspect-ratio: It is defined as the ratio of the width media feature to the height media feature.
- Resolution: It describes the resolution of the output device, i.e. the density of the pixels.

4.3 Progressive Enhancement

To develop a successful web-based mashup tool, it is important to make sure that the application is universally accessible and usable for everyone on every web-enabled device. It should also reserve the visual appeal and interactive capabilities provided by JavaScript, CSS,

Ajax and plugins. “Designing with Progressive Enhancement is a practical guide to web design and development that focuses specifically on how to create sites that deliver the highly interactive experiences that JavaScript, advanced CSS, and Ajax afford, and at the same time ensure that the very same codebase will work everywhere. [34]” In other words, Progressive enhancement is a technique of creating cross-browser compatible web applications that enables developers to prioritize on core features and functionality, while the other complex add-on features remain secondary. The key to progressive enhancement is to separate content (HTML), presentation and styles (CSS), and behavior (JavaScript, Ajax, Plugins). The progressive enhancement process goes through four phases:

(1) Define content hierarchy, priorities and groupings. Design all user interface based on well-structured, rich semantic HTML to provide a functioning basic experience supported in any browser. Build basic markup to provide all essential content and functionality with minimal styles. Rich semantic HTML markup allows text-based, speech-based, robotic and antiquated user agents to view all content on the web page. Don’t consider JavaScript at this phase.

(2) Test CSS and JavaScript capabilities of different user agents and plan for the enhancements.

(3) Apply advanced CSS-based enhancements in user agents that can support them to achieve visual enhancements. This will make the web page more attractive and enhance the user experience with images, backgrounds, colors, gradients, shadows, text effects, fonts, 2D and 3D transforms, animations and a lot more visual elements.

(4) Apply advanced JavaScript-based enhancements in user agents that can support them to achieve behavior enhancements. It focuses more on the performance of the web design using unobtrusive JavaScript or jQuery, which provide the end users with maximum usability and an optimum user experience.

4.3.1 Testing Browser Capabilities

When CSS and JavaScript enhancements are not or only partially supported in certain browsers, it can cause scrambled layouts and JavaScript errors. For example, some browsers can’t properly execute modern JavaScript features with new functions, new objects, additions to current object, new expressions and operators, new statements, etc. Some browsers fail to

render CSS capabilities such as flexible box layout, grid layout, filter effects, 2D transforms, etc. It is important to write a complete testing suite to check for cross-browser capabilities, and then apply enhancements only for browsers that pass all of the tests. More specifically, when a browser passes the entire testing suite, the test then automatically loads the advanced scripts and style sheets that transform the basic HTML markup into the enhanced experience.

Developers can use object detection technique to test a browser's JavaScript support. In the test file, ask the browser whether it recognizes native objects (method, array or property), use an if statement and return a true or false statement. The correct way to check a method is to ask for the method without brackets. For example:

- if (document.getElementById)
- if (window.focus)
- if (document.images)

It is more complicated to determine whether a browser can correctly render advanced CSS enhancements. To write a CSS test, first inject invisible HTML elements into the page using JavaScript, then apply a specific set of advanced CSS rules, and finally run a JavaScript function to check whether the element is rendered properly. For example, to test if the browser supports CSS 2D transforms methods, the test moves an element from its current position using the CSS translate() method, and then runs a script to see if the measured coordinates match the expected coordinates.

4.3.2 Benefits of Progressive Enhancement for Mashup Tools

4.3.2.1 Accessibility

Accessibility is the most important reason for the separation of the markup, styling and behavioral layers. It makes sure all user agents can render the content on the web page, including modern browsers, antiquated browsers, mobile devices, search engine spiders, screen readers (convert text that is displayed on the computer screen into a form that blind or visually impaired users can process), etc. For example, if there is any hidden content that will display on the page after user click on a button element, it listens to a user event with JavaScript, and it must still be accessible when JavaScript is disabled.

4.3.2.2 Portability

Progressive enhancement provides cross-browser and cross-platform support. For mobile devices that support CSS3, HTML5 and media queries, all the enhanced features will be available to users which can provide better user experience.

4.3.2.3 Modularity

Separate the implementation of web applications into different layers makes the application easier to maintain. If changes need to be made in the presentational layer, markup and behavioral layer will not be affected. Also, it allows front-end developers, back-end developers and web designers to work together and focus on their main tasks a large project.

4.4 Drag-and-Drop

Drag and drop is a prominent technique in modern web applications. It is a key capability in any graphical user interface. This process contains dragging a component and dropping it into another component. It consists of four elements:

- Drag source: the component which is being moved.
- Drag initiator: the component which the drag source is dragging from.
- Drag proxy: the visual presentation of the drag source which is displayed on the screen during the dragging process.
- Drop target: the component which the drag source is dropping into.

By using this technique, end users can do complex operations simply by visually dragging and dropping UI components from one location to another. The drag-and-drop behavior of a drag source can be split into five steps:

- Set property of the drag source so that it can be dragged around.
- Place mouse on the drag source, click on it and hold the mouse button down, move the mouse around.
- Release the mouse button after the dragged source arrives at a specific position.

- Define an action that the drag source will perform after it appears in that specific position. For example, drop a file into a specific field in a page will upload the file automatically.

- If the drag source is dropped at a wrong position, recover the drag source to its original location.

4.5 WYSIWYG

In general, a WYSIWYG (What You See Is What You Get) editor is a system that allows users to edit text and graphics in a form that closely resembling the visual presentation of the finished product, such as a web page, or a printed document. This technique reduces the prerequisite knowledge of HTML. For the mashup tools, WYSIWYG refers to the ability of displaying visual appearance of the finished web applications to the end users. It increases the liveness level of mashup tools by providing direct manipulation visual feedback to end users. A mashup tool can provide multiple WYSIWYG modes, for example:

- Layout mode: what end user sees is similar to the finished user interface, but with additional layout information, such as borders, margins lines, grid lines, etc.

- Preview mode: what user sees is exactly the same as the finished user interface.

5. Implementation

The mashup tool has four main sections: navigation, component panel containing all mashup elements, mashup canvas, and custom panel containing all customizable properties. This section presents several implementation details to show how the techniques introduced in Section 4 (MVVM pattern, responsive web design, progressive enhancement, drag-and-drop and WYSIWYG) are immersed into the mashup tool development process.

5.1 Vue.js Framework

The mashup tool is written in JavaScript using the Vue framework, and the PHP Laravel framework as its backend. The core library of Vue is focused on building user interfaces for single page web applications. Compared to other frameworks, Vue has more significant advantages for developing the mashup tool. This section introduces four main advantages:

single file components, supports massive libraries, implements MVVM pattern and high performance.

5.1.1 Single File Components

Vue projects define single file components with a .vue extension. They can be built by different build systems such as Webpack or Browserify. Single file components apply the Separation of Concerns design principle. Each component has 3 parts:

- <template></template> where defines HTML templates
- <script></script> where defines JavaScript logic
- <style></style> where defines CSS styles

To provide all of the necessary functionality to end users to build their own web applications, a mashup tool needs to contain a large amount of mashup components. All of these mashup components are created as separate Vue components. The templates, logic and styles of each component are defined within a single file. It is helpful to achieve high cohesion and low coupling relationships between the components. By creating loosely-coupled components, the mashup tool becomes less viscosity and more maintainable.

5.1.2 Supporting Libraries

There are a great number of Vue.js libraries. Vue.Draggable is the most essential library used in the mashup tool. It fully supports all Sortable.js features and supports touch-based devices. Here is a list of libraries supported by Vue and can be used by the mashup tool:

- Frameworks: responsive framework, UI framework for mobile devices, component collections.
- UI Layout: grid layout, draggable layout.
- UI Components: table, notification, loader, progress bar, tooltip, modal, alert, dialog, popup, navigation menu, tabs and pills, breadcrumbs, form, select, checkbox, switch, carousel, charts, calendar, date and time picker, color picker, maps, audio player, video player, text editor, image editor, file upload, Font Awesome icons, auto-complete/auto-

suggest, drag and drop, scroll, pull-to-refresh, markdown, pdf viewer, tree, social networks link sharing, QR code, search, avatar, keyboard/typewriter simulator.

- UI-related Utilities: event handling of user events (shortcut, scroll, click, etc.), input validation, form validation, routing, lazyload images, pagination, animation, filters.
- Non-UI-related Utilities: internationalization, localization, translation, local storage management, authentication, utilities for building/compiling/bundling/loading assets, plugin for making web requests and handling HTTP responses, plugin for mocking HTTP responses, plugin for async data loading.
- Integrations: YouTube embed, "Add to Calendar" functionality, plugin for Google Analytics, debugging tool.

5.1.3 MVVM in Vue.js

Vue.js is focused on the ViewModel layer of MVVM pattern. It connects the View and the Model through a two-way data binding. Figure 5.1 shows a high-level overview of the structure of Vue.js to illustrate how it implements the MVVM pattern.

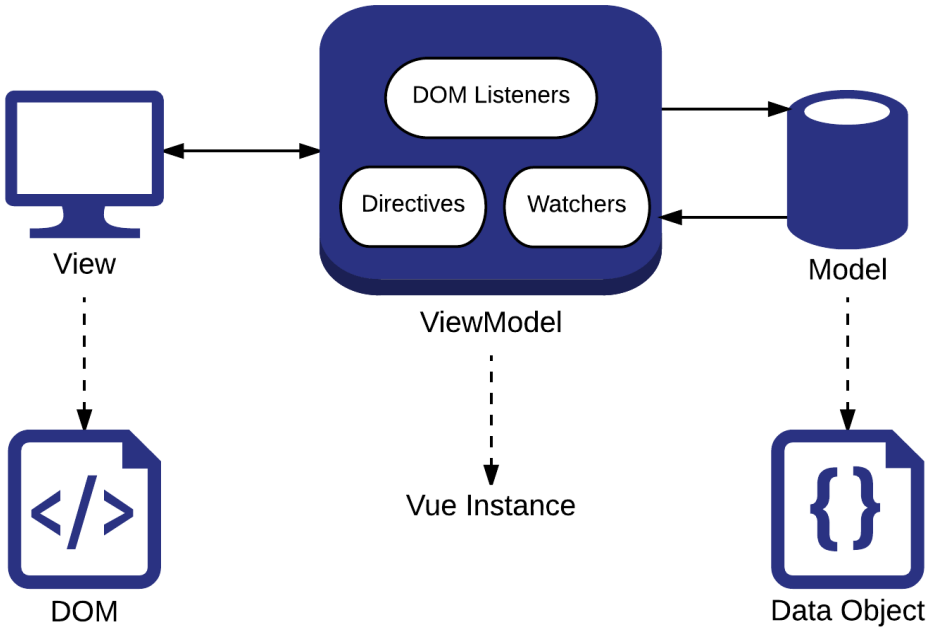

Figure 5.1 Implementation Overview of MVVM pattern in Vue.js

Vue instance represents the ViewModel. It is instantiated with a Vue constructor. Each mashup element has a corresponding Vue instance. When the Vue instance is created, it builds two-way data bindings with the root DOM element and all its child nodes recursively. Models in Vue.js are data objects, or plain JavaScript objects that are defined as data inside the Vue instance. Each data object has multiple properties. For example, the mashup canvas is a Vue instance and the size of the canvas is an object property, a drop-down is a Vue instance and the number of drop-down items is an object property, a header is a Vue instance and the font of header is an object property.

As shown in Figure 5.2, Vue converts all properties of the data objects to getters and setters using Object.defineProperty. Each Vue instance has a corresponding watcher instance that can observe and react to data changes on the Vue instance. So when users manipulate the data properties, the Vue instance will be notified of the changes. Vue performs DOM updates asynchronously. It buffers all the data changes that happen in the same event loop into a queue, and performs the updates in the next event loop. If the same watcher is triggered multiple times, it will only be pushed into the queue once. This deduplication technology eliminates unnecessary calculations and DOM manipulations, and highly improves the performance of the mashup tool.

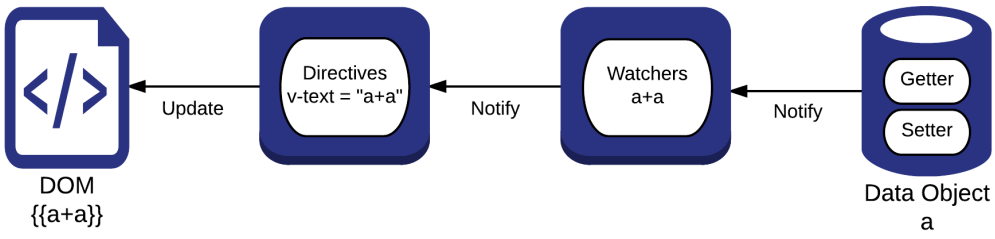

Figure 5.2 Reactivity in Vue.js

5.1.4 Performance

Compare to other JavaScript frameworks, Vue has better performance. The size of a full-featured Vue.js (latest stable version: 2.6.10) is 33.3KB (minified and gzipped), which is significantly lighter than other frameworks.

Vue implements a lighter-weight Virtual DOM. The Document Object Model (DOM) is a language-independent and cross-platform API. Each element in HTML is a node object

representing a part of the document. These node objects are stored as a tree structure. The DOM contains methods that allow developers to interact and manipulate the HTML elements. For example, `getElementById` is used for locating elements, `removeChild` is used for removing elements. Every time developers want to change the page using these methods, browsers need to find the required DOM nodes to make the change. However, there can be thousands of nodes in a HTML DOM, especially for single page web applications, which makes the page updates computationally expensive. It is inefficient, hard to manage, and will slow the page down.

The Virtual DOM is an abstraction of the DOM in a JavaScript data structure. It is a lightweight copy of the DOM. It enables developers to change the JavaScript objects, rather than directly calling the DOM with methods like `getElementById`. Only necessary elements in the DOM will be updated, which significantly enhances front-end performance.

In 2016, Krause published several articles on benchmarking JavaScript front-end frameworks [35]. He compared the performance of a few popular JavaScript frameworks, including Angular.js, React.js, Vue.js, etc. He used the Selenium WebDriver to parse the performance log entries from Chrome's timeline. This measured the duration from the start of a user event to the end of DOM rendering and repainting.

Krause published the results of the latest updates of JavaScript web frameworks benchmark (round 6) in 2017 [36]. There are 12 benchmarks in total that were being measured, including:

- Start-up time: duration for loading, parsing and starting up
- Create 1000 rows: duration for creating a table of 1000 rows
- Replace rows: duration for updating 1000 rows
- Partial update: duration for updating the text of every 10 rows
- Select a row: duration for clicking on the row
- Swap rows: duration for swapping 2 rows
- Remove a row: duration for removing a row
- Create 10000 rows: duration for creating a table of 10000 rows
- Append rows: duration for appending 1000 rows on the table of 10000 rows.

- Clear rows: duration for clearing the table of 10000 rows

To reduce sampling errors, he ran each benchmark on each framework for 20 times and calculated the average results. The results indicate that, compared to all other popular JavaScript frameworks, Vue.js 2.0 has considerably higher performance. It is the fastest JavaScript framework among Angular.js, Angular 2, React.js, Reactive.js and Ember.js.

5.2 Grid System

Fluid grid layout is one of the key features in responsive web design. To build a responsive, cross-platform fluid grid layout, the mashup tool is developed with the Bootstrap grid system. It also allows end-users to mashup their applications with grid system components. More specifically, the Bootstrap grid system is built with the CSS3 Flexible Box. It supports the latest, stable releases of all major browsers and platforms. It uses a series of rows and columns to divide the large content area into smaller full-width containers. Rows must be placed into a container first for proper alignment, margins and padding. There is no limitation on the amount of rows within each container. Then groups of columns are placed within the rows. And page content is placed within the columns. It can scale up to 12 columns as the browser or screen size increases. End-users don't need to know the exact dimensions needed for columns to fill the available space. They can simply create a group of columns with proportional dimensions and drop it into the mashup canvas. Bootstrap grid system also provides some predefined grid classes that enable the mashup tool to display in different layouts across multiple device platforms. It divides all device platforms into 5 types based on the browser width:

- Extra-small device: browser width is less than 576px, e.g., portrait mobile phones
- Small device: browser width is larger than 576px, e.g., landscape mobile phones
- Medium device: browser width is larger than 768px, e.g., tablets
- Large device: browser width is larger than 992px, e.g., laptops and desktops
- Extra-large device: browser width is larger than 1200px, e.g., large laptops and desktops

On extra-large and large devices, the mashup tool displays navigation, component panel, mashup canvas and custom panel on the page. And the ratio between component panel, mashup

canvas and custom panel is 1:4:1. On medium, small and extra-small devices, the custom panel is collapsed, and the ratio between component panel and mashup canvas is 1:5.

5.3 Navigation

Designing navigation to support a large variety of platforms and browsers is one of the most challenging tasks in responsive web design. Two concepts should be considered comprehensively: prioritization and content parity. Prioritization is necessary to avoid a cluttered and inefficient interface. Content parity becomes an established paradigm for delivering content to users. It emphasizes that it is essential to provide all the content and functionality on any screen, even though the priorities among them are clear. More specifically, although all the navigation options don't need to be viewable in the page all the time, they can't be removed, and should remain available on every device and every screen.

The Off-canvas pattern is the simplest and safest strategy to implement the content parity concept when designing the navigation. Navigation options hide in narrow screens, but remain accessible by clicking or tapping on an icon, a "Menu" label, or an "All" label. However, instead of hiding all important navigation options in narrow screens, it is necessary to make full use of the space to show as many options as possible.

To design the mashup tool navigation, I first listed all of the required content and functionality for the mashup tool, then organized and grouped them together, and prioritized them in the navigation based on users' needs. When the width of the screen is less than 768px, the navigation collapses into its vertical mobile view and shows a hamburger icon. An off-canvas navigation menu should slide in from the right side of the screen when a user clicks or tabs the hamburger icon. When the width of the screen is equal or larger than 768px, it expands into its horizontal non-mobile view and shows a series of drop-downs and button groups.

Also, the mashup tool provides responsive Bootstrap navigation components to end-users to design their web applications. Bootstrap is a front-end framework for developing responsive, web-oriented and mobile-friendly applications. Two types of navigation components are available in Bootstrap: Navs and Navbar. The Nav class is relatively simple. It can be further divided into Tabs and Pills. Users are able to edit component properties and make the navigation options vertically stacked or horizontally justified. The Navbar components can provide more content and functionality. Users are able to add brand images, search forms,

buttons, drop-downs and links to the Navbar. They can also align the components to the left or right, or fix the entire Navbar to the top or bottom of the page.

5.4 Component Panel

This tool provides a wide variety of elements for end-users to build the mashup. How to design a user-friendly panel to contain all the elements? How to enable end-users to quickly and easily locate the right element? The simplest strategy is to break down one large section into multiple separate small sections. All the elements are classified into three types: UI layouts, UI components and web services. They are displayed as three separate sections in the component panel rather than one large block. Each of the sections has a clear label. Users can expand or collapse the section by clicking on or tapping on the label, to quickly switch between different categories. Another strategy is visual representation. Each mashup element displays a terminology and a visual icon. In this case, even if end-users don't understand the terminology like thumbnails and media objects, they can still know what the element represents and what the element looks like in the web page.

5.5 Responsive Components

All UI components listed in the component panel are built with the responsive Bootstrap framework. End-users can drag and drop any component from the component panel into the grid system, or from one grid system to another grid system, or within one grid system to change the order. Different components have different responsive behavior across multiple platforms. This section lists some examples and illustrates that, as the screen size changes, how these responsive components change accordingly.

Table content has internal relationships that are manifested by their position and their row and column headings. The content may be hard to read on narrow screens because tabular data requires enough space to be fully displayed. It is likely to be truncated to the size of screen, or rendered too small for comfortable reading. The mashup tool provides responsive table components that enable users to scroll or slide horizontally on small devices.

Charts are built with Chartist.js which is a responsive charting library. In responsive web design, charts need to change the size as the width of the browser reduces to scale and adapt to smaller media queries. Rather than specifying a fixed width and height, end-users can select a certain aspect ratio to initialize the chart. The tool gives a list of chart ratios, such as 1:1, 2:3, 3:4, 9:16,

etc. In addition, there are three types of charts available: bar chart, line chart and pie chart. Currently, the tool only allows users to create a chart with one set of data. But in the future development, it is possible to enable users to draw charts with multiple sets of data and show the visual results of comparison. Animation is another functionality to be developed. With Chartist.Svg animation API, it is possible to offer a variety of animation effect options, such as fly in, fade out, swivel, bounce, etc.

Form is another essential UI component in any web applications. The mashup tool allows users to design forms with different input elements, including labels, inputs, textarea, buttons, selects, stacked checkboxes, inline checkboxes, stacked radio buttons and inline radio buttons. Checkboxes are used for selecting one or multiple options in a list, while radios are used for selecting one option from many. Both the form size and the location of input elements can adapt to the width of the screen automatically. Inline forms will adjust to horizontal forms when the screen width is less than 768px.

5.6 Custom Panel and Toolbars

The mashup tool allows end-users to customize CSS properties of all mashup components. It provides full customization options in the custom panel, such as color, background, opacity, border, margin, size, link url, visibility, etc. Also, for each mashup component that has been added to the mashup canvas, a toolbar should show next to it. It contains all the most frequently used customization CSS properties in an easily accessible way.

Users can either select predefined classes to override the default styles, or they can build their own styles and add to the components. For example, button components have six predefined classes with different background colors: primary button, secondary button, success button, info button, warning button and danger button. Users can either select one from the list, or set the background color with a color picker. Both custom panel and toolbars implement the WYSIWYG principle; that is, when editing CSS rules in the custom panel, all changes will be automatically applied to the components and shown in the mashup canvas.

5.7 WYSIWYG Text Editor

Because end-users are more familiar with text editing interfaces similar to Microsoft Word and Notepad, the WYSIWYG text editor can improve the ease of use and the ease of learning of the mashup tool. It automatically transforms rich text content into HTML code and displays on the

webpage. The mashup tool integrates the CKEditor. Users can also edit the mashup components via the CKEditor user interface, as shown in Figure 5.3. It provides a lot of functionality, including text formatting, undo and redo, copy and paste, and spell check.

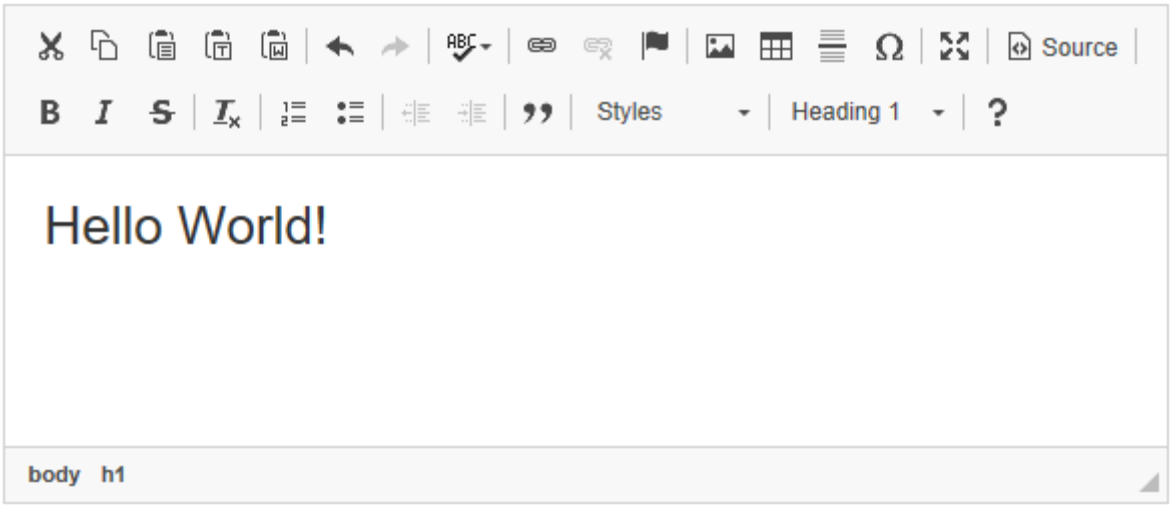

Figure 5.3 CKEditor Interface

CKEditor gives full support to all the latest stable release of IE, Edge, Firefox, Chrome, Safari and Opera browsers on desktop environment, and gives close to full support to IE, Safari and Chrome browsers on mobile environment. In addition, CKEditor is translated into over 60 languages, which enables the mashup tool to provide multilingual support. It is displayed in the user's language as set in the browser or operating system by default.

5.8 Autosave

Autosave means that when users add or edit the content, the application automatically saves user inputs at fixed intervals, or at events of interest such as moving the cursor to the next field. It effectively prevents accidental data loss that is caused by browser crashes, power outages, internet connection breakdowns, or human errors. The mashup tool implements autosave functionality by storing a copy of the mashup application in a temporary file directory every five minutes. It enhances the error tolerance of the mashup tool. An indicator will show at the top of the page, providing immediate visual feedback when the autosave is performed successfully. Although users don't need to explicitly click on or tap on the save button, it is still necessary to keep the save button on the page. It ensures that users have full control of the mashup tool, and they can save their changes at any time.

5.9 Modernizr

To better implement progressive enhancement, I use Modernizr for browser capability testing. Modernizr is a JavaScript library with a collection of tests that run as the web page loads. It checks what HTML, CSS and JavaScript features the user's browser can offer.

To use Modernizr with CSS: Modernizr set HTML class attribute on the root element (<html>). It adds the class for each feature the browser supports, and adds the class with a "no-" prefix when the browser doesn't support it. For example, to detect whether or not elements can be animated using CSS, it will result in either <html class="cssanimations"> or <html class="no-cssanimations"> depending on the browser. And the corresponding CSS should cover both cases:

```
.cssanimations .div { background-color: red; animation-name: example; }
.no-cssanimations .div { background-color: red; }
```

To use Modernizr with JavaScript: Modernizr creates a global Modernizr object. It adds a corresponding property to the object for each test, and can be used like this:

```
<script>
        if (Modernizr.cssanimations) {
                // support
        } else {
                // not support
        }
</script>
```

6. Case Study

This section presents a practical use case of the mashup tool. It is used to build a homepage and an events page of the University of Alberta website. Page transitions are set via proper navigation and links.

Step 1: Create a new project and a new web page

The first step to create a web application is to initialize and name a new project (See Figure 6.1). It will create an empty folder on the server side. Then users can create multiple pages under the project (See Figure 6.2). Corresponding HTML, JavaScript and CSS files will be created in the folder project for each page.

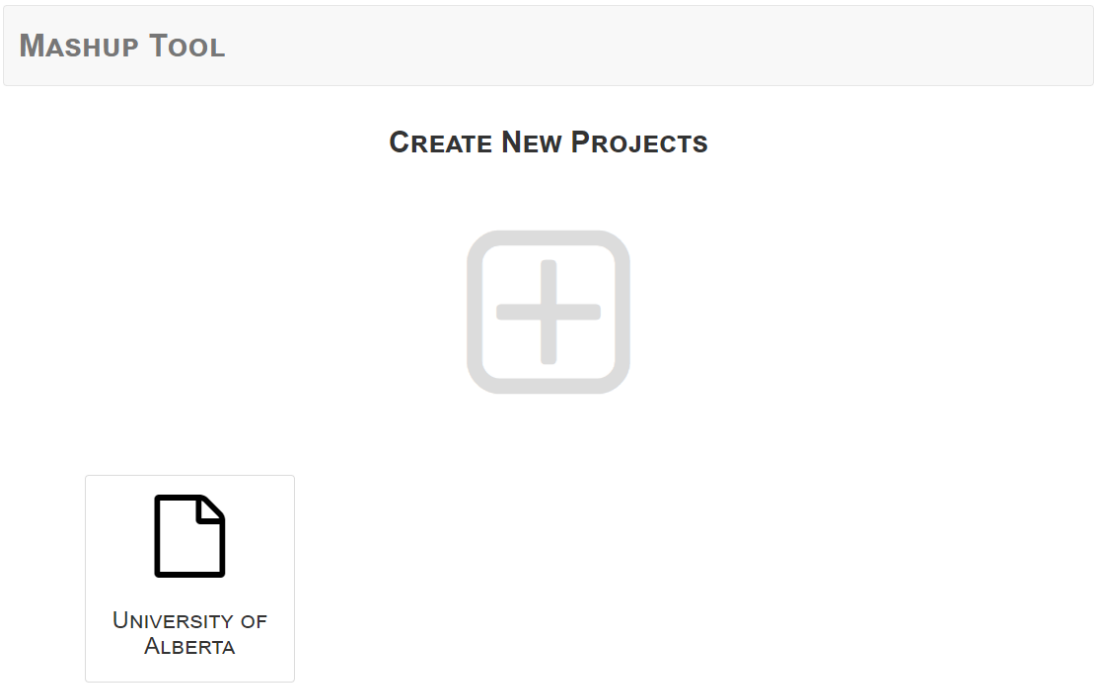

Figure 6.1 Create A New Project

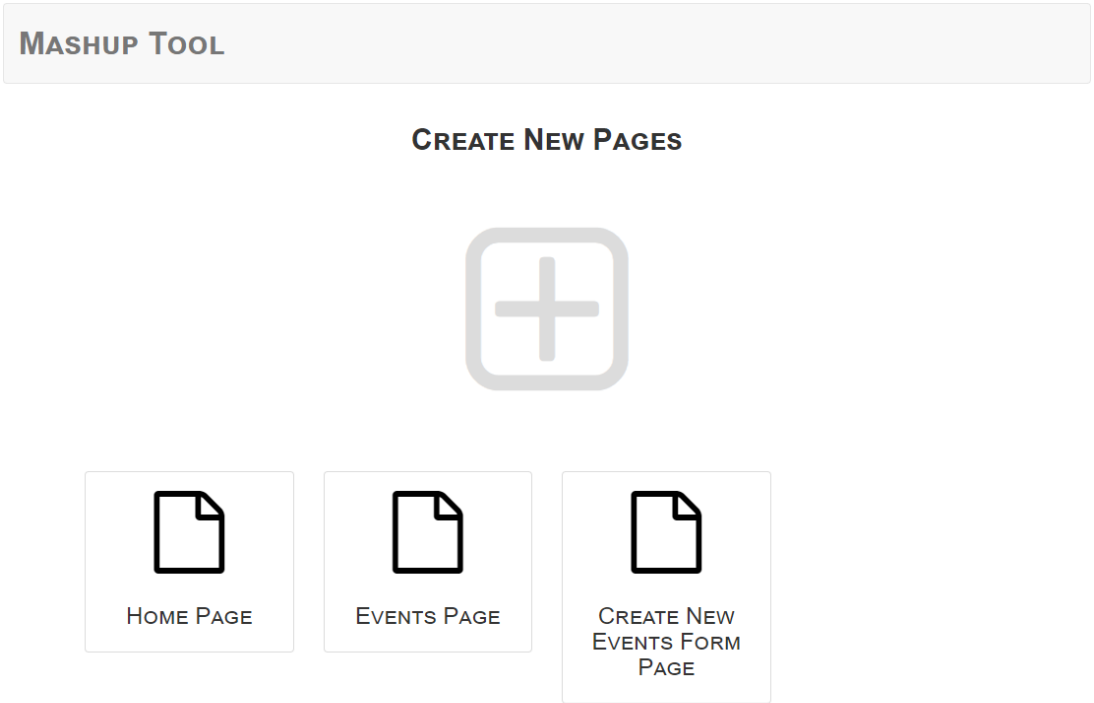

Figure 6.2 Create New Pages

Once page is created, users will enter the main interface of the mashup tool (See Figure 6.3). There are four sections: navigation, component panel, mashup canvas and customization panel. To ensure cross-platform compatibility, canvas can be resized to fit different devices from smartphones, tablets to computers and widescreen computers.

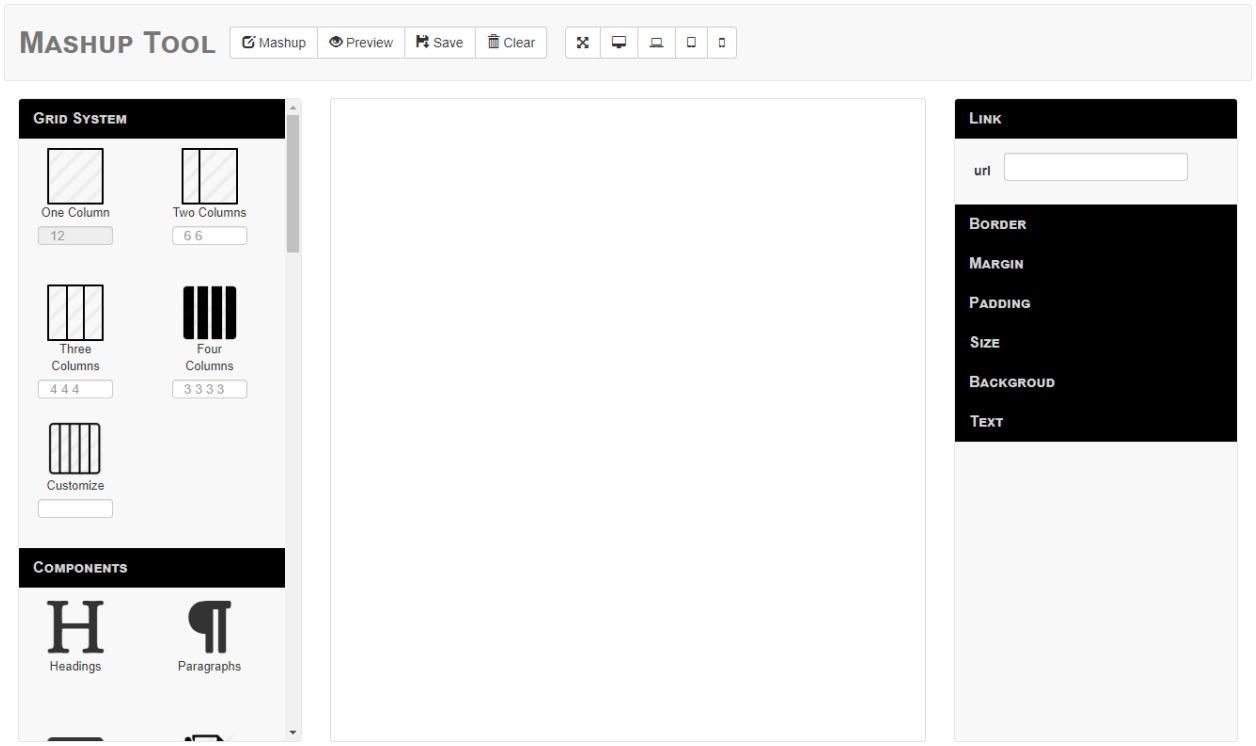

Figure 6.3 Mashup Tool Homepage

Step 2: Design the web page layout

The second step is to design the webpage layout by dragging and dropping layout components into the mashup canvas (See Figure 6.4). The mashup tool implements Bootstrap grid system that uses a series of containers, rows and columns to build responsive layouts and align content. Users can place multiple child containers into a grid column of the parent container to build more complex layouts.

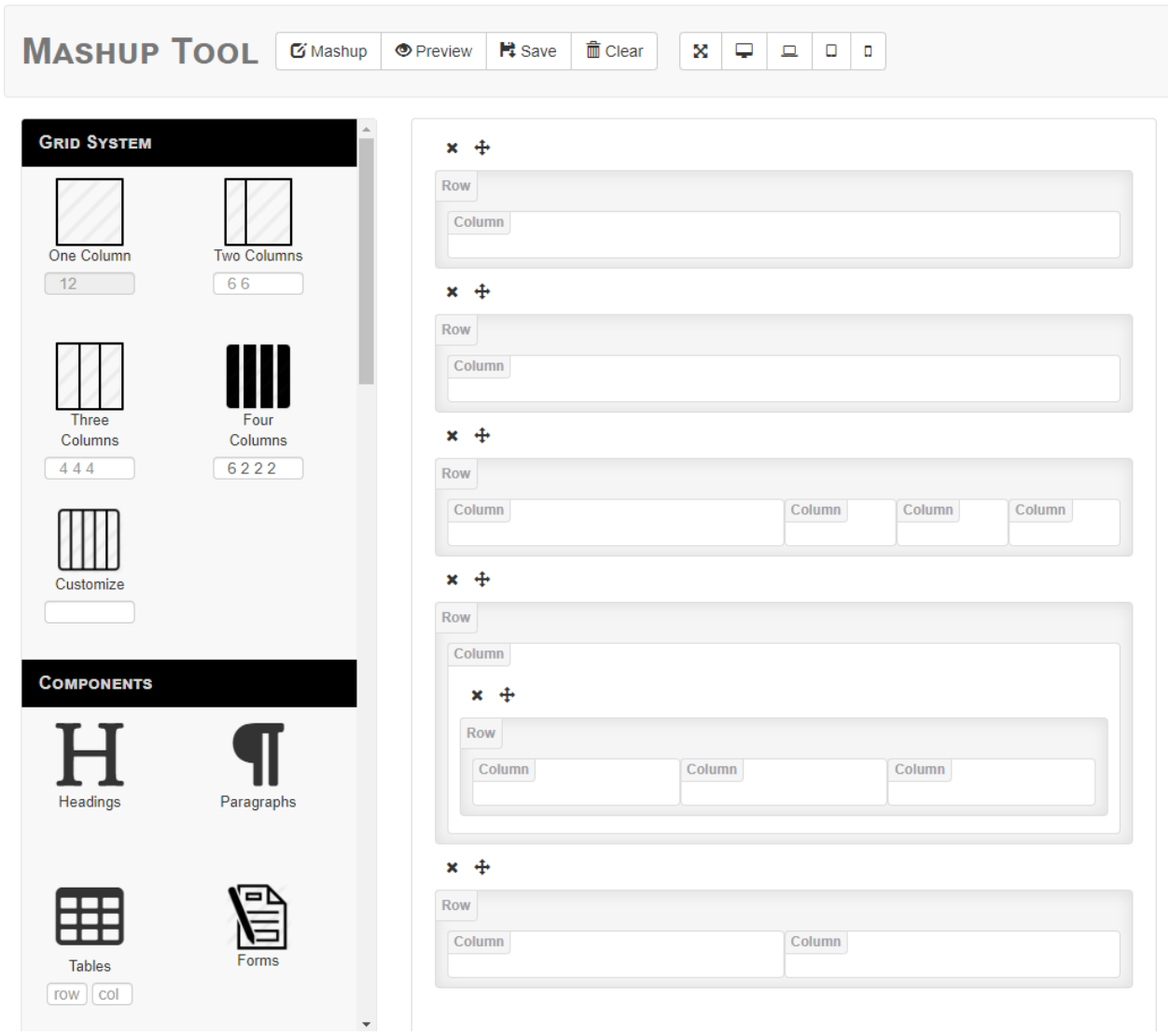

Figure 6.4 Design Page Layout with Bootstrap Grid System

Step 3: Drag-and-drop mashup components into the layout

The third step is to drag-and-drop mashup components into the grid layout (See Figure 6.5). The mashup tool provides massive pre-defined Bootstrap UI components. Following web elements are applied in this use case: navigation bar, tab navigation bar, jumbotron, headings, paragraphs, images, icons, thumbnails, list groups, links and buttons. All text elements can be edited using a WYSIWYG text editor. Users can switch from Edit mode to Preview mode that removes all grid system boxes, utility tools and customization toolbars on the mashup canvas, and shows an exact representation of how the webpage will look on a real device (See Figure 6.6).

It allows end users to validate that the webpage looks as intended across different platforms and devices.

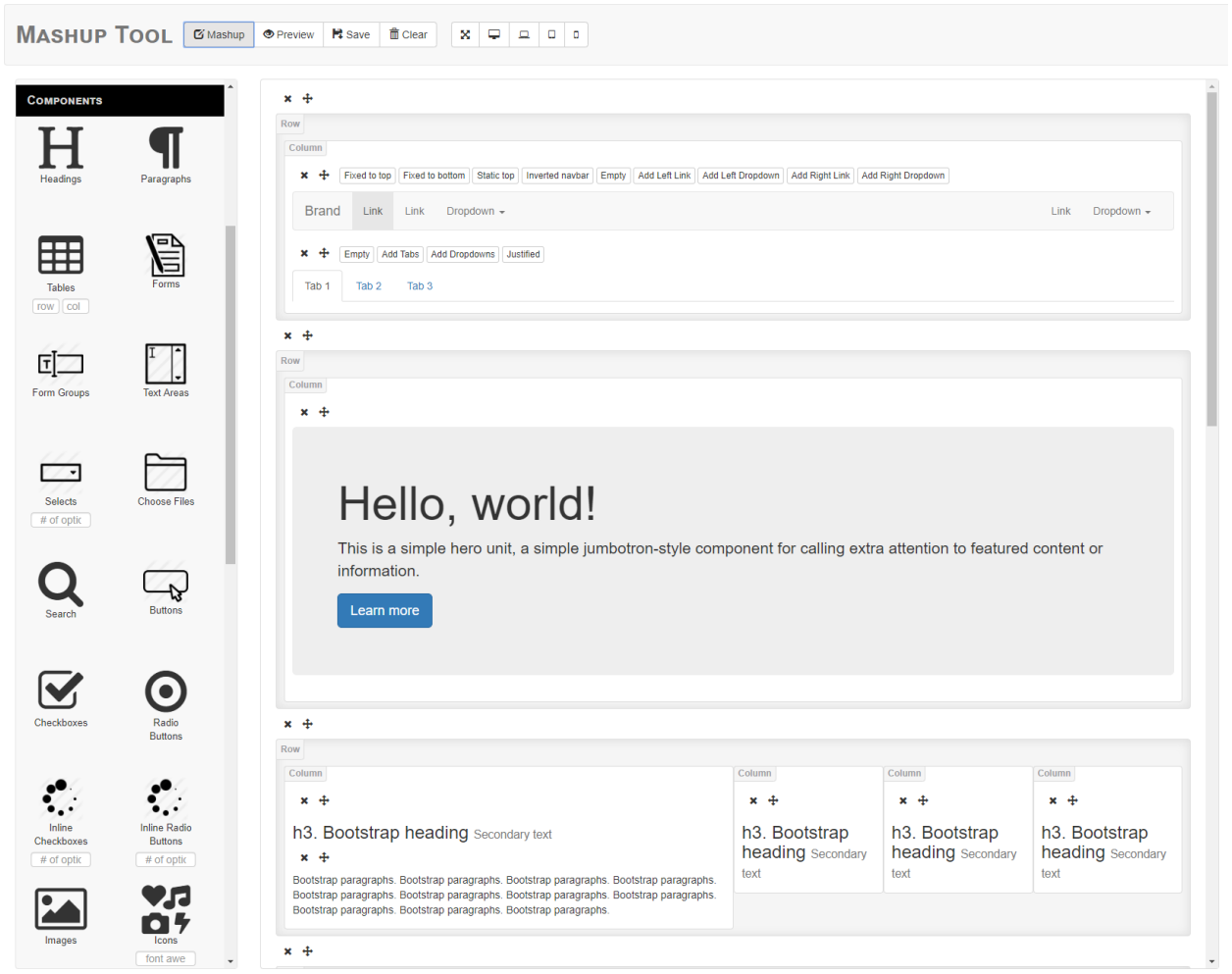

Figure 6.5 Drag-and-drop Mashup Components into The Layout

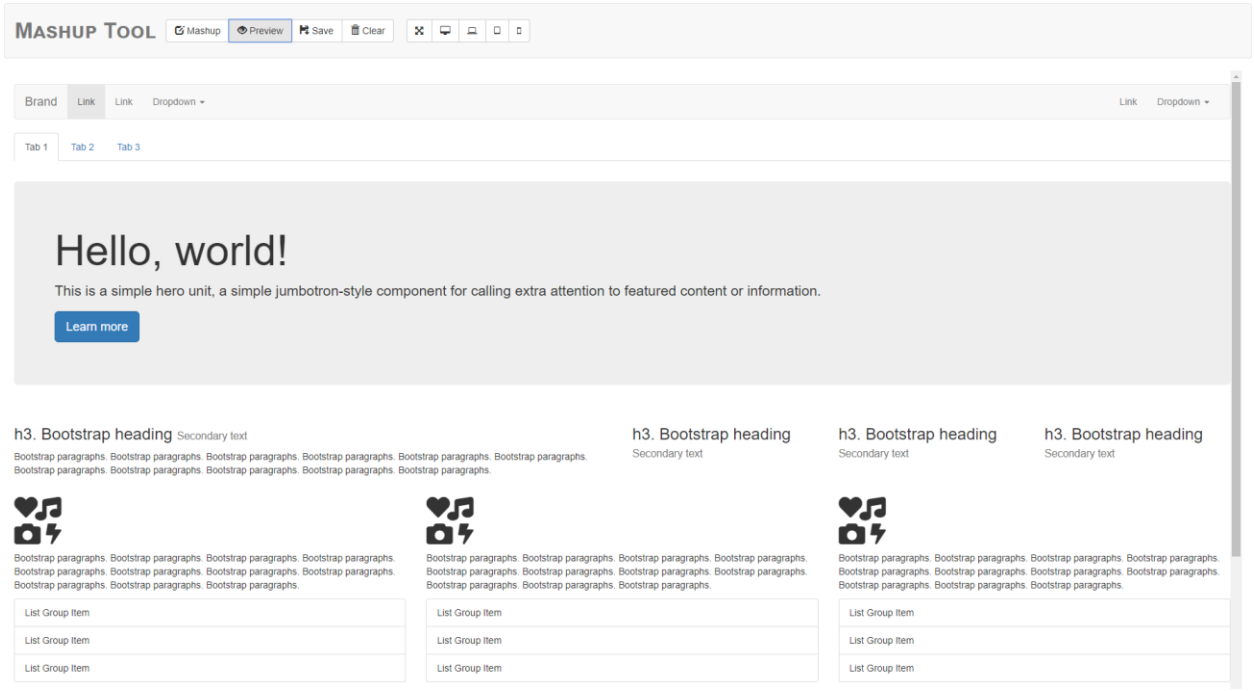

Figure 6.6 Page Preview with Pre-Defined Mashup Components

Step 4: Customize CSS properties of the mashup components

The next step is to customize and style all mashup components by setting CSS properties. The styling panel provides a wide variety of CSS properties and classifies them into different categories (See Figure 6.7). Different mashup components may have different available CSS properties. The mashup tool automatically populates possible attribute values as users type in the field, and shows the autocomplete results in a dropdown list. To generate cohesive and harmonious color schemes, color picker is used for all color-related CSS properties. The preview of the webpage after defining styles is shown in Figure 6.8.

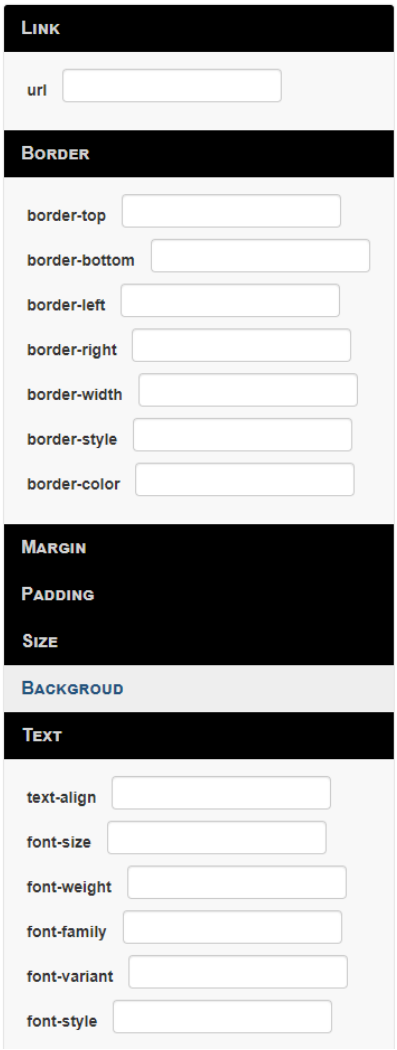

Figure 6.7 Styling Panel

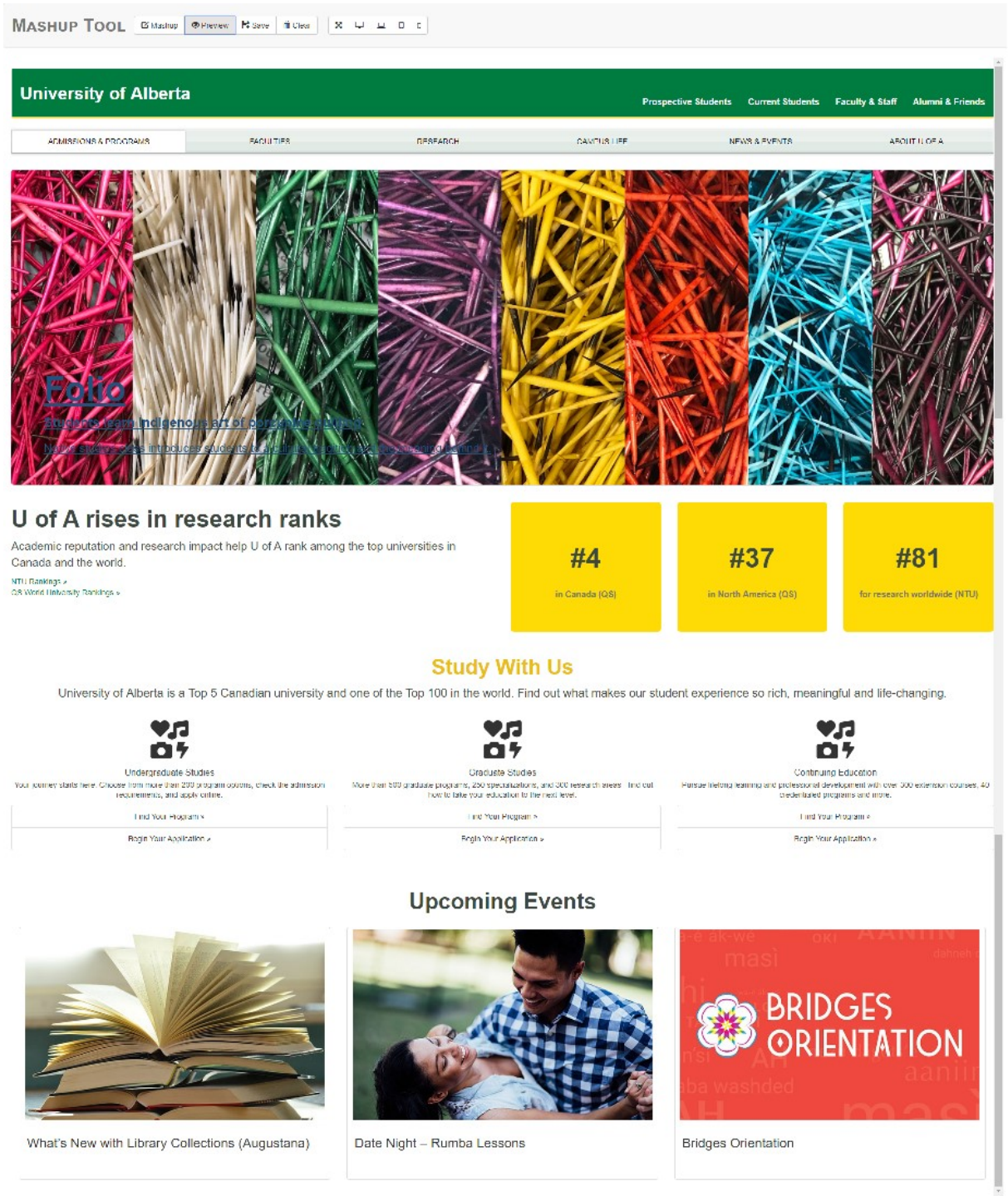

Figure 6.8 Page Preview after Styling

Step 5: Link pages together

After repeating step 1 to step 4 and creating all pages in the web application, the final step is to link them together. The mashup tool allows users to set HTML attributes for all components, such as id, name and hyperlinks. Users can add either external or internal links to elements. To add internal links, a dropdown list of names for all created pages is automatically populated. In addition, when users drag a form element into the page, a table is created within the database with a pre-defined table name based on the form header. Users can override the table name by setting the id attribute of the form. All form fields are stored in the table. Users can display the form data in another view, such as a table or a list, by simply linking the view component to the form table.

7. Evaluation

The aim of the evaluation is to validate the usability of the mashup tool from five specific aspects: usefulness, ease of use, ease of learning, error tolerance, and satisfaction. It also evaluates how the mashup tool implements the ten HCI factors, including cross-platform, levels of liveness, automation degree, modularity, collaboration community, viscosity, closeness of mapping, consistency, diffuseness and responsiveness. Participants were asked to perform assigned evaluation tasks, and then complete a questionnaire with defined evaluation criteria.

7.1 Participants

There were eight participants volunteered in the evaluation. Six of them are graduate students and two are undergraduate students. They were divided into two groups randomly and evenly. To keep the consistency between the two groups, all selected participants had background knowledge related to computer science or software engineering majors, or who at least had taken one programming class. All of them had programming experience with different programming languages, such as PHP, Python, Java, C++, etc. This ensured that they were able to accomplish a series of programming tasks independently. Participants with basic computer knowledge would also allow them to better understand the concepts of all evaluation criteria in the following questionnaire.

7.2 Evaluation Design

A within-subjects design is used in the evaluation process. The principle of within-subjects design is that each participant was tested under each condition [37]. That is, all the participants were required to perform the evaluation under one condition, then evaluate under the other condition. In this case, the conditions were performing all evaluation tasks (a) using the mashup tool, and (b) using any other programming languages.

The reason to apply a within-subjects design is that it can provide more evaluation data with a small sample size. Fewer participants are needed, which makes the whole evaluation process from recruiting, scheduling, to practicing and gathering feedback much easier and time-saving. Another advantage of a within-subjects design is that it reduces variance caused by different personal dispositions. A participant tends to keep behavior consistently across different evaluation conditions. It means that the variability in evaluation results is more likely due to differences among conditions, but not the behavioral differences between participants.

One additional factor must be considered during the evaluation process is learning effects. It is caused by the order of presentation. For example, if participants perform the evaluation tasks under condition A first, then do the same task under condition B, they could potentially give better feedback under condition B because of the prior experience. To reduce the learning effects, all participants were provided with two different tasks under different conditions.

As explained in Table 7.1: four participants in group one were asked to complete task one using the mashup tool, then complete task two with any programming languages they preferred. The other four participants in group two were asked to complete task two using the mashup tool, then complete task one with any programming languages they preferred.

	Group one (four participants)	Group two (four participants)
Using mashup tool	Task one	Task two
Using any programming languages	Task two	Task one

Table 7.1 Within-subjects Evaluation Design

7.3 Evaluation Tasks

Participants were given two different software requirements specification (SRS), SRS 1 and SRS 2, with different objectives and functional requirements (see Table 7.2). Each SRS describes an overall description of a web application. It lists both functional and nonfunctional requirements, and a set of use cases that illustrates how users should interact with the web application. Because that learnability is one of the evaluation criteria, participants were not given any demonstration nor hands-on training for the mashup tool. The total time required for completion of the evaluation process per participant was approximately 3.5 hours, which included an average of 30 minutes for questionnaire completion.

SRS 1	SRS 2
Objectives	
Build an online news system.	Build a university website.
Functional Requirements	
1. News should be categorized and displayed on different pages, including international news, domestic news, business news, entertainment news and sports news. 2. The website should have a navigation menu that allows users to easily switch between different news categories. 3. Users should be able to create, edit and delete news. 4. The news content can be articles, images or videos.	1. The website should contain information about programs, faculties, research activities, campus news and campus events. 2. The website should have a navigation menu 3. Homepage should have a carousel showing the latest news and campus notifications. 4. Programs and faculties pages need to be searchable and sortable in alphabetical order.
Nonfunctional Requirements	
1. The website should contain at least five different pages. 2. The website should be responsive and compatible with multiple platforms and	

browsers, including the latest version of Edge, Firefox, Chrome, Safari, iOS and Android mobile devices. 3. All interfaces should be user-friendly that allow users to easily navigate and interact with on both desktop and mobile devices.

Table 7.2 Software Requirements Specification

7.4 Evaluation Scale

Usability of the mashup tool is evaluated by using the System Usability Scale (SUS). SUS was invented by J. Brooke in 1996 [38]. It is an industry standard evaluation scale for any kind of system, including hardware, software, websites, applications, mobile devices, etc. SUS consists of a ten items questionnaire with alternating positive and negative tones, giving a global view of subjective evaluation of usability. The modified statements for the mashup tool are listed in Table 7.3.

S1	I think that I would like to use this mashup tool frequently.
S2	I found the mashup tool unnecessarily complex.
S3	I found the mashup tool was easy to use.
S4	I think that I would need the support of a technical person to be able to use this mashup tool.
S5	I found the various functions in this mashup tool were well integrated.
S6	I thought there was too much inconsistency in this mashup tool.
S7	I would imagine that most people would learn to use this mashup tool very quickly.
S8	I found the mashup tool very awkward to use.
S9	I felt very confident using the mashup tool.
S10	I needed to learn a lot of things before I could get going with this mashup tool.

Table 7.3 System Usability Scale

SUS is a five-grades Likert scale. It is a method of measuring attitudes invented by R. Likert [39]. A Likert scale assumes that the intensity of experience is linear, i.e. on a continuous form ranging from Strongly Disagree to Strongly Agree. Participants were asked to compare their experience with and without the mashup tool, and rank each survey question from 1 to 5 based on how much they agree with the statement, see Table 7.4.

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
1	2	3	4	5

Table 7.4 Five-Grades Likert Scale

Then, repeat the following three steps to calculate each SUS score:

- For question 1, 3, 5, 7 and 9, subtract 1 from the score.
- For question 2, 4, 6, 8 and 10, subtract the score from 5.
- Add up 10 new values, multiply the total score by 2.5.

Finally, calculate the median of the SUS scores and measure it using Sauro-Lewis curved grading scale [40], see Table 7.5. The average SUS score is 68, which is a grade C. 80.8 and higher is a grade A. 51.6 and under is a grade F.

SUS Score Range	Grade
84.1 - 100	A+
80.8 - 84.0	A
78.9 - 80.7	A-
77.2 - 78.8	B+
74.1 - 77.1	B
72.6 - 74.0	B-
71.1 - 72.5	C+

65.0 - 71.0	C
62.7 - 64.9	C-
51.7 - 62.6	D
0.0 - 51.6	F

Table 7.5 Sauro-Lewis Curved Grading Scale

HCI factors of the mashup tool were also evaluated using five-grades Likert Scale. Participants were asked to rank each survey question from 1 (Strongly Disagree) to 5 (Strongly Agree) based on how much they agree with the statement. The statements are listed in Table 7.6.

S1	Cross-platform	I found the mashup tool functioned well on most platforms including desktops, laptops, tablets and mobile phones.
S2	Levels of Liveness	I found the mashup tool provided immediate visual feedback to me when I made an update.
S3	Automation Degree	I found the mashup tool was fully automated and completed tasks itself during the mashup process.
S4	Modularity	I found that I could use all mashup components or modules independently.
S5	Collaboration Community	I found that I could easily share my work to the public and reuse mashups shared by other people.
S6	Viscosity	I found that I could easily make a change without affecting other parts.
S7	Closeness of Mapping	I found the visual representation of all components matched them closely.
S8	Consistency	I found the semantics were consistent everywhere in the mashup tool.

S9	Diffuseness	I found the languages were reasonably brief.
S10	Responsiveness	I found the mashup tool had fast respond.

Table 7.6 HCI Factors Evaluation Criteria

7.5 Interpretation of Results

The median SUS score is 83.75, grade A (Table 7.7, Figure 7.1). Most participants had positive experiences in regard to the usability. The evaluation result meets the expectations as the mashup tool is aimed to assist end user programming. Six participants agreed that they would like to use this mashup tool frequently. This indicates that they found the tool useful for building their own websites based on their specific needs. The one who disagreed with this statement thought that the tool didn't provide enough mashup components to create complex web applications. Compare to other participants, this person had more work experience in web application development. This shows that the mashup tool is more useful to end users than experienced developers. In addition, two participants strongly agreed and five agreed that they found the mashup tool was easy to use. The majority found the mashup tool were well integrated (1 strongly agreed, 5 agreed), and disagreed that there was too much inconsistency in the tool (6 strongly disagreed, 2 disagreed). Also, most participants thought that people would learn to use the tool very quickly (2 strongly agreed, 4 agreed), and found that they didn't need to learn a lot of things before using this tool (4 strongly disagreed, 2 disagreed). This indicated that the mashup tool is easy to learn. All participants have background knowledge related to computer science or software engineering and so the majority felt confident using the mashup tool (3 strongly agreed, 4 agreed).

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	SUS Score
P1	4	1	4	2	3	2	5	1	4	1	82.5
P2	3	1	5	1	3	2	4	1	5	1	85
P3	4	1	4	1	4	1	4	1	5	1	90
P4	4	2	4	2	5	1	4	1	4	2	82.5

P5	4	2	4	3	4	1	3	1	4	3	72.5
P6	4	3	3	3	4	1	3	2	3	3	62.5
P7	2	1	5	1	4	1	5	2	4	1	85
P8	4	1	4	2	4	1	4	1	5	2	85
Median SUS Score											83.75
Grade											A

Table 7.7 System Usability Scale Score

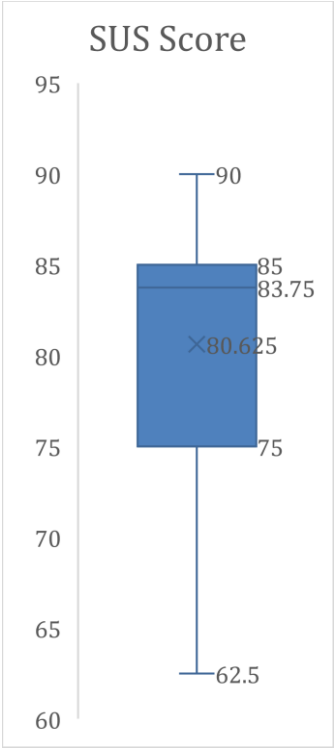

Figure 7.1 System Usability Scale Score

I illustrate the evaluation results with respect to ten HCI factors with a bar chart (Figure 7.2). Five participants agreed that the mashup tool provide good cross-platform compatibility. Three participants were neutral on it mainly because they thought the mashup tool functioned well on desktops, laptops and tablets, but it was challenging to mashup on mobile devices with small screens. Also, the majority had positive responses about the level of liveness and found the mashup tool provided immediate visual feedback (6 strongly agreed, 2 agreed). The number of

neutral and negative responses in regard to “I found the mashup tool was fully automated” is as expected, as the mashup tool only implemented semi-automation. Future implementation will provide more guidance and assistance to improve the automation degree. In addition, all participants found the mashup components well modularized and they could make changes on each component independently (5 strongly agreed, 3 agreed). The ability to share mashups to the public and reuse mashups shared by others is a useful feature and will be added in future implementation. The majority strongly agreed that the visual representations matched the components. They also found the mashup tool had consistent semantics, reasonably brief languages and fast response speed.

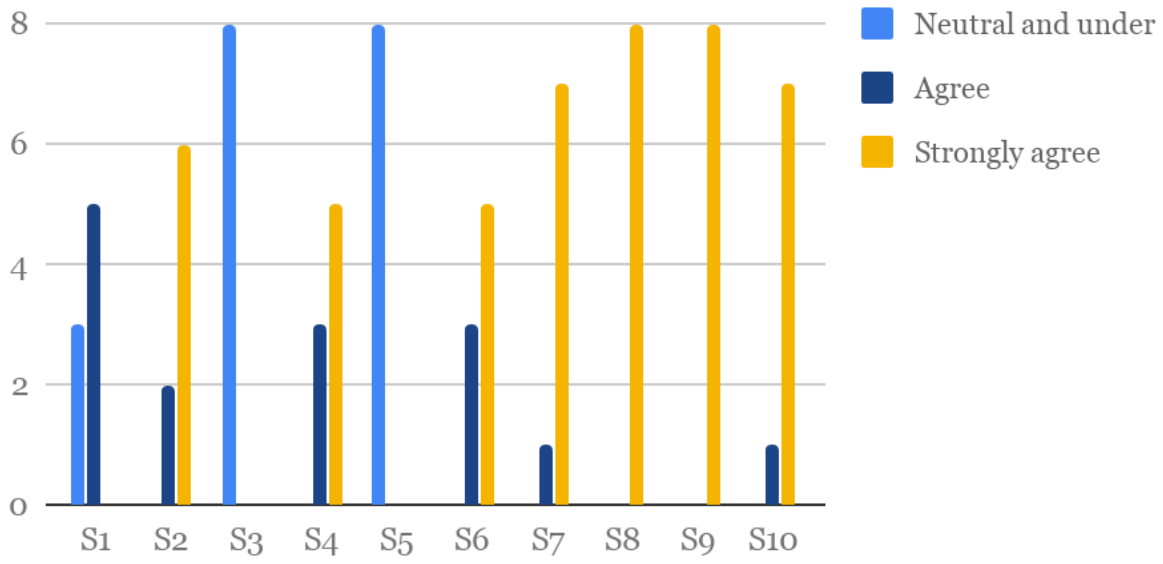

Figure 7.2 HCI Factors Evaluation Results

According to the above analysis of the evaluation results, the mashup tool has achieved a high level of usability. It has proven to be useful for end users to mash up web applications for their specific requirements. It is easy to use, easy to learn, and has high error tolerance and user satisfaction. The evaluation results also demonstrated that most human-computer interaction factors were well implemented by using the techniques and design patterns introduced in Section 4. The mashup tool is semi-automated and it functions correctly on most platforms. Mobile browser support and collaboration community are currently left as future implementation items. The tool has high level of liveness, modularity, responsiveness, and low level of viscosity. The visual representation and semantics have high level of accuracy, consistency, and low level of diffuseness.

8. Conclusion and future work

The main contribution of this paper is the analysis of design guidelines, success factors, implementation techniques and methodologies of end-user programming tools for web mashups. The mashup tool aims at assisting end-users with little or no programming skills in creating web applications to meet their personal or business needs. This paper proposes that usability is an important principle for the development of user-centered mashup tools. Usefulness, ease of use, ease of learning, error tolerance and user satisfaction are essential factors for user experience enhancement. This paper also presents ten HCI factors that guide developers to present the mashup tool effectively to end-users in a highly interactive way. Mashup tools are supposed to have cross-platform capability, provide immediate visual feedback to users, implement semi-automation, host a collaboration community, have a high level of modularity, closeness of mapping, consistency and responsiveness, as well as a low level of viscosity and diffuseness.

To apply the theoretical analysis of these key factors into practice, this paper introduces several techniques and methodologies that can be used in the design and implementation of the mashup tool. Then this paper presents in detail the implementation of a rich-featured, user-friendly and highly-interactive mashup tool. This tool provides a WYSIWYG visual programming environment and enables end-users to build web applications via drag-and-drop. The mashup tool is developed using Vue.js framework to implement MVVM design pattern. MVVM follows the separation of concerns principle which highly improves modularity and reduces the viscosity of the mashup tool. It also makes the tool easier to test and maintain. To implement responsive web design and build cross-browser web applications, the mashup tool uses Bootstrap grid system as mashup layout, and Bootstrap UI components as mashup UI components. As for cross-platform UX design, the mashup tool is developed with progressive enhancement and uses Modernizr for browser capability testing.

A user-centric within-subjects design experiment, followed by a questionnaire with predefined evaluation criteria were conducted to evaluate the usability and HCI of the mashup tool. The results provide a positive feedback in regards to usability, with an 83.75 median SUS score. The results also strongly prove that the mashup tool has achieved a high performance of HCI.

As for future work, a further step is to move the mashup tool to a secure cloud. This allows users to launch their web applications immediately. It is especially helpful for end-users who have limited knowledge about deployment process. Business and marketing tools like Google Analytics can also be added to enable users to monitor visitor actions on their websites. The

development of collaboration community within the mashup tool is a demanding activity. This can significantly enhance team-based cooperation in the enterprise environment, and increase reusability of mashup modules. In addition, add more easy-to-use APIs that provide data, application logic and user interface components to facilitate the composition integration of web mashups and improve the usefulness of the mashup tool. Besides, provide more website templates to the library so that end-users can easily reuse the layout and customize the style and content based on their needs. Showing tooltips when users hover on elements for the first time is another approach to ease the learning curve.

References

[1] Scaffidi, C., Shaw, M., & Myers, B. (2005, September). Estimating the numbers of end users and end user programmers. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)* (pp. 207-214). IEEE.

[2] Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., ... & Rosson, M. B. (2011). The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)*, *43*(3), 21.

[3] Andersen, R., & Mørch, A. I. (2009, March). Mutual development: A case study in customer-initiated software product development. In *International Symposium on End User Development* (pp. 31-49). Springer, Berlin, Heidelberg.

[4] Mørch, A. I., Hansen Åsand, H. R., & Ludvigsen, S. R. (2007). The organization of end user development in an accounting company. *End user computing challenges and technologies: Emerging tools and applications*, 102-123.

[5] Marttila-Kontio, M. (2011). Visual data flow programming languages. *Publications of the University of Eastern Finland, Dissertations in Forestry and Natural Sciences*, (30).

[6] Segal, J. (2007, September). Some problems of professional end user developers. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)* (pp. 111-118). IEEE.

[7] Dietterich, T. G. Department of Computer Science Oregon State University Corvallis, OR 97331.

[8] Hils, D. D. (1992). Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages & Computing*, *3*(1), 69-101.

[9] Burnett, M. M., Baker, M. J., Bohus, C., Carlson, P., Yang, S., & Van Zee, P. (1995). Scaling up visual programming languages. *Computer*, *28*(3), 45-54.

[10] Solutions, V. (2010). VisSim.

[11] Takatsuka, M., & Gahegan, M. (2002). GeoVISTA Studio: A codeless visual programming environment for geoscientific data analysis and visualization. *Computers & Geosciences*, *28*(10), 1131-1144.

[12] Daniel, F., & Matera, M. (2014). Mashups. In *Mashups* (pp. 137-181). Springer, Berlin, Heidelberg.

[13] Liu, X., Hui, Y., Sun, W., & Liang, H. (2007, July). Towards service composition based on mashup. In *2007 IEEE Congress on Services (Services 2007)* (pp. 332-339). IEEE.

[14] Belleau, F., Nolin, M. A., Tourigny, N., Rigault, P., & Morissette, J. (2008). Bio2RDF: towards a mashup to build bioinformatics knowledge systems. *Journal of biomedical informatics*, *41*(5), 706-716.

[15] Aghaee, S., & Pautasso, C. (2014). End-user development of mashups with naturalmash. *Journal of Visual Languages & Computing*, *25*(4), 414-432.

[16] Nielsen, J. (1994). *Usability engineering*. Elsevier.

[17] Quesenbery, W. (2014). The five dimensions of usability. In *Content and complexity* (pp. 93-114). Routledge.

[18] Igbaria, M., Zinatelli, N., Cragg, P., & Cavaye, A. L. (1997). Personal computing acceptance factors in small firms: A structural equation model. *MIS quarterly*, *21*(3).

[19] Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly*, 319-340.

[20] Hong, W., Thong, J. Y., Wong, W. M., & Tam, K. Y. (2002). Determinants of user acceptance of digital libraries: an empirical examination of individual differences and system characteristics. *Journal of Management Information Systems*, *18*(3), 97-124.

[21] Bannon, L. J. (1995). From human factors to human actors: The role of psychology and human-computer interaction studies in system design. In *Readings in Human–Computer Interaction* (pp. 205-214). Morgan Kaufmann.

[22] Green, T. R. (1989). Cognitive dimensions of notations. *People and computers V*, 443-460.

[23] Blackwell, A., & Green, T. (2003). Notational systems–the cognitive dimensions of notations framework. *HCI models, theories, and frameworks: toward an interdisciplinary science. Morgan Kaufmann*.

[24] Tanimoto, S. L. (1990). VIVA: A visual language for image processing. *Journal of Visual Languages & Computing*, *1*(2), 127-139.

[25] Aghaee, S. (2014). *End-user development of mashups using live natural language programming* (Doctoral dissertation, Università della Svizzera italiana).

[26] Fox, R., Cooley, J., & Hauswirth, M. (2011). Collaborative development of trusted mashups. *International Journal of Pervasive Computing and Communications*, *7*(3), 264-288.
[27] Baldwin, C. Y., & Clark, K. B. (2000). Design rules: The power of modularity (Vol. 1). MIT press.

[28] Weik, M. (2000). *Computer science and communications dictionary*. Springer Science & Business Media.

[29] Smith, J. (2009). Patterns-wpf apps with the model-view-viewmodel design pattern. *MSDN magazine*, *72*.

[30] Gillenwater, Z. M. (2010). *Flexible web design: creating liquid and elastic layouts with CSS*. Peachpit Press.

[31] Jehl, S. Responsive Images: Experimenting with Context-Aware Image Sizing EB/OL.

[32] Bos, B., Çelik, T., Hickson, I., & Lie, H. W. (2005). Cascading style sheets level 2 revision 1 (css 2.1) specification. *W3C working draft, W3C, June*.

[33] Lie, H. W., Celik, T., Glazman, D., & van Kesteren, A. (2012). Media queries. W3C Working Drafts, https://www.w3.org/TR/css3-mediaqueries.

[34] Parker, T., Jehl, S., Wachs, M. C., & Toland, P. (2010). *Designing with Progressive Enhancement: Building the web that works for everyone*. Pearson Education.

[35] Stefan Krause (2016). *Benchmarking JS-Frontend Frameworks*. Retrieved from http://www.stefankrause.net/wp/?p=218#more-218.

[36] Stefan Krause (2017). *JS web frameworks benchmark – Round 6*. Retrieved from http://www.stefankrause.net/wp/?p=431

[37] MacKenzie, I. S. (2012). *Human-computer interaction: An empirical research perspective*. Newnes.

[38] Brooke, J. (1996). SUS-A quick and dirty usability scale. *Usability evaluation in industry*, *189*(194), 4-7.

[39] Likert, R. (1932). A technique for the measurement of attitudes. *Archives of psychology*.

[40] Sauro, J., & Lewis, J. R. (2016). *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann.