A Project Report

on

Smart Resto- A smarter way of dining

Submitted in partial fulfillment of the

requirement for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

Under The Supervision of
Mr. Sreenarayanan NM
Associate Professor
Department of Computer Science and Engineering

Submitted By

18SCSE1010336 – MAYANK RAJ

18SCSE1010449 – ABHISHEK UPADHYAY

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GALGOTIAS UNIVERSITY, GREATER NOIDA, INDIA

DECEMBER - 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the project, entitled **"Smart Resto-A smarter way of dining"** in partial fulfillment of the requirements for the award of the **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** submitted in the **School of Computing Science and Engineering** of Galgotias University, Greater Noida, is an original work carried out during the period of **JULY-2021 to DECEMBER-2021**, under the supervision of **Mr. SREENARAYANAN NM, Associate Professor**, **Department of Computer Scienceand Engineering** of School of Computing Science and Engineering, Galgotias University, Greater Noida

The matter presented in the project has not been submitted by me/us for the award of any otherdegree of this or any other places.

18SCSE1010336 – MAYANK RAJ

18SCSE1010449 – ABHISHEK UPADHYAY

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Supervisor

(Mr. Sreenarayanan NM, Associate Professor)

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of **18SCSE1010336 – MAYANK RAJ, 18SCSE1010449 – ABHISHEK UPADHYAY** has been held on _____ and his/her work is recommended for the award of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING**.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date:

Place: Greater Noida

Acknowledgement

We would like to thanks our supervisor, Mr. Sreenarayanan NM for providing continuous: mentoring, general support and helpful advice throughout the project. He has inspired the us in continuous learning and self-improvement during the preparation of this paper.

We also want to express gratitude to the School of Computer Science for providing rich academic resources to support the learning process. We also feel graceful for receiving support of English language improvement from the University.

Finally, the we would also like to thank family and friends for their unwavering support and continuous encouragement.

Without these helps, we would not have completed the report.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Description
SRS	Smart-Resto System
UI	User Interface
HTML	Hyper Text Mark-up Language
CSS	Cascading Style Sheet
OO	Object-oriented
XP	Extreme Programming
HTA	Hierarchical Task Analysis
CRUD	Create-Read-Update-Delete
DAL	Data Access Layer
BLL	Business Logic Layer
PL	Presentation Layer
ORM	Object Relational Mapping
VS	Microsoft Visual Studio
CI	Continuous Integration
TFS	Microsoft Team Foundation Services
EF	Entity Framework
MVC	Model-View-Controller pattern
MVVM	Model-View-ViewModel pattern
Ajax	Asynchronous JavaScript and XML

Abstract

Managing restaurant operations is more challenging than it appears. A restaurant generally relies on paper-based system for manual information flow. However, such system soon meets its limitations. This is mainly because individuals in the restaurant have limited capability to handle massive information flow when the restaurant is at peak capacity. Consequently, many restaurants have adopted computerized restaurant systems to allow efficient operation management.

This project seeks to research, develop and experimentally implement and validate a computerized restaurant system to replace error prone and monotonous paper-based systems. The project proposed a *Smart-Resto System* (SRS), to handle restaurant operations such as order handling, payment processing and inventory control. The two main research sub-domains investigated during the project are *Human-Computer Interaction* (HCI) and *Software Engineering* (SE); as well as the history behind restaurant management and information systems. The project demonstrated SE methodologies from the initial requirement gathering phase to the software testing and validation phase. Some noteworthy practices include establishing software architecture that could promote *separation of concern* and *reusability*, designing essential data structures and algorithms forrestaurant data processing, applying presentation separation patterns such as *Model-View- Controller* and *Model-View-View Model* to decouple software components, and adopting web technology for real-time communication. The project also created intuitive and mobilefriendly user interfaces by utilizing *Hierarchical Task Analysis* (for user interface and task-modelling) and adopting *Responsive Web Design* (for dynamic content presentation), both of which are directly aligned to the HCI methodologies.

A sequence of software prototypes were developed after extensive researches, designs, implementations and testing phases were conducted sequentially. The final prototype satisfied most of the high priority functional and non-functional requirements. Many subsequent features were integrated into the prototypes as the project evolved; these covered the most important restaurant operations. They were each tested and validated in order to demonstrate their capabilities to fulfil the project's objectives.

Literature Survey

Various wireless applications for restaurant ordering have been developed, analyzed and implemented in restaurants. These have been implemented using PDA's (Personal Digital Assistant), Windows Mobiles or Android Mobiles. Also many wireless technologies are available today. The PDA technology has been developed specifically for medium and large-scale restaurants which uses Wi-Fi (Wireless Fidelity) systems.

Captain Pad, a web-based ordering system, is a wireless technology which was being used for automating the ordering system in hotels and restaurants. Using Captain Pad, orders can be sent directly by the customer to the kitchen, this ensures that the customer will be served faster.

Chapter 1. Introduction

This chapter will provide an overview of the project and report. It first introduces to the project context and identifies the associated problems of managing a restaurant's operations. It then discusses the motivations that drove the project to conduct the researches and implement a software solution. Following this, it outlines the objectives that the project intended to achieve. Finally, it describes an overview of the report and the writing styles used in authoring the report.

1.1 Project Context

This project sets to design, build and test a *Smart-Resto System*. Generally, a *computerized restaurant system* aims to solve restaurant problems with Information Technology (IT). A *computerized restaurant system* may be familiar to the reader considering that most restaurants are equipped with a basic cash drawer to process payment. In fact, the terminal used to process payment in restaurant is the origin of such system. This dates back to 1974, when William Brobeck and Associates built microprocessor-controlled cash register systems for McDonald's Restaurants. In this system, tapping on associated item keys and numeric keys would place orders for a customer, it would then continue to calculate the bill when the operator by pressed the total button. This was followed by the invention of the first graphical point-of-sale (POS) system with touch screen support by Gene Mosher in 1978. "*We've eliminated the need for keys*," he said and stated that menu can be changed frequently without programming. Over decades, the *computerized restaurant system* had evolved to cover more operational aspects in a restaurant. Some systems provide full coverage in supporting operations such as: inventory control, customer relationship management, table reservation and staff shift planning.

One of the driving forces behind the innovation of such a system is the attempt to replace the error prone and monotonous paper-based system. Commonly, the workflow of the system would start from waiters gathering orders from the customer on an order sheet, then passing this to kitchen chefs for meal preparation and finally collecting payment from the customer. This process can promote certain risks however, especially during peak period,

they are not limited to the loss of order sheet, incorrect sequence of meal preparation, and added cost due to mistaken orders. Eventually, they may lead to low productivity andcustomer dissatisfaction. Realizing these problems can affect business performance, so the restaurant owner quickly seeks a remedy by adopting IT into their business model.

1.2 Project Motivation

IT has become important tools to support business operations. Especially in the restaurant business, IT is playing increasingly important roles in resources administration, managing services, and assisting strategic decision making. Several analysis and research workshave also suggested that competitive use of IT in a restaurant has significant advantages. In term of operational benefits, it can improve process efficiency, reduce possible human errors, and maximize use of resources. Additionally, it also supports long term business goals, including achieving cost-effectiveness, maximizing profits, and the potential to penetrate wider markets.

Motivated by the IT benefits, the project intended to utilize IT further to improve restaurant operation. To achieve this, the project will investigate the principles and techniques of the *Computer Science* (CS) domain, particularly *Software Engineering* (SE) and *Human-Computer Interaction* (HCI), to build a prototype of *computerized restaurant system*. The prototype are intended to be deployed as web application to support collaboration of various users, thus it will be referred to as a Smart-Resto System (WRCS) subsequently in this report.

The general goal of this project is to develop and experimentally validate a *Smart-Resto System*. This is supported by researching sufficient knowledge in the SE and HCI domains, and then applying this to the development of the system. To achieve this, the project will be underpinned by various stakeholder requirements – which link to the requirement engineering research domain in SE. The project also investigates practical software design methods (a sub-domain of SE) in order to build a high quality system. In addition, it also investigates the possibility of build a real-time information system to allow effective communication – again linking to the previous research domain ofweb development. Lastly, the project will also explore hierarchy task analysis and responsive web design to specify, model, and develop appropriate graphical user interface (UI) behaviours – these are embedded in the HCI research domain.

The following objectives are defined in order to accomplish the project goal:

i) **Developing an efficient multi-tier system architecture**

The system required an architecture design that supported *separation of concern* and highly reusable implementation. It should allow full or part of the functionalities to be accessible by a range of devices. Hence, it should have presentation layer to facilitate an appropriate user interface, a business logic layer that hold common business functions and a data layer for managing database transaction. In-depth analysis and good design practise will be required to realize this implementation.

ii) **Developing efficient information sharing methodologies**

Sharing information should happen instantly since a delay in fulfilling order certainly reduces customer satisfaction. The project will need substantial investigation on how real time communication should occur to allow effective collaboration among staff. Besides, data transfer for cross layer communication could also affect system performance and should be addressed by an optimized solution.

iii) **Designing a simple and intuitive user interface**

In any system, users will need to perform several tasks to achieve a high-level goal. A user interface should guide user through the tasks and help them to attain final goal. Since operations in a restaurant involve numerous tasks, extensive analysis should be performed into designing a UI that is simple and intuitive and addresses the users' goals (functional requirements) effectively.

iv) **Developing an efficient mobile friendly user interface**

The system should be easily accessed by different type of devices; so that it is portable and reusable. Considering that each mobile device may have a different screen resolution and size, the UI of the system should provide a responsive mechanism to offset these limitations with a dynamic UI layout and content resizing.

v) **Ensuring quality of the system through adequate software testing**

A significant amount of testing should be in-place to ensure that the prototype system is free from errors and bugs. In addition, the prototype's performance should be evaluated to analyse the effectiveness of the proposed methodology.

Chapter 2. Background

This chapter describes the background of the system and looks at the project's nature in a wider context. It begins by understanding the characteristics of *computerized restaurant system* and looking at early attempts at such a system. The chapter then discusses how web technology fits into the development of such system. Next, the chapter investigates software process models that best meet the interests of the project. The last section explores the UI design techniques that could enhance user experience and accessibility of mobile devices.

2.1 Computerized Restaurant System

The term, *computerized restaurant system*, which is utilised throughout this project could be obscure to the reader. The general concept for this term is an integrated IT system that supervises, manages and facilitates the planning operations in restaurant. It is not odd that such a system is often associated with a *point-of-sales* (POS) system, a terminal that is use to process sales transactions – e.g. when the meal bill is paid. As stated in §1.1, this was derived from a simple electronic cash drawer which was utilised to collect payments, then it evolved to the basic POS system to the assist order phase and payment process. During 1990s, much investment in IT development focussed on integrating POS with back-office systems such as accounting and payroll systems. Technology advances have allowed POS system, which previously use multiple software packages for different operational purposes, to evolve to fully integrated solution that automate restaurant operations. The all-in-one system, including front-desk service control to back-office planning, is actually a computerized restaurant system. Some drivers behind such evolution, highlighted by are:

- network connected system allows instantaneous connection to services and information;
- real time communication increasingly important to meet customer satisfaction;
- data warehouse and data mining emerge as important tools for decision making; and
- rapid technology changes have challenged the IT capabilities of restaurant stakeholders.

2.1.1 Early Attempts at Computerized Restaurant Systems

Early attempts at *computerized restaurant systems* aimed to improve the workflow of food ordering and kitchen preparation, and proposed a *Process Management System* (PMR) that expand POS system to share order information in real-time. The system addressed the customer order management with timely tracking and validation. It demonstrated potential to reduce fraudulent orders and improve meal preparation efficiency. In addition, there are several research studies that focus on encouraging user interaction in a restaurant system, such as *Multi-touchable E-Restaurant Management System*] and *Mojo iCuisine*. The proposed solutions allow self-ordering of food items by interacting with touch-screen interfaces. Both solutions consist of a touchable digital menu, which can be dynamically updated. Besides enhancing the dining experience, this approach also features flexibility over menu engineering and real time customer feedback.

Finally, adopting mobile devices as part of the restaurant system has gained much attention lately. As noted by, mobile services are "*available at any time and any place*.". They demonstrate the great potential of more portable and accessible functions in a restaurant system. This is aligned to mobile solutions for food ordering in restaurant. The main technologies used to realize their solution are web and wireless connectivity. Web technology provides loosely coupled and platform-independent ways of accessing application services while wireless technology lifted the restrictions of close range operations. This approach is still applicable despite the recent evolution of mobile devices, from the *personal digital assistant* (PDA) to the smart phone.

2.2 Web Application Development

World Wide Web (WWW) application, or *web application*, is any software application that is executed on the web. Originally, the web functioning as an information medium and most of its content remained static. *Web application* evolved though, from statistic textual content with limited interactivity, to rich interfaces with dynamic content and responsive interaction, known as *Web 2.0*. The role of the web has transformed from simple information publication to distributed enterprise-scale workflow systems.

Web application has proven that web technology could help in software development. Three basic elements of *WWW* that are found useful to software application development are

highlighted by as the following:

- *Uniform Resource Locators* (URLs), is a naming scheme to identify computer location, the requested resource in the file system and a protocol to communicate with the resources. The requested resource is not limited to file document, instead, developers may also use an URL to access a particular software service. Modern software applications enable communication across server boundaries by pinpointing the remote resources with the URL.

2.2.1 Designing Web Applications

The concerns of designing and developing *web application* are generally similar from a SE perspective as specified by. The user will interact with the user interface, often a browser, to view and manipulate with the data managed at the server. The tight coupling between web page logics and contents result in poor maintainability and reusability. This isuntil the developer realized that *Model-View-Controller* (MVC) pattern, a well-known software pattern applied in SE, could be applied just as well to many *web applications*.Models are classes containing data and business logics, the Views are web pages with formatting instructions of data, and the Controllers will facilitate communication between Views and Models for data presentation and manipulation. This approach achieves the SE principle, *separation of concerns*, by decoupling presentation logic from business logic. The MVC framework has recently become the dominant development framework and some object-oriented (OO) programming languages (e.g. *J2EE* [2] and *.NET* Framework[3]) would have their own MVC frameworks.

Web application development could be different from general software application development, if following aspects were considered:

- It is concerned with creativity and interactivity of interface presentation.
- It is often content-oriented and required techniques to structure content.
- It needs to cater for a diverse environment as is exposed to various wider range of access device.
- Its distributed architecture promotes unpredictable remote transactions.

The general approach used to develop web-based application is mainly ad hoc. It involves continuing patching of documents on a running web server. Such unmanaged development process lacks quality control and maintainability. This has leads to research of more disciplined approaches, which involve employing *Software Development Process* discussed in the next section.

2.3 Software Development Process

Software Development Process commonly comprises sequence of work activities, actions, and tasks that are undergone to create the final product. In the context of software development, the final product is a software application, a plug-in component, or a software service solution. However, software development processes are complex and unmanaged process could easily lead to catastrophic failure in delivering a usable system. While there are many factors that contribute to its complexity, the two main reasons described by and are:

1) Intellectual and creative processes rely on people's decisions and judgement; and
2) The environment may vary hence producing rapidly changing software requirements or strictly defined criteria.

Consequently, careful planning of development activities is required and this results in the adoption of software development process model. A s*oftware process model* is an abstract representation of interrelated activities in software development. It describes the general approaches in structuring activities and some techniques to produce deliverables. Selecting a suitable model for WRCS would cut down the development time and increase the quality of the output. Following sections describes several widely applied software process model in the software industry.

2.3.1 Waterfall Model

Waterfall model is a traditional software process model introduced by Royce. It is a rigid and linear document driven methodology. This model is known as the *waterfall model* because it proceeded from one phase to another in a cascading order. Before each phase can begin, each of the phases has a definite set of deliverables that must be approved by project sponsor, after the stakeholders have elicited them. However, the process of producing and approving these deliverables will incur significant cost.

The *Waterfall model* often receives criticism on its inability to accommodate changes because the project freezes system specification upon deliverables sign-off. In a dynamic business environment, it is often difficult for user to state all requirements explicitly. The *waterfall model* lacks the ability to accommodate natural uncertainty and the changing need of users.

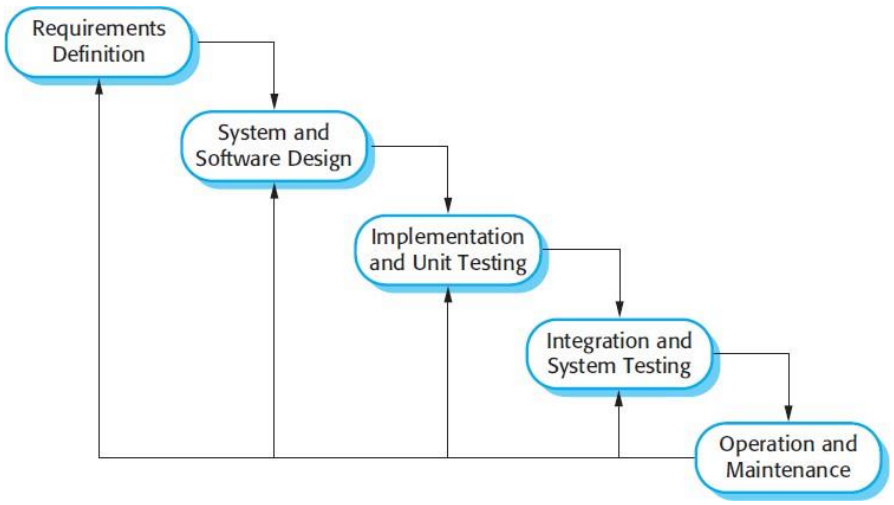

Figure 2.1: Overview of the waterfall model .

Another serious disadvantage of the *waterfall model* is that testing is often left to the end of the project. Errors and feedbacks obtained in later stages will require additional effort to resolve. Eventually, this will lead to a software product that not fit for user need. An enhanced variant of the *waterfall model* known as the *V-model* has improved to this issue. Figure 2.2 illustrates the quality assurance actions associated with deliverables of earlier phases in the *V-model*. Verification and validation approaches applied to earlier engineering work could significantly reduce errors found in later stages. However, the *V-model* does not explicitly describe actions taken in order to deal with errors found during testing. Nevertheless, *waterfall model* does show its strength when used in project where requirements are well understood and stable during development. Documents produced during each phase provide traceability to address safety and legal issues when such concerns are critical to the user.

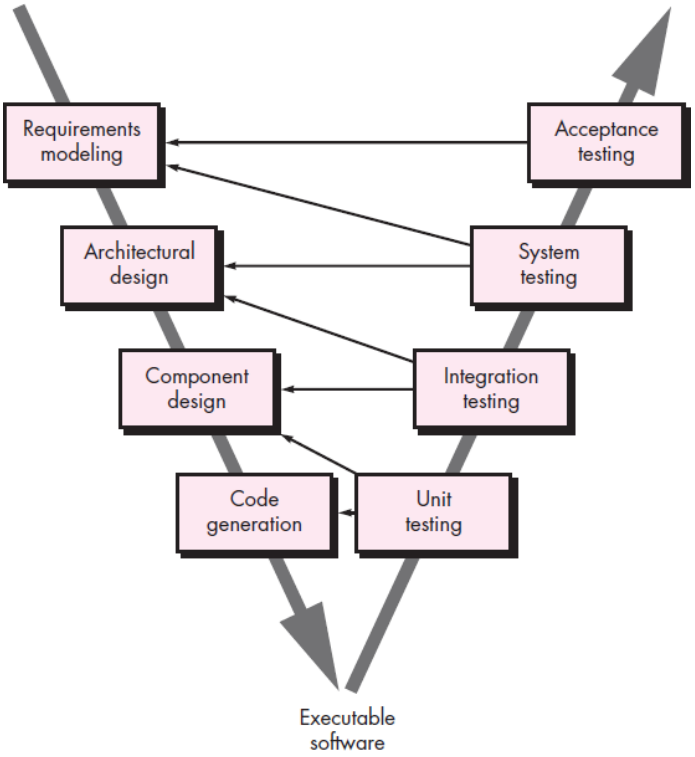

Figure 2.2: Overview of the V-model.

2.3.2 Evolutionary Model

Evolutionary model encapsulates two fundamental approaches: incremental and interactive; when addressing changing requirements. It organizes processes in a manner that enables the development of increasingly complete versions of software based on customer feedbacks through a series of iteration. The two fundamental types of *evolutionary model* that will be covered are *prototyping* and *spiral model***.**

Commonly, there are two types of prototype:

- Common prototypes aim to explore customer requirements through building an incrementally usable system. Prototypes with a minimum set of basic requirements are built and presented for the customer's evaluation. The prototypes evolves by the implementation of customer proposed features and changes until it's functionalities finally agreed by customer; and
- Throwaway prototypes aim to gather information and generate ideas on how system should be built. Commonly during project start up, the user may not fully understand their need and the developer may not share understanding on certain features. To clarify these uncertainties, a design prototype which contain just enough details is

built for evaluation. Once issues have been clarified, developers could then moveon to an actual design and implementation.

Allowing requirements to be implemented rapidly is the key advantage to *prototyping*. However, this may lead to stakeholder confusion by mistreating what they see as final version of the system. Stakeholders should be well aware that some prototypes only serve as tools to gather requirements and may vary from the final product.

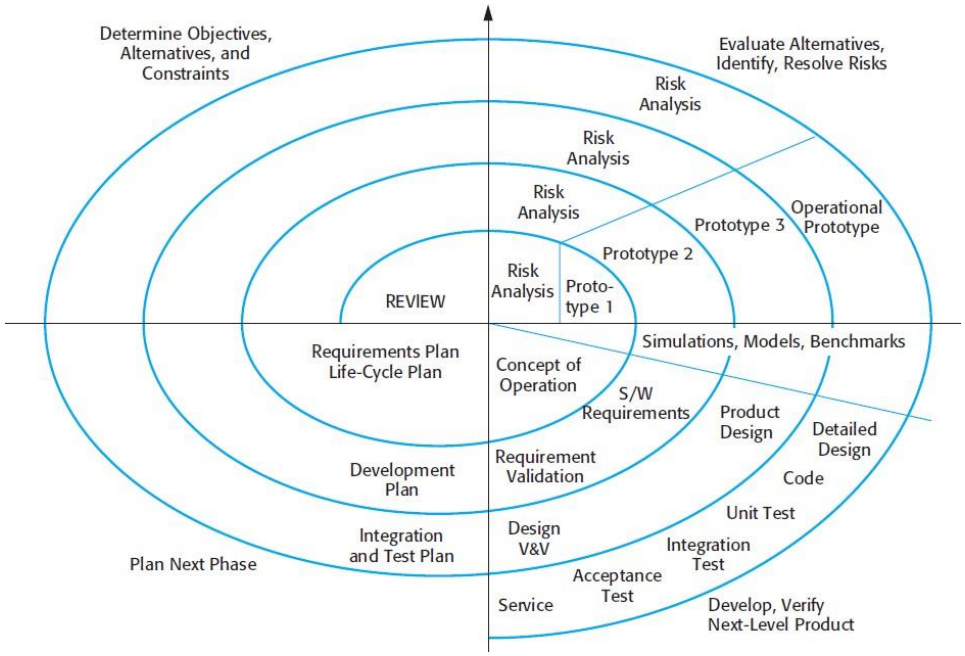

Figure 2.3: Overview of the spiral model.

2.3.3 Agile Development

Agile development processes have emerged to be the dominant software process model in recent years. Agile processes focuses on people, communication, working software, and responding to change as opposed to plan-driven models that have high process bureaucracy. These are best explained with *Agile Manifesto*[4]. Design and implementation are the central activities in agile development processes. It would also be possible to incorporate requirements elicitation and testing into these activities, for instance, applying *test-driven development* (TDD). In TDD, the developer first writes test cases before writing actual implementations. This serves as the preliminary steps to clarify requirements and understanding for problem domains. Developers then code the actual implementations and

execute tests to verify the implementations.

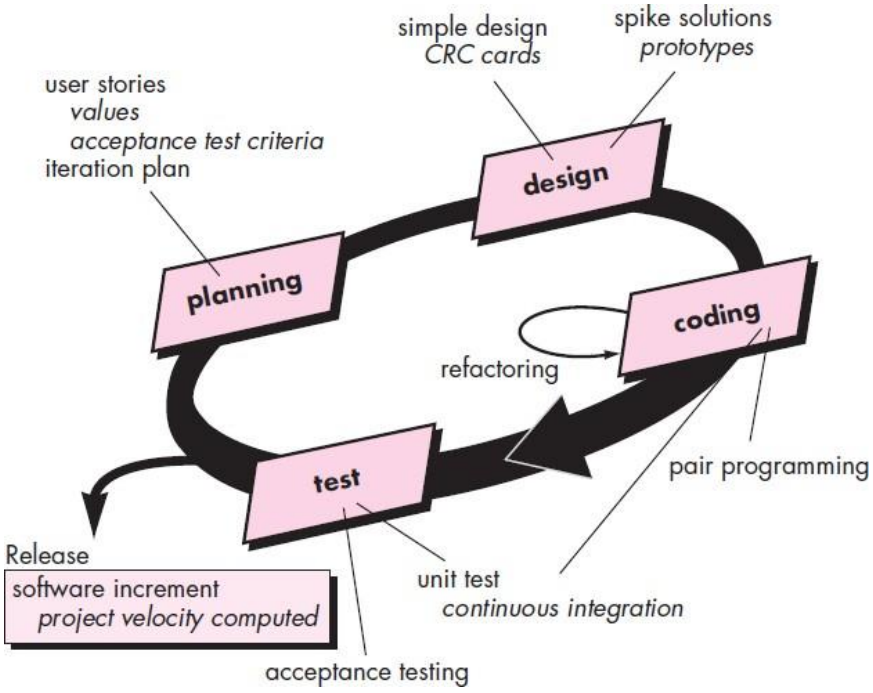

Figure 2.4: Overview of the Extreme Programming (XP) process .

Extreme programming (XP) has been widely known approach since the introduction of agile development concept. Figure 2.4 shows the XP processes and its practise during each phase. XP captures requirements in the form of customer stories or scenarios to determine the features required. In XP, continuous customer engagement in development is important for feedback and acceptance testing. XP favors small and frequent releases of software version like any other agile methods. Thus, design should only meet the current needs and expect refactoring when future improvement is required. XP recommended pair programming among developers because it can enable real time problem solving and quality assurance on solution applied. XP is a lightweight process and fits well for small size projects. However, in a large-scale project where physical interaction among team members is difficult, it could be challenging for XP principles.

2.3.4 Comparison among Software Process Model

Comparison of the three models discussed above based on several concerns that may affect WRCS development activities. These concerns, together with their explanations, are listed below:

- Requirement elicitation, presents approaches to gather requirements for system;

- Change management, reflects how changes will be handled throughout project;
- Validation, explains when testing will be done during project;
- Delivery discuss how quickly and often the software features will be delivered; and
- Design modelling covers the depth of design processes during modelling activity.

Based on the comparison, agile development clearly exhibited features that meet SRS needs. SRS will require segregation of user tasks and roles to model intuitive UI. Apparently, user stories of agile development fit better with these requirements. In addition, agile development has factored change management in the model. Its ability to cope with changes reduces the risk of delivering products that does not meet the objectives. The earlier the system is tested, the less effort will be spent on the error that may arise in end of the project. TDD of agile practise embraces this idea and encourages testing done before development. Connected to this, frequent delivery also implies that new enhancement have actually been verified in smaller scale. It reduces complexity by testing only parts that have been changed. Upfront design often leads to "*design paralysis*[5]*"* when the developer tries to adopt concerns and considerations that may not be materialized in the future of project. This is why agile development prefers modelling just enough detail to support current need and refactor as required. Finally, lightweight agile process such as XP fits well into small-scale development, as in WRCS, which involves only single developer.

Table 2.1: Comparison among Software Process Model.

Concerns	**Waterfall Model**	**Evolutionary Model**	**Agile Development**
Requirement Elicitation	Formal system specification	Requirement and prototyping	User stories or scenario
Change Management	Change is minimum or ignored.	Accept changes and will introduce changes at future incremental version	Accept changes and re-prioritize with current objective for future incremental version
Validation	Testing left to the end project	Testing done at the end of each iteration	Testing done in parallel with development in each iteration
Delivery	Slow and delivers as whole system at the end of project	Delivers increasingly complete software when requirements and design decisions are defined	Always delivers incremental working software with prioritized features

Design modelling	Excessive and lengthy	Limited but may grow quickly if too much emphasis on upfront design decision	Minimum with just enough details to meet current need

2.4 Concluding Remarks

This chapter has covered the background of the research domain and cast it in a wider context. It considered sub-topics that align to the project interest. By covering these topics, the reader should able to understand better the concepts and terminologies that will be used in later sections. However, these topics introduce their concepts from high-level viewpoint. The project will further investigate some of their subdomains, including techniques and tools, in other chapters.

The next chapter documents requirement gathering and project management techniques, marking the start of software development life cycle.

Chapter 3. Requirement

This chapter investigates the necessary requirements of the project based on an understanding and analysis of the needs of system users. It first introduces the process of *Requirement Engineering* in SRS. The techniques utilized to gather and analyze requirements are discussed which then leads to documents on functional and non-functional requirements.

3.1 Requirement Engineering

The first step of the project is to understand user (and indeed stakeholders) requirements for building SRS. Requirements of a system can be defined as descriptions of what services it could provide and the constraints on its operation . These requirements directly address user needs in term of using the system to achieve their business operation – or process. In SRS, business operations are rather obvious to those in the restaurant business.

3.1.1 Requirement Elicitation

Requirement elicitation are concerns with identifying problems to be solved, what the user (or stakeholders) are trying to accomplish with the system, and how the system addresses the business need. The process begins by understanding and analysing the restaurant business problems. Business analysis often reviews that people within an organization would have different needs (or opinions) and views concerned with the overall requirements of the system. They are stakeholders who either directly interact with or are indirectly affected by the system requirements. Hence, the focuses of *requirement elicitation* in SRS are to analyse the stakeholders' roles and how the operations that they performing
– affects the system; and hence must be specified in the requirements.

3.1.1.1 Stakeholder Analysis

Stakeholders are of primary importance to any project due to enormous project resource that has been invested to know exactly what the user wants. If stakeholders are approached earlier in the project, it is easier to communicate their requirement and work out their high priority concerns. The initial step to discover stakeholders' requirements would be via

Stakeholder Analysis. *Stakeholder Analysis* views a system as "*a complex set of interacting elements which working together to satisfy needs or objectives*". The idea is to discover how, when and where stakeholders are involved in the process. As for SRS, the different levels of stakeholders' involvement in the system can be viewed from a stakeholder analysis.

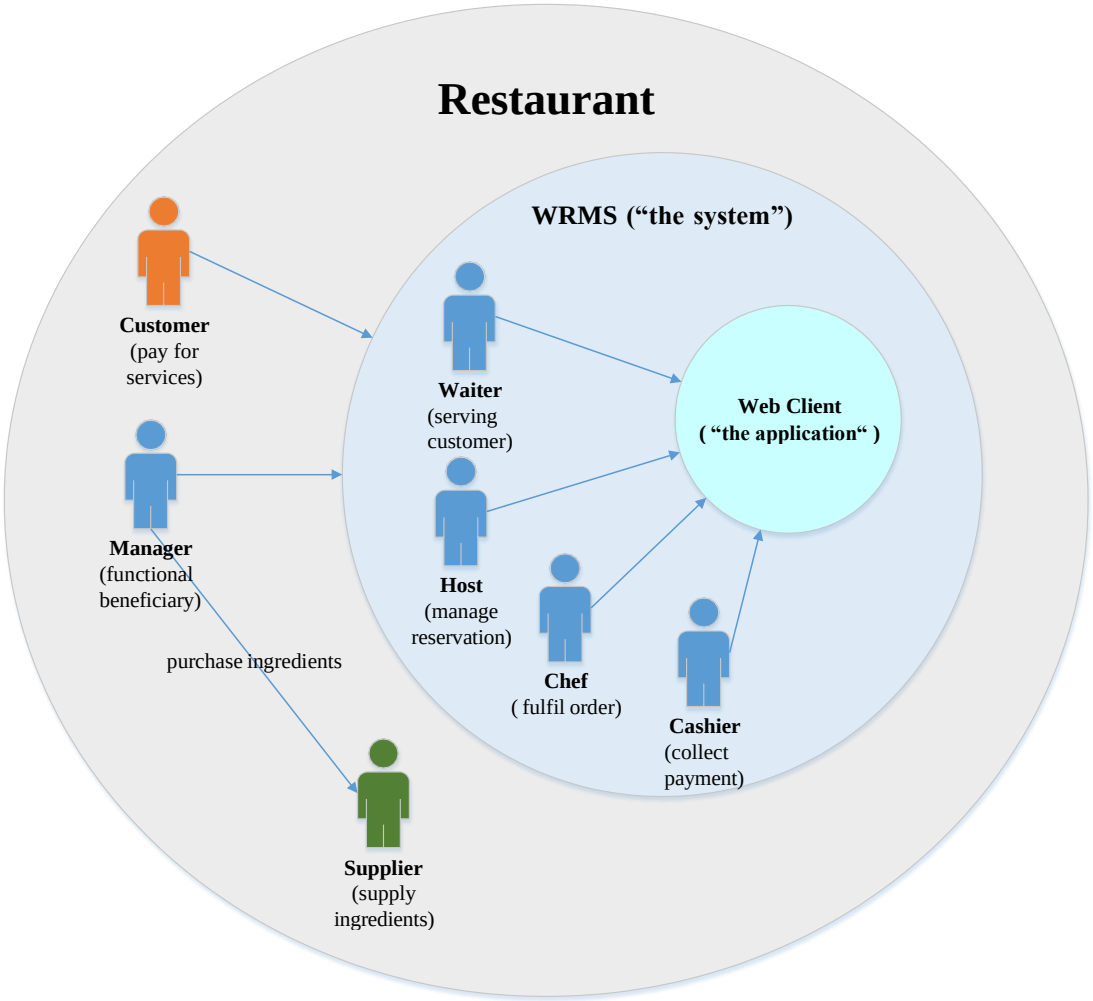

Figure 3.1: Shareholder Analysis of SRS.

3.1.1.2 Identifying Stakeholder Operations

After the initial analysis of stakeholders as shown in Figure 3.1, the next step is to understand the responsibilities of stakeholders. Table 3.1 depicts an overview of stakeholder involved in the restaurants operations. It states the responsibilities and operations of each stakeholder. Although it shows clear segregation of Waiter, Cashier and Host roles, their positions often overlap in reality and can be referred to merely Waiter in general. However, precisely identify the operations involved in each distinct roles is a prerequisite for detailed task analysis in design phases. This table will serve as foundation for consideration of required

system features to support stakeholder responsibility. The functions that support their operations usually are requirements of the system, except Customer and Supplier who do not directly interact with the system – they are considered external actors.

Table 3.1: Stakeholder and Roles in Restaurant Operation.

Roles	Responsibilities	Operation Goals
Waiter/Waitress	Responsible for servicing customers at restaurant dining hall. They gather order information from customers and serve cooked food dishes to customer table.	i) Present menu to customer for order; ii) Collect order information from customer; iii) Submit orders to kitchen; and iv) Answer customer enquiries about order status.
Cashier	Prepare bill and collect payment from customer. They ensure transactions occur and recorded correctly.	i) Calculate amount to be paid for order; ii) Prepare bill for customer; and iii) Collect payment and record transaction
Host/Hostess (Maitre D')	Ensure fine dining experience and smooth customer communication. They manage table allocation and reservation list.	i) Make reservation for table on request of customer ii) Manage table allocation based on reservation list
Chef	Prepare food dishes based on customer order. They also create recipe and track usage of kitchen materials.	i) View order information; ii) Update order status after cook dishes; iii) Design and create recipe; iv) Keep track of inventory status; and v) Assist manager in ordering ingredient.
Manager	Oversee restaurant operations. They ensure sufficient resources such as food materials and staff to operate the restaurant. They also plan menu and promotion for restaurant.	i) Manage staff information; ii) Order material from external suppliers; iii) Generate and view report of restaurant operation; and iv) Maintain menu information.
Supplier	Supply food materials to the restaurant. They receive purchase orders and deliver them periodically.	i) Receive purchase order for material from restaurant; and ii) Supply ingredients to the restaurant.

Customer	Visit restaurant for food and dining experience. They are main source of income for restaurant.	i) Ask for menu information; ii) Order food item based on provided menu; and iii) Make reservation for table.

3.1.2 Requirement Analysis

It is important to gain insight in to what kind of system should be implemented and the level of change that may affect organization before determining which requirements are appropriate for a given system. Hence, the steps taken after gathering the initial requirement involve performing an analysis on information obtained. Some of the basic techniques that could be applied during this process as discussed by and are described in the following:

- Classification and organization of requirements, involves grouping related requirements and organizes into logical clusters or modules. Using model of system architecture is a common way to discover possible modules (sub-system) andassociate related requirements to them.
- Prioritization and negotiations, involves prioritization requirements resolve conflicting requirements through negotiation with stakeholder. The concern is to achieve a set of agreed requirements that considered views of stakeholder involved.

3.1.2.1 Requirement Classification and Organization

As specified above, organizing requirements involves grouping requirement into related logical clusters to identify their relationship and dependency. Requirements may be vague at this stage because they are stated in the form of stakeholder's operational goals. Nevertheless, they could be translated into sets of required system functions. As for the complex software system, grouping related system functions often leads to a modularity view. Adopting modularity views in software architecture allows developers to have clear segregation of each subsystem concern and their relationships.

The result of grouping requirements based on system functions. Eachgrouping is given a module naming. Additionally, it relates the operators (stakeholders) operation goals specified in Table 3.1 to system functions. The last concern in the table is the dependencies of each module, serving as important criteria when prioritizing the requirements in the next section.

Table 3.2: Requirements Grouping for SRS.

Logical Grouping	**System Functions**	**Operators**	**Dependencies**
Recipe Module	i) Create and manage of recipe collection ii) Consider required amount of materials to cook recipe iii) Calculate recipe costing	Chef, Manager	Inventory Module
Menu Module	i) Create and manage menu plans for customer selection ii) View menu item	Chef, Waiter, Manager	Recipe Module
Order Module	i) Submit order for customer ii) Visualize of order items for kitchen interaction iii) Manage order status	Waiter, Chef, Cashier	Menu, Table Module
Payment Module	i) Compute payment amount for order ii) Print bills for payable order iii) Collect payment and record transaction details	Cashier	Order, Table, Menu Module
Table Module	i) Create and manage reservation list for table ii) Provide updated table status	Host	-
Management Module	i) Record and maintain employee information ii) Control and maintain employee privileges level iii) Product report for sales and orders iv) Produce report for material usages	Manager	Order, Payment Module
Inventory Module	i) Record and trace material inventory level ii) Plan material resupply and purchasing	Chef, Manager	-

3.1.2.2 Prioritizing Requirement

Prioritizing requirement involves ranking requirements by weighting the characteristic of requirements in terms of user needs and dependencies. High priority requirements should be addressed first because other requirements often depend on them. These requirements would also fulfil basic operation goals of stakeholders. Prioritizing requirement is an activity in XP according to the principle of incremental planning. Requirements can be changed depending on the time available and their relative priority.

Table 3.3: Features List of SRS.

Important Features
• Creation and management of recipe collection; • Creation and management of menu; • Submission and management of order; • Mean to interact with pending orders in kitchen; • Order processing and notification order status on cooking completion; • Payment computation for order; • Generation of bill and associated VAT; • Mean to collect and store payment transaction details; • Material inventory level monitoring; • Recording and maintaining employee information; • Employee login and privileges level control; and • Reporting of sales and orders

Optional Features
• Mean to attach and store recipe photo; • Creation of composite menu item; • Adjustment of Menu available time; • Mean to attach remark to order; • Mechanism for cancelation and changing order; • Processing order for dine-in and take away order; • Mean to view and search order history; • Reporting on materials usage; • Creation and management of reservation; and • Materials resupply planning.

3.1.3 Requirement Specification

Requirement specification aims to define requirements in clear and unambiguous language based on requirement identified during *requirement elicitation* and *requirement analysis*. The requirements are evolved over time and become more accurately reflect the needs of the stakeholders.

Requirement documents – known as *software requirement specifications* (SRS), contain important statements describing the software product to be built. The level of details may vary depending on the type of developing system. Safety critical and complex systems often require detailed description of constraints or essential domain knowledge. On the other hand,requirements for commercial software are often changing and become out-of-date quickly. It appears that *use case* and *story card* from agile development methods are more flexible in capturing business requirements. Based on suggestion, the agile approaches in documenting requirement are:

1. **Focus on software, not documentation**. Create it only if it is essential to the work effort;
2. **Keep it simple.** Create the most minimalist version of each artefact and use simple tools such as index cards
3. **Proceed iteratively.** Start by identifying a high-level model and gather the details as the work proceeds; and
4. **Work as Team.** Close collaboration could improve communication thus reduce need for documentation.

Following on from this, the next steps are looking at two major categories of requirements: *functional requirements* and *non-functional requirements*.

3.2 Requirement Modelling

In order to understand the system to be built, developers create model(s) to identify important ideas and decisions. There are two types of models that could be created during the software engineering process:

i) *Requirement models* – known as analysis models, represent requirements in terms of informational, functional and behaviour. Requirement modelling explores analysis models to help developers understand the requirements in an intuitive way. These are important tools to prompt user feedback and "*provide a*

means for assessing quality once the software is built".

ii) *Design models* represent software characteristics that inform developers about how the software should be built.

The following sections will cover requirement models for SRS while design models are discussed.

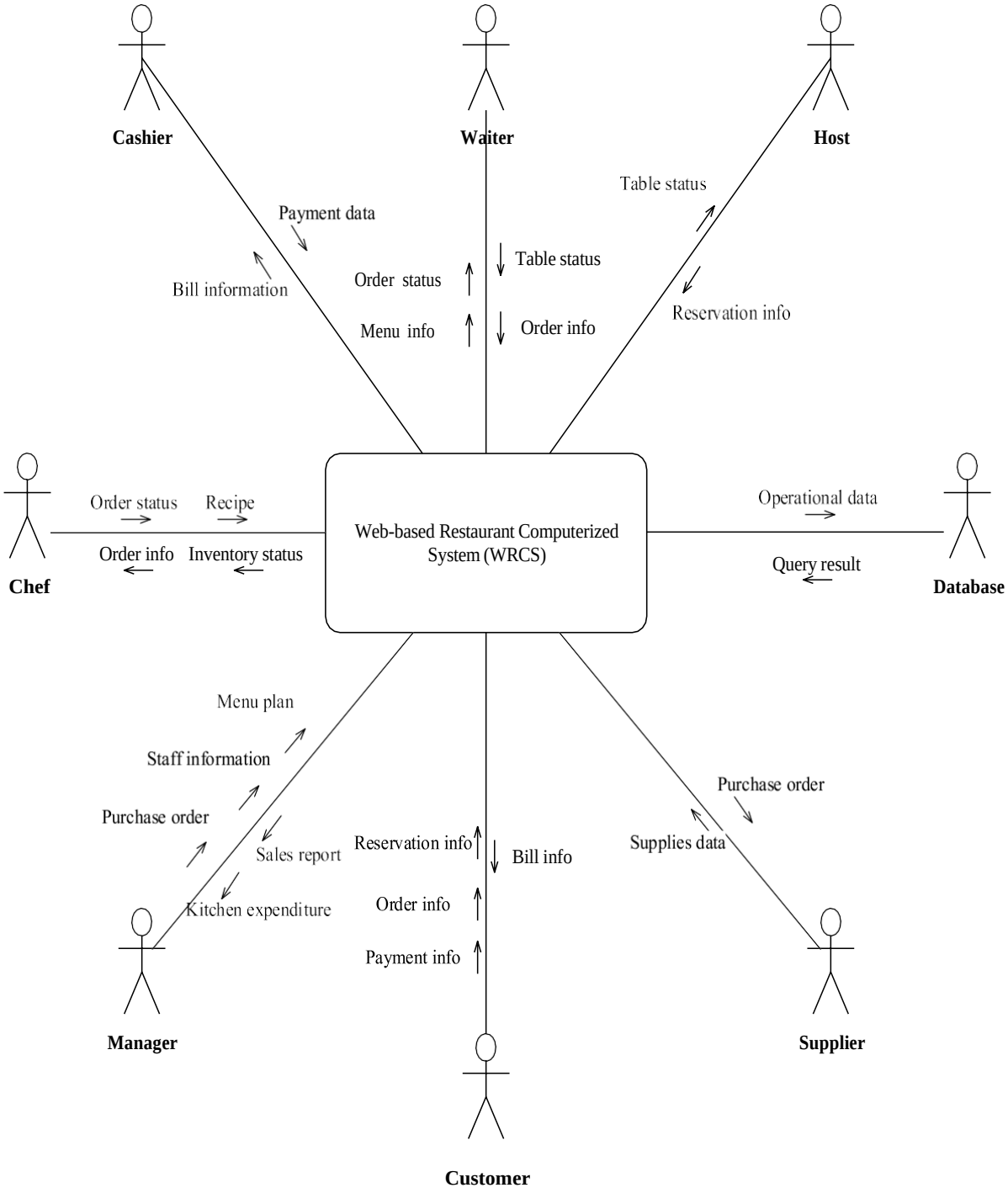

Figure 3.2: Context Diagram for SRS.

Context Diagram (CD) which depicts overview of system working environment is one of the techniques utilised to model the system context. It shows system interaction with external entities and is used to identify information and control flows among these interactions. The two fundamental components in CD are *actor* and *message*. *Actor* represents the external entities with which the system interacts. It refers to a particular user's role who uses the system to perform task or external systems that are required by system to provide functionalities. *Message* encapsulates information flow and controls as part of connection between system and actors. Each connection is labelled with information or particular functions that flow between actors and system. These connections provide insight into possible events that the system must response if the message is a particular type of command. For instance, the system will send notification to kitchen when Waiter submitted order information. A typical message will contain two essential properties: *data content* and *arrival pattern*. *Data content* depicts the information that the message carry while *arrival pattern* describes the nature of message occurrence (e.g. periodically or asynchronous) and possible events that trigger its occurrence.

Figure 3.2 shows the CD for SRS. The list of actors is identical to the stakeholders described – as they are the primary operators of the system. The CD also included external actors that the system was interacting with, such as Database. The message flows show typical information used in a restaurant environment. An example detailed description of the message is listed in Table 3.4.

3.2.1 Use Case Model

Use case modelling is one of the commonly applied modelling techniques in requirement modelling. Its primary use is to capture interactions between users with the system. Interactions that occur within a system could be user interactions such as input gesture, communication with external systems, or collaboration between components of the system . Knowing users' preferred ways to interact with the system also allows developers to capture precise requirements and build a more usable system.

Use cases are simple descriptions of system features from the point of view of users. *Use cases* also capture scenarios of what the user could perform with the system and the expected response from system. Nevertheless, a *use case* is often used to capture *functional requirements* of the system and generally are inappropriate for *non-functional requirement*

.*Use case modelling* involves two major artefacts: *use case diagram* and *use case description*.

Use case diagram is a simple representation of what functions the system allows actors to perform. It provides a high-level view of the relationship between actors and functionalities. Figure 3.3 illustrated the *use case diagram* for SRS. Each *use case* is represented as an oval shape and each actor is represented as stick figure. An actor could provide input and receive output from associated use cases and these associations are depicted by line. The diagram shows all the actors that interact directly with the system features. Managing Software Development Activities

Based on the requirements discussed in previous sections, the project needs to formulate a project plan or sets of development activities derived from software engineering methodologies. As discussed, the XP process (agile method) will be adopted as thesoftware process model for the project. The important concern of adopting software processmodel is not strictly follow every principles and steps – but to use it as guiding principles. This section intended to cover some methods that could be used to improve traceability of project activities to requirements. It involves structuring tasks to be performed (i.e. software process model) and consideration of significant milestones.

3.3 Small Iteration or Releases

The development activities could be easily organized based on the system modules and features. Each modules depicts a major release of the software features thatare then verified against a sub-set of the requirements. It is important to note there are various dependencies between these modules; some lightweight tasks such as defining interfaces forother modules took precedence to enable development smoothness. At the end of each iteration, there are possible chances to re-evaluate requirements and adjust the plans – onceeach is: evaluated, tested, and validated against requirements. Each version of release prototype will be traceable to a FR or NFR . This ensures high priority works are focused on first.

3.3.1 Project management: Gantt chart

A *Gantt chart* presents a series of tasks and their associated period in multiple bars spawning across the entire lifecycle of the project. Each task should have a clear timeframe and due dates as well as proper indications of its dependency. The chart includes a list of significant

milestones. A milestone is defined as a "*significant event in the course of a project that is used to give visibility of progress in terms of achievement of predefined milestone goals*" . It is used to measure and access the project success in meeting the deadline of original planning.

3.4 Concluding Remarks

This chapter covered requirement: theory, understanding and software development methods that could be used to manage requirements – and as stated some were utilised. Based on the identified requirements, the author can estimate the scope of the system and formulate an expectation of the final product. The requirements once analysed show that the system required considerable effort and time to be a fully functional artefact and to meet all requirements. Thus, the high priority requirements were given focus and designed, implemented and tested first whereas low priority requirement – those that were not fulfilled due to time constraint will be discussed in the conclusion chapter. Following this, the chapter also discussed the theory behind managing software development activities and project planning which are key considerations to enable requirements to be compiled and then built. Usage of effective methods has helped the developer to achieve maximum output from the development activities.

The next chapter covers the design concepts and techniques of the project. It describes the process of solving requirements with logical and rational thoughts.

Chapter 4. Design

Once the requirements of the project are established, the design phase will be followed. The design phase is intended to transform the requirements into conceptual solutions that could set a baseline for software implementation. This chapter intends to identify the design needs, investigate the relevant techniques and propose design solutions. It starts by identifying the design principles of agile development and taking them into heart of design activity. The project will then establish a high-level vision of the developing system through architectural design. It moves on to system modelling to create design models to understand characteristics and constraints of the system. Finally, the appropriate user interface design techniques are discussed.

4.1 Software Design

Software design comprises a set of principles, concepts and practices to build high quality system. It is intended to form a solution by appropriate consideration of requirements and technical issues . Software design can be defined as:

Further, gives another view that describes the design space as focused on attaining the stakeholders goals by adapting inner environments (means) to the outer environments (tasks). The outer environments refers to requirements, goals and need; while inner requirements is the set of software, languages, components and tools used to build software (see Figure 4.1).

Software design may have broad spectrum of meanings and objectives based definitions above. Nevertheless, the primary goal of the design process in SRS is to develop concepts and ideas that could answers our research questions while satisfying project requirements. There are four key considerations when performing design activities:

- Manage problem complexity through *separation of concerns*;
- Produce abstract representation of design decision;
- Provide unambiguous meaning to the concepts and terms used in the design models; and
- Establish guided paths to achieve specific end-user task.

4.1.1 Design Process in Agile Development

Design process in traditional software development follows a series of planned design activities or steps, or phases which produces design models that become guidelines to the developers – for the implementation phase. However, this can conflict with agile methods, as if considering an interesting fact:

In order to understand this conflict, explained two styles of design in software development, namely: i) *planned design*; and ii) *evolutionary design*.

Evolutionary design means the design for a particular system grows as the system is being developed. However, evolutionary design is a disaster in common usages because:

- Aggregate of ad-hoc tactical decisions lead to code base that hard to change;
- It leads to poor design when ability to make changes deteriorates; and
- Bugs become exponentially expensive to fix.

4.2 Architecture Design

The design process in SRS started with an initial architecture design. The architecture of software system is described by TOGAF as:

It acts as the important entry point to the design process in software development. described this step as "*architecture envisioning*" and considers it particularly important in scaling software development as it gradually become large and complex. As the project progresses through the various decision-makings, it needs to be well-established architecture that acts as a baseline to such activities. Thus, it is concern about the evolution of the system as specified by the second meaning in TOGAF.

Architecture design is also connected to design goal of achieving *separation of concern* by the decomposition of functional elements into different subgroups. The mechanism of decomposition (or division) will affect every functional modules including the structural definition and their interaction. Thus, the architecture should be planned carefully to ensure that the system and its sub-modules align to the project objectives. This is again linked to the first meaning in the TOGAF definition.

4.3 System Modelling

System modelling is a process to construct abstracts representations of a system in several models, with each models encapsulate different views and analysis perspectives towards the system. In fact, it primarily focuses on creating design models as discussed. It works closely with *requirement modelling* by transforming requirement analysis result to design representation for building software. These models encapsulated requirement understanding such as specification of software operational characteristics; software interface with other system elements; and constraints that software must meet. Often,these design models could easily translated from *functional* and *non-functional requirements* and via versa. They are used throughout development process and they are commonly used for:

- Facilitating discussions about existing or proposed systems;
- Documenting an existing system; and
- Acting as a detailed system description, which could be used to generate system implementation.

Design models are often represented by different types of graphical notation with additional labels to describe their meaning. Depending on the development approach, different types of notation could be used to express a particular design.

Each model could employ several possible modelling techniques and artefacts to represent its perspectives. The project goal does not exhaustively produce every artefacts of these models. It is rather to investigate how some modelling techniques could be applied to achieve the design objectives. The models also may not include every fine-grained level of details, yet these initial models will continue grow as developer refactors the design of the system.

4.3.1 Structural Model

A complex system consists of several sub-components and possibly some external systems. *Structural model* displays the organisation of these components and their relationships. It embodies important consideration about entities that will operate within the system. In additions, stated that *structural model* is to create a vocabulary that can be used by the analyst and the users. Things, ideas, or concepts discovered in the problem domain are represented as given object types in *structural model*, including relationships among such objects . In fact, this process is also known as *domain modelling* in OO development. *Class responsibility-collaborator (CRC) cards* and *Class Diagram* are the two focuses of the SRS structural modelling techniques.

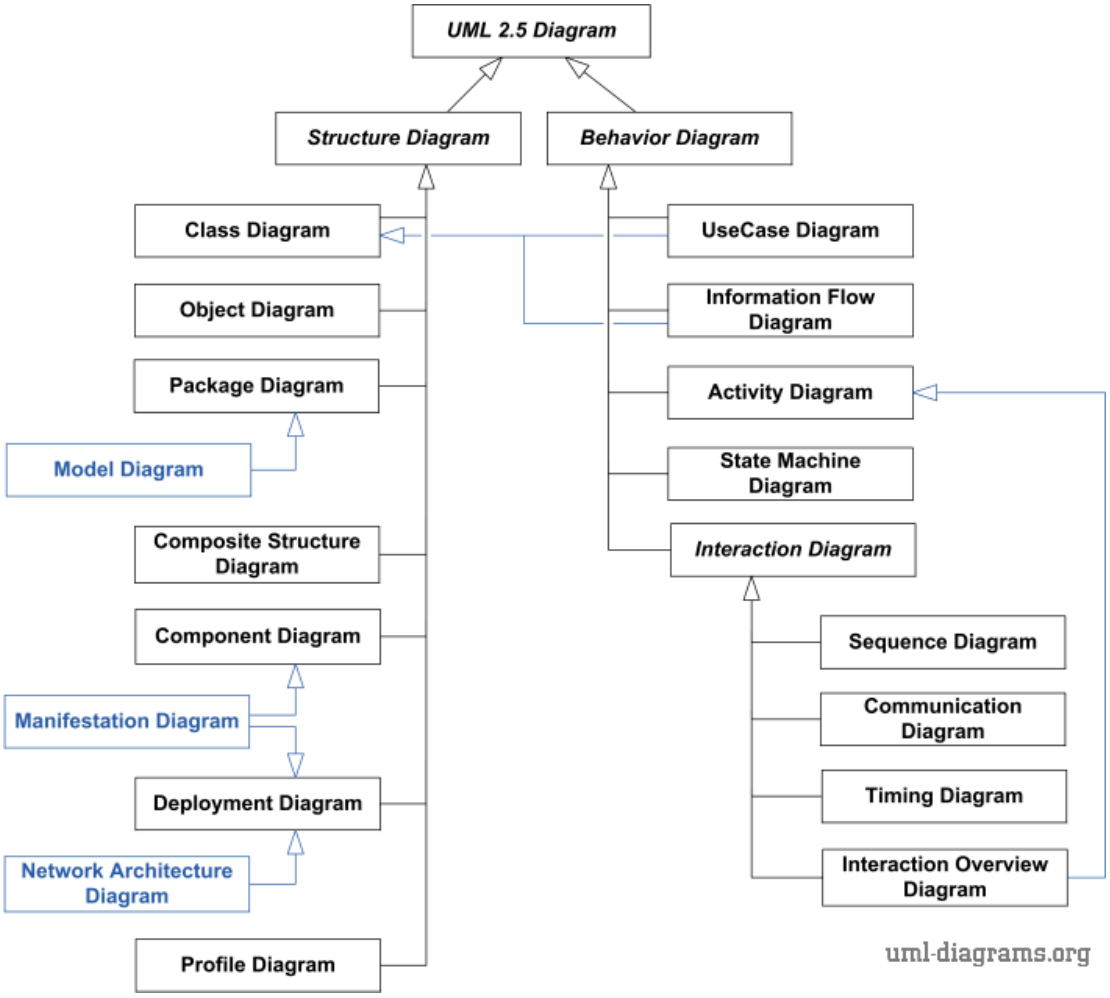

Figure 4.1: UML Diagram Overview.

4.3.2 Behaviour Model

Behaviour model aims to model the dynamic behaviour during the execution of the system . It can used to describe a use case at a specific time and event . *Behaviour model* is particular useful to model a possible business process. The business process can be expressed in a series of continuous interacting objects in the system. Suggested that there are two types of behaviour modelling as shown in Table 4.1.

The table shows a comparison of two types of approaches in behaviour modelling. It describes the purpose of modelling when they should be used and the tools that could help during the modelling phase. As a system to support business operations, SRS is expecting various explicit inputs from end users. For instance, when a waiter submits an order for a customer, the type of item and its quantity needs to be specified. Hence, *data driven modelling* is adopted because it fits the data processing nature of SRS. In addition, since

the modelling process focuses on OO design, *Sequence Diagram* is used to model behaviour as opposed to DFD that is mostly used in a structured analysis design.

Table 4.1: Comparison of Behaviour Modelling types .

	Data driven modelling	**Event driven modelling**
Purpose	Shows the sequence of actions involved in processing input data and generating an associated output	Shows how a system responds to external and internal events.
When to use	Useful when used to show entire sequence of actions that take place from an input being processed to the corresponding output	System has a finite number of states and that events (stimuli) may cause a transition from one state to another
Tools	Data-flow diagrams (DFD), Sequence diagrams	State diagrams

Sequence Diagram is one of the UML diagram to model behaviour and interaction between objects, including actors. It demonstrates a sequence of interaction activities during a system flow. The interaction activities could be a reflection of an explicit sequence of messages that have passed between objects. All the objects will be arranged in a parallel line and a vertical dotted line indicates its active timeline. Specified that not every detailed shouldbe included in a *Sequence Diagram* unless it is used for code generation. The reason being it may lead to a lot of premature implementation design. Eventually, it could easily fall into entropy of "*big design upfront*".

depicts a *Sequence Diagram* to capture behaviour to submit a new order. It consists of high-level view abstraction on the potential classes and messages exchanges required to be carried out to submit an order use case scenario. Messages flows will inspires operations that need to be implemented for the potential classes.

4.3.3 Data Model

Data model is concerned with exploring data-oriented structures. It aims to define the structure of data objects, their relationships and related information that describe the objects and relationships . *Data model* is important because every application, at a certain point,will need to persist its data to certain type of storage. The structure of data, if not carefully designed, will affect data retrieval performance. The *data model* design is closely related with the decisions of data structures. Before this process can even begin, the type of data and storage strategy needs to be investigated and carefully selected to accompany the data

processing need within the system.

4.3.3.1 Handling Data in SRS

The author has explained the characteristics and content of data that flow through the system in different sections in this paper. This issue was initially addressed in *Context Diagram* . The *Context Diagram* let us: the designer (and/or the developer), realise a view of input and output data. The views are further developed into more concrete ideas regarding inner content of data, through structural models. This section develops the concepts related to the explicit data structure concern in this project.

The first issue, if data structure is concerned and related to this project, that needs to be addressed is: persistent or non-persistent. Once the decision of persistent or non-persistent has been made, the next issue is whether this data is going to be stored externally or internally to the application. This implies an external data store that will need to be selected, designed and implemented. In fact, it involves designing different types of data structure. If different types of data structures need to be designed, we need to think about different type of physical files that need to be processed. Hence, the next decision would be the type of files: whether they are simple or complex. If the file is going to be processed implicitly, it could be a text file, or *comma-separated values* (CSV) file. However, if the file required more processing and involves complex read-write operation, it will be more appropriate to select a formal mark-up file as the data store. This could be a data structure of relational model or hierarchy model (e.g. Extensible Mark-up Language).

It shows the type of data that will be required by the SRS. The table explain these data descriptions, their persistent requirement and processing nature. *Application Setting* is the top-level setting that affects the entire behavior of the system. For instance, the tax rate setting will affect every payment received by the system. It hardly changes but often read by application for decision-making and computational purpose; hence, it should be persisted and allowing user to change it. Next, the *Business Entity* refers to the object that holds crucial information representing a real-life entity, e.g., order. This serves as the foundation building blocks to construct system functions. It needs to be stored and often involves heavy CRUD operations throughout application runtime. Finally, *View Data* is a model constructed to carry relevant information to UI for display purposes. It is often reconstructed using *Business Entity* objects and does not need to be recorded. Once we understood the

necessarycharacteristics, the next step would be investigating a data store that could enable effective processing of them.

Table 4.2: Type of Data in SRS.

File Type	Description	Persistent or Non-Persistent	Processing nature
Application Setting	File contains necessary configuration that affect application behaviour	Persistent	Read intensively and Write infrequently
Business Entity	Model that represent areal life entity of the business domain	Persistent	Read and Write intensively
View Data	Model that constructedduring the execution time to display necessary information to user	Non-Persistent	Read intensively

4.3.3.2 Data Storage

Data storage is the container resource for data objects. The data structures that will be persisted in the system need to be stored with one or more data storage methods. This section investigates two main options of data storage and describes the chosen methods for SRS.

4.3.3.3 Relational Database Management System (RDBMS)

RDBMS is data-processing software that employs relational model as its fundamental data structures. Describes the relational model as following:

> *"In the relational model, all data is logically structured within relations (tables). Each relation has a name and is made up of named attributes (columns) of data. Each tuple (row) contains one value per attribute,"* .

It presents the data in tabular form identical to spreadsheet format. The standard language used for data manipulation in RDBMS is Structured Query Language (SQL). SQL consists of two major components: Data Definition Language (DDL) for defining the data structure and controlling access to data; and Data Manipulation Language (DML) for retrieving and updating data. SQL can be very powerful depending on the usage. Some applications leverage its capability to transfer part of the system computation logic to RDBMS through

writing stored procedures with SQL .

RDBMS has become the dominant data storage methods for most software systems today, particularly web applications. There are many existing RDBMS solutions; all share similar sets of essential data processing functions varying slightly in the provided features. There are several mature commercial solutions such as Microsoft SQL Server and Oracle Database. Conversely, there are also free open-source solutions such as MySQL. These have been widely adopted by industry, particularly in small and medium business. The advantages of RDBMS are:

- Support for controlling concurrency and transactional access;
- Support for security management;
- Minimal data redundancy;
- Simple user interface to manage data schema;
- Ensuring data integrity through constraints; and
- Fast data retrieval through query optimization.

4.3.3.4 Extensive Mark-up Language (XML)

XML is a meta-language that allows designers to define their own customized tags in a document. As a type of semi-structured data, XML is designed to be self-descriptive, readable by both humans and machines. It has a loose restriction of schema, thus allowing it to handle data structure that changes rapidly and unpredictably. This is especially true for information on the Web, where it requires certain degree of flexibility to accommodate ever-changing HTML design. The software industry today uses XML as the de facto standard for data communication. It has evolved to be the primary medium of data exchange with external systems and among businesses.

Many technologies have built upon XML by a predefined structured format for XML with XML Schemas. These schemas lead to standardization of XML format when adopted widely by the industry. SOAP and RSS are some of examples of spin-off technologies based on XML. XML also has become popular thanks to a wide range of query languages available, including XPath and XQuery. These languages allow manipulation and retrieval of data to become relatively simple. XML could be considered as the data storage if following advantages can be utilized:

- Ability to deal with frequent and unpredictable schema changes;

- Modelling hierarchy data structure effectively;
- Minimum data conversion if data is used directly from source; and
- Support for multiple platforms.

4.3.3.5 Storage Method Chosen

The storage method chosen for both persistent files (*Application Setting* and *Business Entity*) is the RDBMS. The main rationale was the *Business Entity* – primary data structure of SRS, required extensive cross-referencing. For instance, an order needs to know which recipes been selected and the recipes need to know what are the materials that need to be consumed. Putting this data into hierarchical models will result into redundant data everywhere. RDBMS is also has strong security measures (e.g. access control) and they are important for *Application Setting*, which contains critical data that would affect entire system behaviours.

In addition to that, another key factor that leads to this decision is the existence of *Object Relational Mapping* (ORM) solutions. ORM solution mainly help in converting the relational model into interconnected object graph, coined as the "*Object-relational Impedance Mismatch*" problem. This significantly reduces the complexity of retrieval relational data into OO environment because complex SQL queries could be simplified into normal OO method calls.

Conversely, using XML as data storage could be relatively complex and verbose. The processes to retrieve the data in documents, map them to object and primitive data type in programming language, is difficult. Because the data type constraint is not enforced within XML, the designer will need to handle various kinds of data processing issues such as null value and format mismatch.

4.3.3.6 Entity Relationship Diagram

Entity Relationship Diagram (ERD) is widely used data model to represent relational data model of database. It is "*a picture which shows the information that is created, stored, and used by a business system*" . There are three major concepts in ERD. First, each data object in ERD is a named *entity*, often mapped to a *table* in RDBMS. Second, information

with an *entity* is represented by a set of *attributes* – which capture the data segment (e.g. recipe title) of a data object (e.g. recipe). Finally, association among *entities*, also known as *relationships*, depicts high level business rules of a system. The ERD for SRS is illustrated. This model is reflecting actual implementation in the database fordata persistence.

4.4 User Interface Design

This section develops the concepts related with modelling the graphical user interface (GUI) of SRS. Two important concepts utilized in designing the UI in SRS are *Hierarchical Task Analysis* and *Responsive Web Design*. The following sections will described how the design concepts contributed to the final design with high fidelity design utilizing wireframe diagrams. These solely serve as tools to communicate ideas and thus not every design is shown.

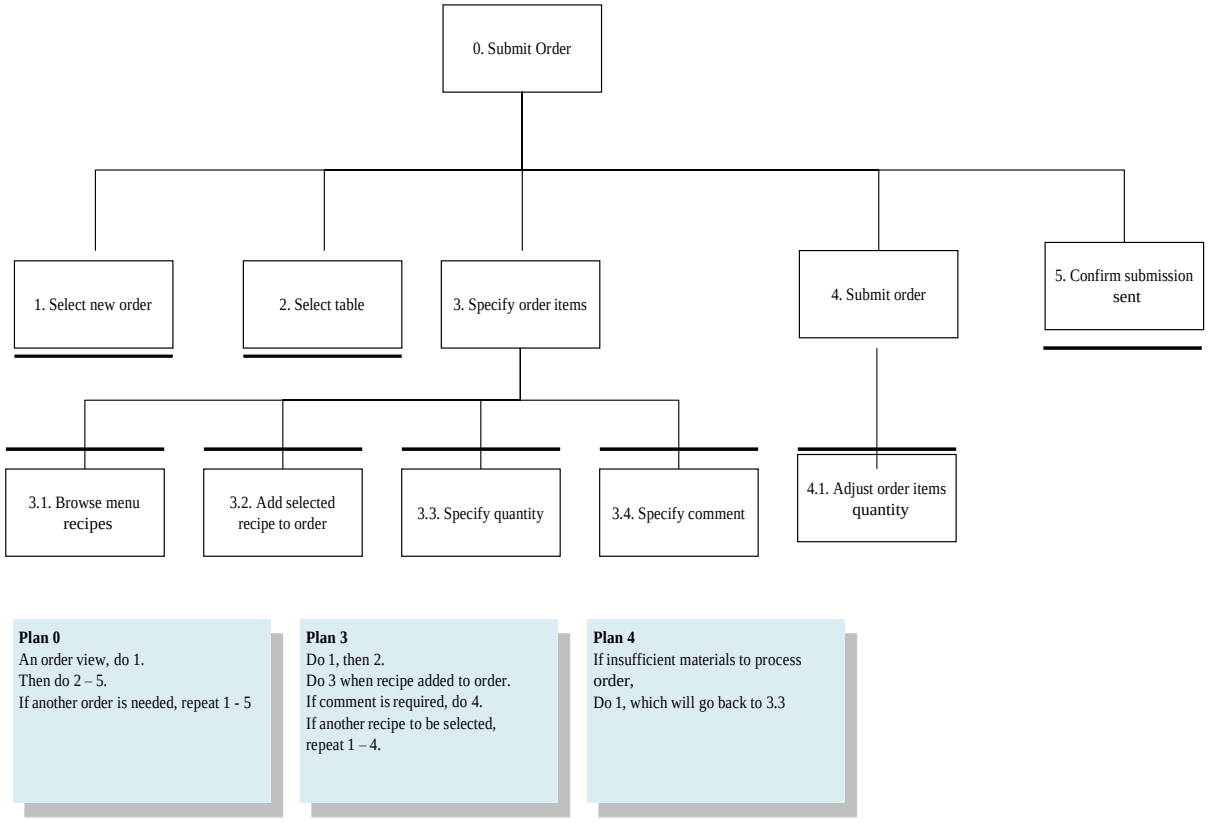

Figure 4.2: HTA diagram for Submit Order.

4.4.1 Responsive Web Design (RWD)

As a web application, SRS mainly interact with its user through a web browser. Web browsers existed in almost every computing device, including mobiles and smart televisions. Hence, this implies the system is potentially accessed through any kind of device. One could argue that we could develop UIs that target each type of device. However, the user's roles may overlap and explicitly forcing them to switch to other devices to carry out the roles could be troublesome. It also places a restriction on the medium to be used to access system, resulting in an additional cost involved to acquire the right devices for the UI. The UI of SRS therefore needs to adapt screen resolution of different devices through RWD. RWD's technical concepts initially discussed in and this section explains how it addresses specific UI design issues in SRS. This is mainly based on Letchford's comprehensivelist of UI issues and their solutions.

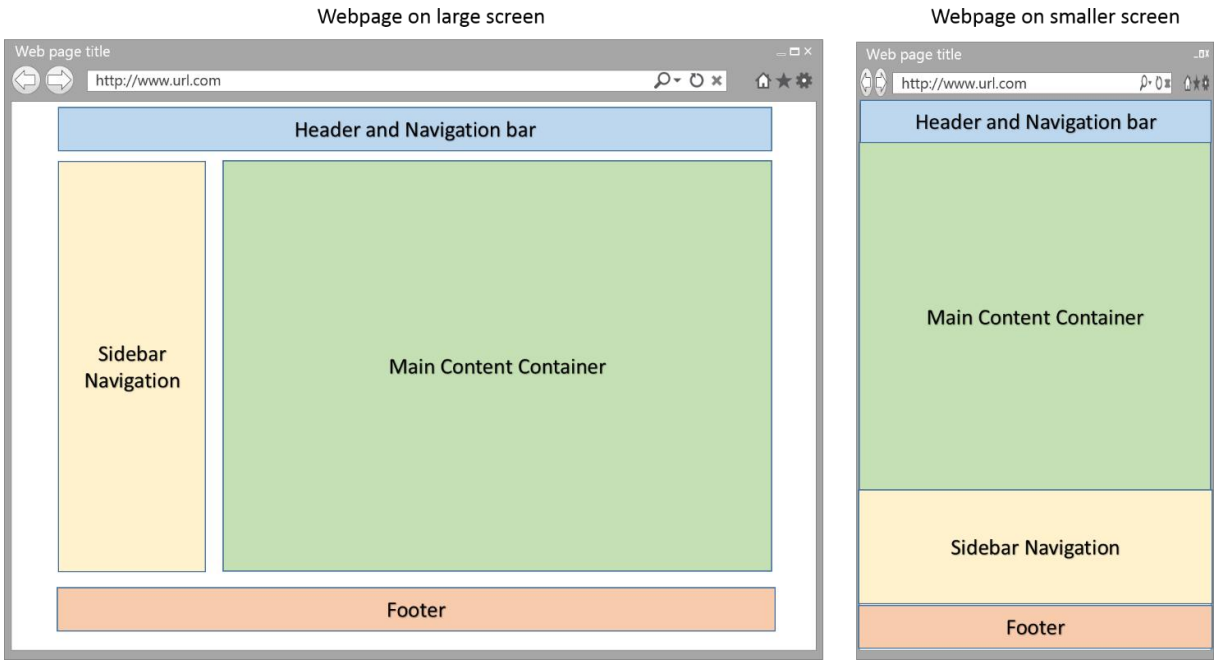

Figure 4.3: Layout changed when viewing on smaller screen.

One of the common issues in UI design is mismatched layout of content on different screen orientations. Landscape layout is wider and its content can exploit the width to present information horizontally. In contrast, portrait layout would have presentation issues with content that with a large width. Particularly on smaller devices, the user needs to browse the content with a horizontal scroll or zoomed out. This could significantly degrade the user experience. To bring this into context, Figure 4.15 presents the common layout of a web page. It usually contains header, main navigation, sidebar navigation, main content container and footer.

Figure 4.4: Flexible Grid System.

Another issue of UI design is the navigation menus that are often span across the header section. A low-resolution screen could not accommodate many menu items in its header. However, these menu items are important navigation concerns and could affect usability of the system. Thus, the system needs a solution to present the menu items while not explicitly occupying spaces in the header section when they are needed. A good solution of would be temporary hiding the menu items from the header section and presenting a button to toggle its visibility as shown in Figure 4.17. This allows users to access the navigation menu whenever they need.

In addition, presenting tabular data on the mobile could be challenging due to table row spanning horizontally. The column that has more text if shrunk down will result in the text wrapping together and making the table look untidy. The solution would be hiding less important columns when viewing at smaller solution as shown in Figure 4.18. The hidden information can be accessed through the "*details"* button, which bring users to a more detailed page for the selected row.

Most data entry in web applications involves form. Users will need to go through every field as in filling out a paper form in real life. On a wider screen, the form could be arranged with its field labels staying side by side with its input element. This again is a problem for the smaller screen because the input elements will span out of view. This introduces the risk of unfilled fields due to poor visibility. The solution would be stacking up the labels and inputs vertically as shown in Figure 4.19. Users could browse every field by scrolling vertically; hence, it less likely to miss a field.

4.5 Conclusion Remarks

This chapter has explored software design concepts in detail – after conducting rigorous researches and explaining how the author formulated the solutions to the requirements. The principles of agile development for software design set the best practice for design activities – and were used by the author. Architecture design established the high-level vision for all the system components design. System modelling enabled abstract representation of a solution that could guide implementation activities. Finally, user interface design addressed the usability and presentation need of the prototype software.

While it is not possible and realistic to include every model that has been produced, the design process has proven that it has carefully considered requirements and constraints that

must be met – and indeed were met. The design models are guidelines but not intended to set implementation details in stone. Thus, it is quite possible that more good practises adopted during implementation have not been explicitly specified.

The next chapter specifies the actual implementation process. It explains how the software that has been implemented, based on the design ideas that are developed in this chapter.

Chapter 5. Implementation

This chapter will cover the theory behind software implementation and complex features; that had to be researched and then implemented. It will mainly discuss the development details of the previously discussed design ideas (§4.3 and §4.4) while addressing project requirements. It first explains the implementation technologies – which are required for web application development. Then, it describes the server-side implementation and client-side implement of the system. The server-side implementation will mainly broken down by architecture layers (see §4.2.1.3) which also includes the algorithm of data processing. Client-side development is concerned with the UI implementation. Finally, this chapter presents a series of walkthroughs that depicts the actualusage of the implemented system.

5.1 Implementation in Agile Development

As discussed, XP was employed as the software process model. Implementing the project requirements will involve iterative and incremental release – a core agile practise. In this phase, the development process will be broken down into series of time- boxed implementation iterations. The iteration will involves software designing, programming and testing. Before each iteration begins, a set of requirements to be implemented is specified. The deliverables of each iteration will be an incrementally functional prototype software based on the specified requirements. In SRS, the breakdown of iteration follows a series of prototype version. Source code repository and software version builds.

5.2 Web Development Framework

Implementation of a software product requires various software development kits (or frameworks) such as programming tools, build system and source code repository. Utilizing the right tools will significantly improve the development efficiency, performance and software quality. Given that this project was set to develop a web application, it involves both the server side and client side of development. Client side development mainly involves constructing UIs (mainly written in HTML) for the web browser. Conversely, server side development employs the OO paradigm when implementing the required algorithmic and data structures. Thus, it needs a *web development framework* that could bridge the gap and the mismatch between the two-programming environment; client- & server-side.

This project employs ASP.NET MVC as the *web development framework*.

The term *design patterns* refer to the enormous number of built-in software libraries that could be used to solve common web development issues such as model-view separation, UI, routing, dynamic content, etc. The framework and its architecture are based on the MVC pattern; which is an excellent candidate to achieve *separation of concern.* Another key factor is ASP.NET's support for server push technology[10], which employs the publish-subscribe model[11] to maintain communication between server and web client. This is particularly important for the order notification between chef and waiter to ensure that the order requests are handled in a timely manner. In addition, ASP.NET MVC has security features that are built into the application framework. Particularly *role-based authentication*[12] is a perfect candidate for SRS since it will be accessed by various user role. In additional, ASP.NET MVC employs a web template engine, named *Razor*, to create dynamic HTML. Web template engine refers to a solution that separates the presentation of HTML from its data, allowing them to be interchangeable at the run time. The steps to use a template engine are specified by as below:

- Specifying a template to use;
- Assigning data to the parameters as the actual content; and
- Injecting the template with the parameters to generate HTML results.

5.2.1 Server-Side Programming Language

The underlying server side technology for ASP.NET is the Microsoft .NET Framework. While there are a number programming languages available under .NET Framework, C# and Visual Basic (VB).NET are the most widely adopted languages. VB.NET employs OO paradigm and emerges as an evolution to its predecessor Visual Basic[13]. The syntax of VB.NET is more literal and closer to the pseudo code expression. In contrast, C# syntax is much closer to the other widely adopted OO programming language such as Java and C++. The code block is wrapped with braces and could be omitted for a single line statement. Both languages are fully supported by the .NET Framework and easily port to the counterpart.

C# has been selected as the programming language for this project due to the author's familiarity with the language and the fact that many code samples in .NET community are published in C#. Other interesting features of .NET that heavily exploited in the project are LINQ and *Lambda Expression*. LINQ stands for Language-Integrated Query and is particularly useful to perform query and operation against data. *Lambda Expression* is a style of anonymous function that could be passed as arguments to another function call. A

combination of both features allow the developer to define efficient queries against the data source such as entity class of ORM or enumerable object collection. Client-Side Development Language

The client-side of the web application refers to the web browser. Regardless of whichever technology is employed to construct the UI, the web browser will only process the received view as HTML. Hence, the client-side UI of SRS is mainly written in HTML5. HTML5 is the new standard of HTML and currently supported by a wide range of web browsers.Some HTML5 characteristics desired by the project are :

- Adding new functionality is purely based on HTML, CSS, Document Object Model (DOM), and JavaScript. This avoids unnecessary installation of external software plugins to the user device; and
- HTML5 prefers more mark-up to replace scripting. Many common features (e.g. validation, UI elements) are included in the standard of HTML5, hence the project can avoid reinventing the wheel.

However, HTML is static and its presentation unlikely to change once rendered by the web browser. Therefore, the project also utilizes JavaScript to introduce dynamic behaviours to the HTML pages; e.g. responding to user inputs, changing content structure and styles.

JavaScript has become the dominant client side of scripting technology in recent years. It is open platform and has numerous libraries that enable the streamlining of modern web development. Most web browsers today support *JavaScript* execution, thus allowing more interactive and responsive experiences built into web applications. ASP.NET comes with several helpful built-in JavaScript libraries. The most notable *JavaScript* library is *JQuery* and its validation plugin. *JQuery* provides API for HTML document traversal and manipulation, event handling, animation, and Asynchronous JavaScript and XML (Ajax). It allows the developer to create rich and dynamic UIs at the client side. Besides, the validation plugin contains the common validation functionality for HTML input such as textbox. This could significantly reduce data entry error and improve data integrity

Plain HTML is not appealing to users and is likely to provide a poor user experience. Hence, HTML often uses CSS to define its look and feel. Aligned to the concept of RWD discussed in §4.4.2, the project needs a CSS framework to provide the responsive features and to enable the views scale properly across different resolution devices. Besides, it may also save

significant development time when customizing the layout. Hence, the project utilizes Bootstrap as the main CSS framework. Bootstrap is a

5.2.2 Integrated Development Environment (IDE)

The default *Integrated Development Environment (IDE)* for ASP.NET development is *Microsoft Visual Studio* (VS). It is developed by Microsoft to support a broad range of application developments, ranging from simple standalone to large-scale enterprise solutions. VS is an excellent choice because it has built-in support for almost all aspects in software development. The project mainly utilize following features to ease development activities:

- Package management for third party libraries;
- Rapid code refactoring and design warning;
- Step through debugging;
- Unit test management and execution;
- User friendly code editor with responsive code completion;
- Source code version control; and
- Database management.

5.2.3 Relational Database Management System (RDBMS)

As specified in §4.3.3.2, this project employs RDBMS as its data storage method. The project does not explicitly constrain the RDMBS selection to a particular vendor. In fact, the RDBMS usage could be particularly different when viewed from the development stage and the deployment stage. In the development stage, the data is often dummy and easily generated. The data schema also changes rapidly as design decisions changed. The database's tables need to be constantly dropped, altered, recreated; and facilating testing. Such required functionality would be better implemented by a lightweight RDBMS; that would allow instant creation of a database from scratch. On the other hand, the performance and functionality of RDBMS is more demanding when the system is deployed into the actual environment.

5.2.4 Continuous Integration (CI) Software

As discussed, the project will require CI software for source control and manage software

builds. The source control software is mainly used to manage the different versionsof source codes for the prototype software – as it is being developed. This is particularly important in XP because new features are introduced to the code base rapidly. Versioning source code allows developers to roll back software changes in the event of these changes affect the existing system – or present release versions. It also helpful when the developer is fixing errors, indicating what changes introduced bugs and failed builds – if test harnesses are utilised. Although not relevant to this project, the software is also valuable to the project when the project involves multiple developers. The software provides friendly UI to manage and resolve conflicts between concurrent editing on same version of source code.

Managing the software builds is another scenario where CI software was found to be useful. In XP development, tests should be passed before committing new changes to the code base. However, building and testing all the software packages of the system for small changes could be time consuming. Build management software addresses this issue by automating the build process when new changes are committed. The built software then verifies these changes by a series of automated tests. This helps to identify compiling error and run-time errors and thus provide meaningful indications if the new changes work.

SRS employs *Team Foundation Service* (TFS) as its CI software. TFS is a cloud- based solution for *Team Foundation Server* – which provides full support for application lifecycle management.

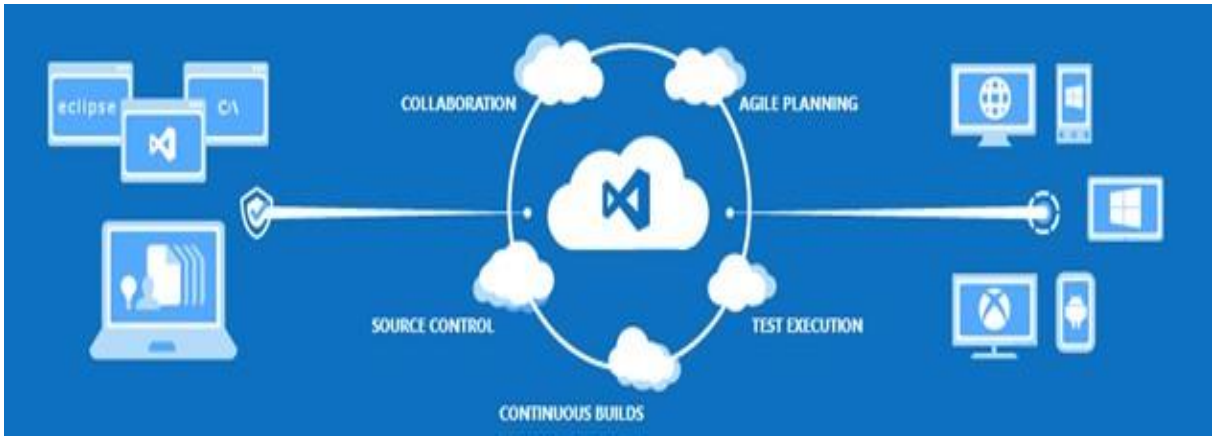

Figure 5.1: Overview of TFS Features.

5.3 Implementation Details

The implementation of the SRS is best explained using the architecture layer design methodology established. The *software architecture* is broken down to three major layers: *Presentation Layer (PL), Business Logic Layer (BAL)* and *Data Access Layer(DAL)*. Each layer involves using different technology to solve their issues. In order to manage these layers, the software packages are structured identically. However, the *domainentity* classes of BLL, which will be used across all layers, are placed in another package.

The development and implementation of a specific requirement will involves all three layers. For instance, developing and implementing the *manage recipes* feature involves defining the *Recipe* entity class. This is then followed by configuring the ORM class to accept this entity. All the operational logics that is related to *Recipe* such as CRUD operations will be specified in the *RecipeServices* class. When the backend logic is ready to perform its function, the controller class and relevant view will be implemented to support user interaction. The view implementation is divided into client side and server side implementation. The server side implementations construct the relevant HTML views and send them to the client's web browser. At client side, JavaScript will be written if it requires dynamic functionalities or behaviours. The entire process is abstracted.

The next sections will explain the implementation details of each layer. In these sections, some of their implementation details will be depicted in the form of actual code snippet or algorithmic abstraction. Variables will be preceded by a $ symbol when abstraction form is used.

5.3.1 Data Access Layer (DAL) Implementation

SRS requires a solution to manage its data access and persistence. The answer to these requirements will be *Entity Framework* (EF). EF is:

> *"an object-relational mapper (ORM) that enables .NET developers to work with relational data using domain-specific objects. It eliminates the need for most of the data-access code that developers usually need to write,".*

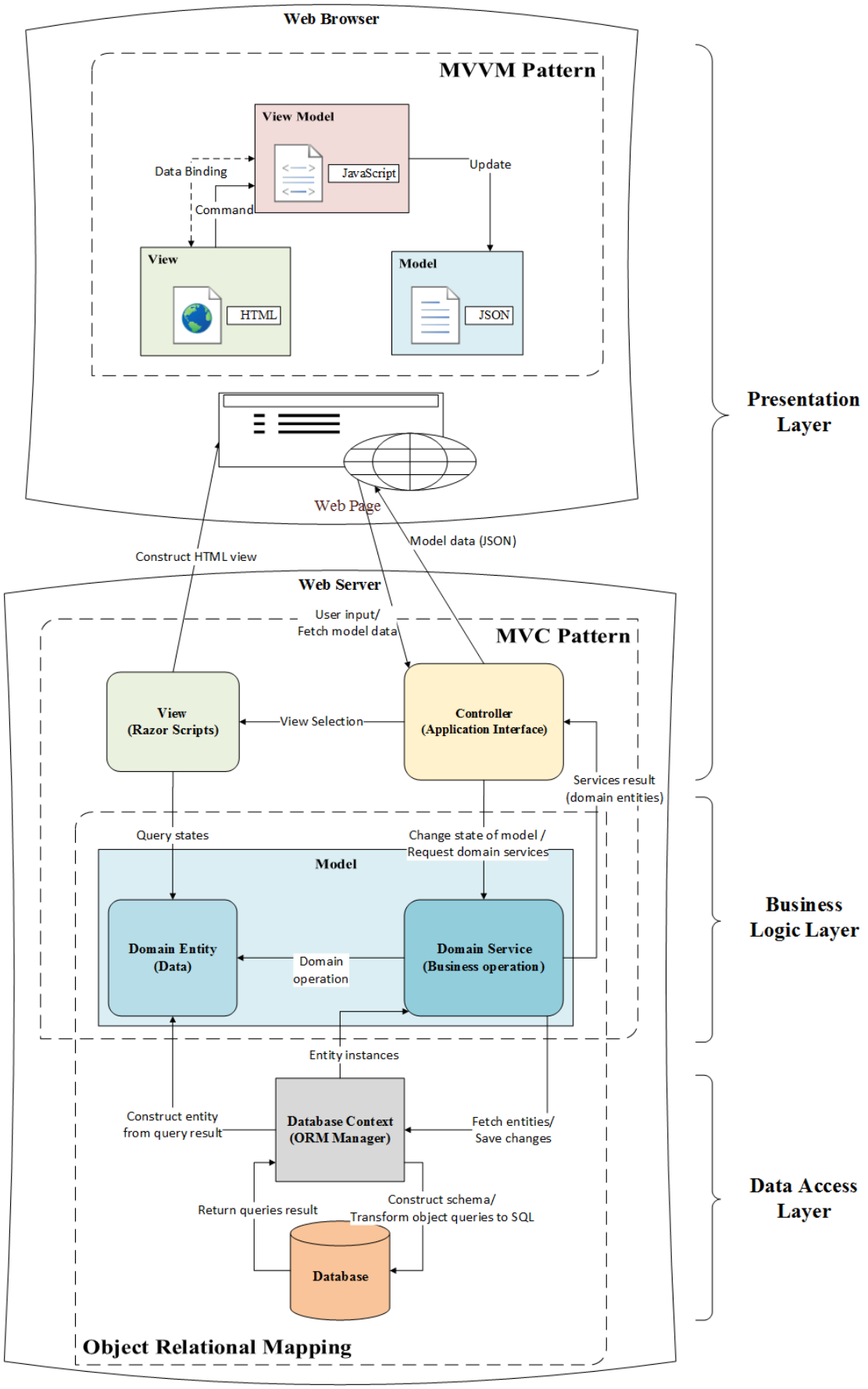

Figure 5.2: Overview of SRS implementation.

5.3.1.1 Domain Services

Generally, implementing *domain services* in SRS is about defining set of operations. The operations needed by the services are often reflected by the client who depends on it. In other words, they exposed the necessary business functions to the PL that interacting with these services. In any data-oriented application, most of the business operations will involve CRUD operations. However, in a more complex scenario, this will need sophisticated data processing logic. This section will concentrate on several important algorithmic of the business operations.

Most of the data processing regarding the CRUD operations involves integrating the business rules. Business rules are often aligned to domain problems and usually involve checking multiple domain entities at the same time. For example, it is meaningless to create redundant *recipe category* with same name. It depicts the algorithm required to process (and check for) the redundant name in the *recipe category*. The algorithm is applicable to creation of *Table* and *Staff* entity. The different being identifier of *Table* and username of *Staff* is checked instead.

Handling order submission is a complex operation that requires intensive data processing on multiple entity. When the system receives an order submission, it must evaluate whether the current material stock level can handle this order request. If the current stock level is unable to fulfil the order request, the system will return the cook-able quantity for each recipe to allow the waiter to informs the customer. If the order is submitted successfully, the system will then update the table status to busy, deduct the material quantity and send a notification to kitchen.

5.3.2 Presentation Layer (PL) Implementation

Implementation of the PL mainly focuses on the UI and presentation logic. The UI is often the main concern of users when using a software product. It usually gives the user the first impression as to whether they can accomplish their goal easily – which is termed usability. The UI of any good software should guide the users through series of tasks to achieve their goal. Initially this is addressed in §4.4.1, which formed the ideas regarding the required layout and content of UI. In this section, the design idea will be evolved into an actual UI for SRS. To address these issues, the system utilized two *Presentation Separation* patterns: *Model- View-Controller* and *Model-View-View Model*. The term, *Presentation Separation*

pattern, is just a generalized name for the all the design patterns that encourage separation of the UI related logic, from application logic and data. At this point, an interesting question may arise: Why are two patterns of similar goals required in this project?

To answer this question, the focus areas of both patterns are explained in the following sections.

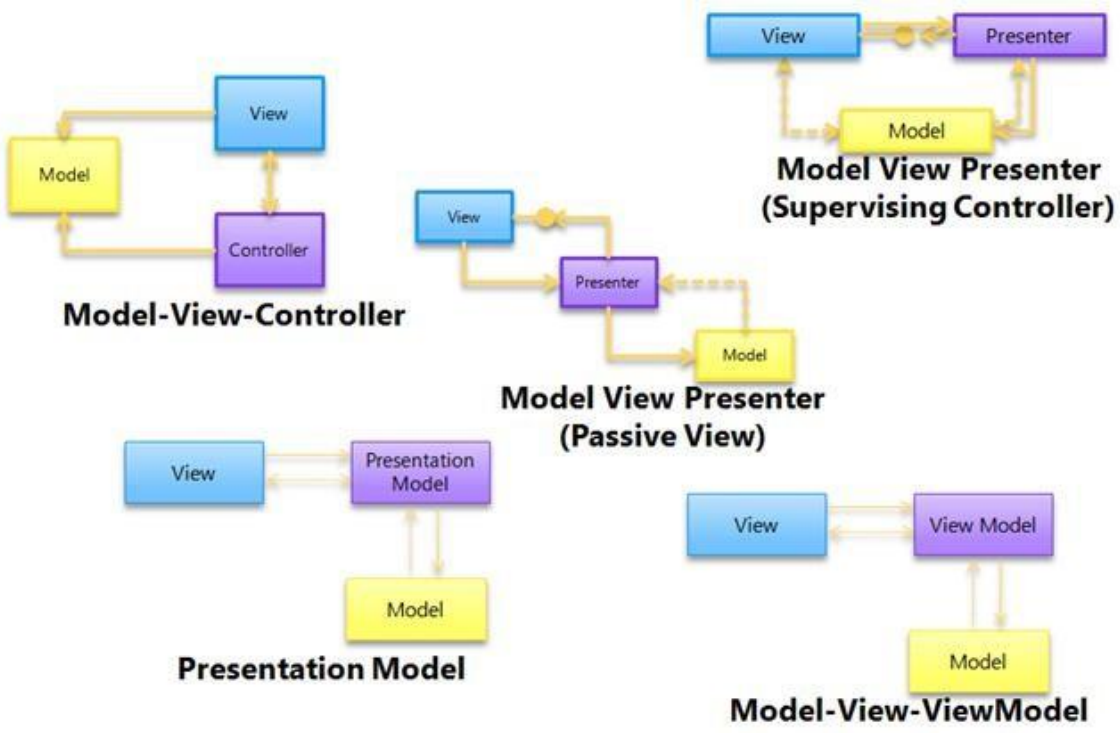

Figure 5.3: An overview of Presentation Separation patterns

5.3.2.1 Model-View-Controller (MVC)

Since Trygve Reenskaug introduced MVC pattern for Smalltalk application in late 1970, it has become one of the major practices in software engineering history pattern addresses the responsibilities of three major components in UI construction, they are as below:

I. *Model* refers to data and behavior of the application. It is responsible for providing the current state of its data and handle instructions for states change according the client requests;
II. *View* is the user interface that presents the state of data and manages the way they are presented; and
III. *Controller* captures the user inputs from user interface and manages the flows to update the model state and view information.

The clear segregation of their responsibilities results in several benefits :

- The logics of presentation (view), input (controller) and business process (model) could be changed and allowed to evolve independently, hence achieving separation of concerns;
- The view is decoupled from the model allowing multiple ways to present the same data that accommodate user concerns; and
- Loose coupling between the view and the controller also enhance the testability of the application.

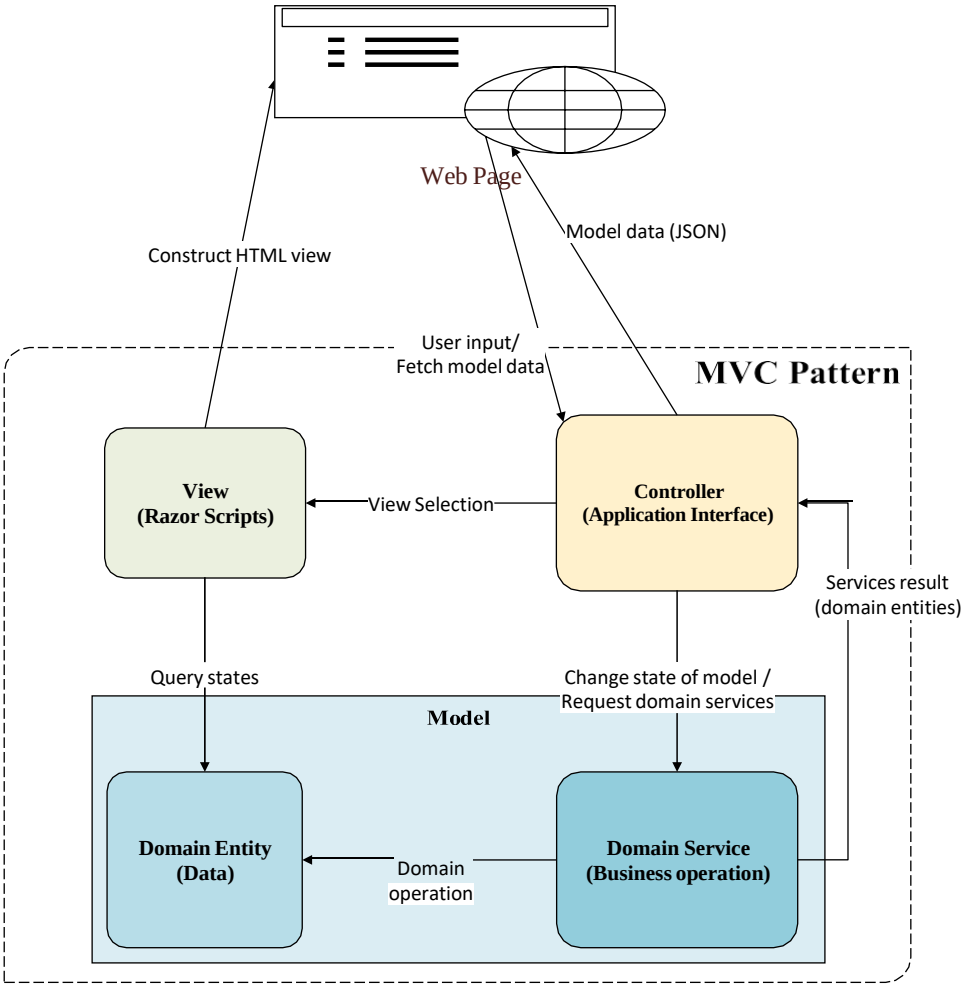

Figure 5.4: MVC pattern at Presentation Layer.

The request URL follows *"http://domain/{controller}/{action}/{parameters}"* pattern where

- *controller*, is the name of controller;
- *action*, refers to the request that user initialize such as view details, create new, etc.; and
- *parameters*, are the values that would need by controller to perform the action suchas order id.

When a HTTP request is routed to the *Controller* class, it finds the appropriate method to execute based on the pattern described above. The *Controller* class can provide overloaded version of the method depending on whether the request type (GET or POST). This is illustrated**,** which shows *Order Controller* class that maps the URL to the methods in the class. Next, the controller performs the method by invoking the *domain services* to retrieve or update the *domain entities*. It then selects the *Views* to be presented to the user. The selection of the view also follows the convention of the method name. For instance, the *Details* method will expect a file named "*Details.cshtml*" at the "Order" sub- directory within the "Views*"* directory at the application file structure. *Razor* code (view engine) is then used to construct the View content based on selected template and supplied data. The data refers to the *domain entity* of BLL in this context.

In summary, the implementation of SRS can accommodate changes easily by adopting the MVC pattern. The clean separation of concerns also allows individual part to be tested easily hence simplify the debugging experience. However, MVC pattern may address the problem of view creation on the server side but not at the client side. As specified once the view is served to the client side. JavaScript is used to introduce behaviours to the view. The JavaScript often needs to target a particular HTML element in the view and changes to view structure will easily break existing implementation. The problem of tight coupling between presentation and application behaviour still exits if they are mixed together at the client side. Hence, the second design pattern, MVVM pattern is used to address this issue.

5.3.2.2 Model-View-View Model (MVVM)

MVVM general goal and concepts similar to MVC are to enforce better *separation of concerns* among UI components and allows them to change easily. John Grossman first

introduces MVVM for building WPF[17] applications in his blog. Its name implies that they are similar to MVC but different in the sense that presentation logic and data is encapsulated in *View Model* (VM) rather than *Controller*. At client side of PL, they all also has different semantic compared to MVC. It shows an overview of MVVM pattern and description of its component are listed as below:

I. *Model* refers to JSON (JavaScript Object Notation) that mirrors the domain entity of server side;
II. *View* is the generated by HTML through the view engine of the server; and
III. *View Model*, written in JavaScript, is an abstraction of view that consists of View'sstate and behavior. It exposes the model's properties, commands and additional states to the view.

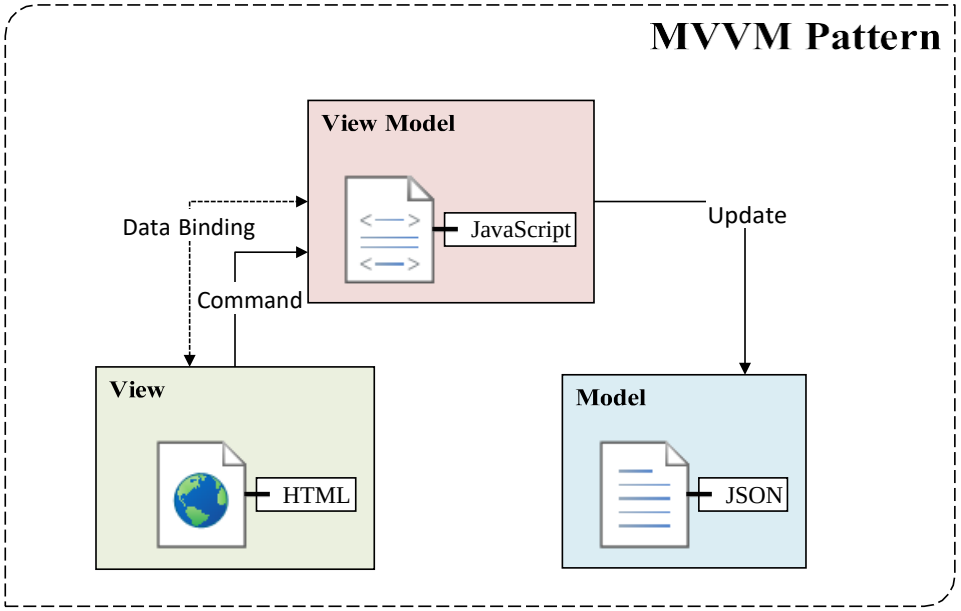

Figure 5.5: MVVM pattern at Presentation Layer.

One of the important differences between the MVC and the MVVM is that *ViewModel* does not directly reference the *View* as managed by the *Controller*. Instead, it leverages the data binding technology to bind the view to properties and functions of *ViewModel*. The properties often consist of the data contained in the model and other states specific to the view. When the properties change, the *View* that binds to the properties is updated. The functions of *ViewModel* are a set of commands that reflect application behaviour, for instance, adding a new order item to the order. In short, the state synchronization and command between the *View* and *ViewModel* are handled automatically by the data binding technology. Hence, it is a key enabler of this pattern.

It shows how the *KitchenViewModel* leverages *Knockout.js* to update UI dynamically based on the status of the Order. It contains two main properties: a list orders and their total count. In addition, the *ViewModel* exposes the *Change Status* function to update the order status. The *ViewModel* retrieves the orders data from the server throughAjax. These data are exactly one to one mapping with the domain entity from BLL (includingtheir hierarchy). Hence, the order data is are actually the *Model* in this pattern. When the *ViewModel* receives the data, it converts the necessary properties to observable to update them dynamically.

On the *View* side, these properties are bound to the relevant HTML elements through the "*data-bind*" attribute as shown in. The binding could be directly presenting thevalue of the *observable* property such as "*text*" or "*css*". It is also possible to introduce control flow such as "*foreach*". The "*foreach*" binding is a powerful binding that loops everyelement of *observable array* and creates the appropriate HTML elements for the property inthe array. When the item is removed from the array, so are the HTML elements. The "*click*"

binding deals with the click event on the HTML element and maps to the corresponding command (e.g. Change order status).

With this kind of binding, the *ViewModel* does not need to know about the structure of the *View*. The way of presenting the *View* could be changed any time and new behaviours can be added to *ViewModel* easily. It simplifies the development process without writing boilerplate code to synchronize the view. In addition, the application is capable of providing an interactive and richer user experience.

5.3.2.3 Remote Procedure Call (RPC) and Server Push

One of the important considerations at the client side processing is getting data for the view and sending data back to the server. In most cases, they are done through initializing a RPC with the web server. RPC is process of sending a request with parameters over the network to another environment where the required procedure will executed, the result is then returned to the caller. While the caller process is waiting the result, other processes can continue to execute. RPC is necessary in SRS because it is constantly committing data to the server based on user inputs.

The RPC is often use as pull model – which the request is initiated by the client (web browser). In order to obtain the latest data from the server, the client constantly needs to start

a new request to pool server data. As the number of clients increase, significant resources of the server will be consumed. To address this issue, the *Server Push* approach is adopted. *Server Push* is a technology that allow the server to push data without required to start a new connection. It mainly operates around the publish-subscribe model to deliver data to interested clients. In combining both RPC and *Server Push* approach, the system could achieve real time communication among the connected clients. This is particularly useful for order notification among waiters and chefs. In SRS, these approaches are achieved by *Ajax* and *SignalR* respectively.

The order notification in SRS heavily relies on *SignalR* to push the new order notice and status change notification to the waiter and chef. It starts by defining the order received functions at the client side. At the server side, an *OrderHub* class that handles client subscription is defined. When a new order is received at the

OrderController, the *OrderController* will access the Hub class and invoke the services defined at client side .

Both *Ajax* and *SignalR* use the same data structure, JavaScript Object Notation (JSON), to carry the data back and forth between client and server in SRS. JSON is a format for data-interchange that takes forms in name values pairs. JSON formatted data has a smaller data payload compared with XML, thus making it a good option for data transfer. Particularly in modern web application, JSON is increasingly popular as the communication

medium. Sending data from the client side to the server side involves serializing the JavaScript object into JSON string. At server side, the received string is then de-serialized and parsed to a compatible form that is usable to the system. In ASP.NET, the conversions of JSON string into domain entity object are automatically handled by model binding of the framework. On the other hand, converting the domain entity into JSON string involves serializing the class's public properties. The framework also handles this conversion.

5.3.2.4 Security

The project involves multiple users and their details are stored within the system. The data may be access and tampered with by an external party if security is not ensured. Hence, the implementation of the system needs to take precautions when considering security issues which could be detrimental to the user.

The project mainly controls the user access through *role-based authentication*. The *Controller* is aware of the viable (allowed) roles that have access to the system data. When the user tries to submit a request to the *Controller*, his or her role will be verified. If the user does not incorporate the specified roles, the system will route the user back to the login screen and inform them that the request is not authorized.

Utilizing ORM for the database access also shield the system from SQL Injection. SQL Injection is essentially passing malicious code in the string of SQL code as parameters, which in turn is executed at the server and yields destructive results. The SQL queries generated by ORM framework is parameterized queries. This helps to escape the malicious code from the request parameter string before it is passed to the server. The system also encrypts sensitive data such as password and credit card details before storing to the database. The encryption solution is provided by ASP.NET hence it is tested and trustable.

5.4 Walkthrough

In order to demonstrate the usage of the system, a series of walkthroughs will be presented. This section will cover the important process of restaurant operations, starting from food order, kitchen preparation and payment made.

5.4.1 Submitting Order

The first time the waiter accesses the application, the system will prompt the user to login into the system. After the waiter, successfully logs in, the waiter can then expand the navigation menu on the top right to access the order view, as illustrated by. The order view in presents orders submitted on that day and their status. At this view, the waiter initiates order submission by selecting the *Submit New* button.The system will then present the view to construct the order.

As shown, the user identity is automatically associated with the order. The waiter will first select a table for the new order. The waiter can then browse the menu recipesby selecting the *Browse Menu Recipes* tab. The tab shows all the menu recipes that are currently available for order. It also allows the waiter to filter the list by using *Browse by Menu* dropdown box. The waiter then can select the *Add* link to order that recipe. The selected recipe will remove from the available menu list and bring the waiterback to order details tab. The waiter

can then specify the quantity and comment on each item.

5.4.2 Updating Order Status

The Kitchen view, as shown in, presents the order information side by side. When the chef want to prepare the meal for the order, the chef first updates the order statusto *Preparing* . This will change the colour of the order across all clients viewing kitchen views in real time. This is to prevent the same order being attended to by multiple chef.

When the order has been prepared, the chef will then select the *Completed* button. The selected order will be removed from the kitchen view, as shown in. These changes also synchronize across all connected kitchen views. In addition, the system will send a notification to the waiter to inform them that the order is ready to be served.

5.4.3 Processing Payment

In order to process payment, the cashier will first present the bill to the customer through Payment view. The Payment view will list the order that is currently unpaid, as shown in. Selecting the *Bill* button will send the order details to the printer, an example ofwhich is shown in. The cashier can then select the pay button to record the payment details. Record payment view will also present the bill information in case the cashier needs to reference the order details. The payment method could be cash or credit card, as shown in. Finally, the receipt will be dispensed to the customer.

5.5 Concluding Remarks

This chapter has covered the underlying concepts and technologies to build the SRS system. Segregating the implementation into different software layers has proven useful to underpin the principle of separation of concern. It allows the author to focus a smaller set of problems when implementing each layers. This reduces the complexity of the implementation process while enhancing the implementation design.

The next chapter discusses testing process of the system. It describes how the functionalities of the system could be verified and tested.

Chapter 6. Testing

This chapter describes the software testing concerns of the project. It covers different testing approaches that adopted in the project and some of the techniques applied to realize this process. The chapter first provides an overview of software testing. Then, the processes of each adopted testing approaches are discussed in depth.

6.1 Software Testing

Software testing is the process of verifying software implementation work against its requirements. This is meat to inform the developer that possible errors are present before the software artefact is utilised by the actual user – final stakeholder or client. Software testing generally has two main objectives:

- To ensure that the implementation is as intended and the requirements haves been met; and
- To uncover system defects, including those incorrect, undesirable or inconsistent to requirement behaviours.

Software testing could also be viewed as a key contributor to software quality. A high quality software simply means a product that has high user satisfaction while maintaining a low defects rate. Software that has gone through rigorous testing will be likely to yield a better quality product.

Testing was often left to the last phase of development in traditional software development methods. If however this is a complex software artefact, testing is required at the start; as it requires significant effort to fix the bugs at the end of the project. The complexity of bugs is greater when they are accumulated across the different parts of the system, resulting in more time being required to analyse and investigate how they occurred – and then find a solution. In SRS, testing is done before or in parallel with the actual implementation to ensure that the features are worked as intended – this could be viewed as design for testability. This approach adheres to the test first principle of XP where the sooner the testing concerns are addressed, the better, as the developer will have a clearer understanding of what is to be expected and integrated into the test cases

Testing can be viewed from two broad perspective: *functional testing* and *structural testing. Functional testing,* sometimes referred to black box testing, is testing on the functionality of the system based on the specified requirement. The test itself has little knowledge about the

testing target's internal structure. In contrast, *structural testing*, also known as white-box testing, involves examining the internal implementation. It tests the design used by the implementation to verify it correctness

6.2 Test Automation

Software testing is a repetitive and monotonous task because it involves a repeating cycle of performing action and verifying result. Particularly when the number of items needed to test grows, doing the testing manually could be time consuming and not effective. Hence, most software testing utilizes tools to automate the testing process. A test automation tool could make testing more efficient and quicker by automatically executing the set of defined test cases and verifying their result. This involves writing the scripts to define these steps.

While plenty of unit testing frameworks are available for VS, this project uses the default unit testing framework of VS, known as *MS Test*. *MS Test* creates templates for typicalunit testing flow. It allows the developer to define the initialization code at class level or method level. When initialization of the code is undertaken at class level, it executes once the class is instantiated. At method level, the code will be executed for each test method within the class. This helps to avoid redundant boilerplate code for initializing the data required for testing.

6.3 Regression Testing

Iterative developments of software products often introduce new changes to existing system behaviours and interfaces. However, the degree of their impact on existing systems could be difficult to estimate. Current features may stop working or new bugs may emerge after these changes. This is again aligned to the *continuous integration* principle that each changes should be properly verified before integrating them into the code base.

The conducted test cases are an excellent medium to document existing system behaviours. They encapsulate previous assumptions and concepts regarding the system. If they fail after new changes, it means that the system behaviour has deviated from previous intention. This is an excellent time to revise the requirements before proceeding further. In addition, the testing may failed because of implementation errors or poor design. This provides additional chances to improve the system design rather than let it perform poorly when shipped to the user.

Regression testing is naturally a part of the testing process in the project. It can be executed at a different level depending on the purpose. Rerunning unit tests and integration tests are the most common scenarios when the developer is implementing a feature. When the development reaches a stable stage and is ready for release, it often involves full scale regression testing. In SRS, full scale regression testing is especially important at the endof each prototype version.

6.4 Integration Testing

Integration Testing is testing multiple components which work together. Often, these components have been tested individually before the integration test. The concerns of integration testing are aligned to testing the interfaces of components and their interaction. This also investigates the techniques used for data exchange among components. It helps to uncover issues such as exposing invalid interface or that the data passed is not in a compatible format. *Integration testing* also verifies the control flow of these components' interaction and ensures that they are in correct sequence. In the context of SRS, *integration testing* has two main objectives:

- Testing interaction across multiple software layers; and
- Testing coordination among functional modules.

The system contains software packages that are distributed across software layers due to its three-tiered architecture. Testing interaction across multiple layers is essential to ensure that they expose the correct interface and the received data seamlessly at either end. This project utilizes a bottom-up approach where the components at the lowest level are tested first. In this case, the DAL is tested with an actual database first. Testing with the actual database is particularly important to check if data persistence takes place correctly. Switching to use the actual database is relatively easy, it involves replacing the database context class used in Unit Testing. Next, the BL layer was tested with the PL to ensure that it exposed necessary operations used for client side and view data. This type of testing is done by testing Controller operations with actual BL services and communicating with the real database.

6.5 System Testing

System testing is testing the fully integrated system as a whole. It is the testing phase after *integration testing*; but undertaken on all system components. It aims to discover undesirable

behaviours when all the components are working together. It also checks if the system conforms to the requirements and expectations. In addition, it also helps to understand the limit of the system and ensures that the system is reliable. At this stage, *unit testing* and *integration testing* have addressed most of the functionality concerns. Hence, *system testing* in SRS focuses on the non-functional aspects of the system.

6.6 Security Testing

The open access nature of web application could be a threat to the system if security concerns are involved. It can be accessed easily by devices with a web browser within the connected network. This opens up the possibility of unauthorized access and malicious security threat that would comprise the system. Hence, *Security Testing* is performed on the system to review the degree of reliability and safeness.

The security testing ultimately is concerned with the user authentication and authorization. It first tests the login process of the system and ensures that proper credentials are required to access the system. Both valid and invalid login credentials are tested and their login results are compared. As discussed, the system utilizes *role-based authentication* to restrict the accesses of users. It will present only the navigation link for particular functionsif the user has a role associated with their account. This feature is tested by assigning roles to the user account and verifying if the access is granted correctly. Then, the roles are removed from the user account and tested if access is denied.

6.7 Performance Testing

Performance testing is another testing focus with the system's reliability and capability to withstand an intense load.

Since the prototype system is targeting restaurant environment, the estimated amount of concurrent user access is lower compared than ordinary Internet applications. However, understanding the limits of the system is essential because it should not fail when it is needed most, especially during peak time. Hence, this project utilized the *Web Performance Testing* and *Load Testing* tools provided by VS to execute performance testing.

The testing environment is one of the crucial factors in performance testing because a high performance hardware will definitely give a better result. In this project, the testing environment is same as the development environment. The test machine is considered to be excellent for data processing due it's quad-core processor. Having 8GB of memory is

sufficient in handling load for small to medium restaurant, though, the system will prefer a higher memory capacity in a large restaurant.

After the configuration is completed, the Load testing is ready to be executed. It shows the load testing is executing on the test machine. The load testing tools provide informative graphs and statistics regarding the current system performance. As for web application, the page load time is the most relevant statistic because it tells the developer which page need to be optimized for performance. When the execution completed, a report of the testing result will be presented to the tester, as shown in. The report's *TestResults* section indicates that the system survived the load testing, as it shows none of the order submission flows have failed. It also shows the slowest pages that need developer attentions.

6.8 Concluding Remarks

This chapter has documented the extensive testing approaches that were utilized in this project and the process of how they were carried out. Different and all-encompassing testing approaches have been conducted to ensure the system is as robust as possible. The system has proved to be functional and usable after all the testing was done. The chapter has marked the end of the software development process of the project.

While rigorous testing processes have covered (tested) many aspects of the system, the testing activities are likely to carry on throughout the system life cycle. This is aligned to the popular software engineering mantra by Dijkstra,

There will be errors that not have yet been discovered by testing and will only emerge during user usage. Nevertheless, the project is structured in a way that will enable changes and fixes to be easily undertaken, hence their risks have been mitigated.

The next chapter is the final chapter of the report. It concludes the report of the project and evaluates approaches taken to complete the project.

Chapter 7. Conclusion

This chapter concludes the overall process of the project. It looks back at the work completed and evaluates the research and approaches taken. Finally, it marked the end of the report by a summary.

7.1 Project Achievement

The project has gone through a series of activities to develop a complex solution for the computerized restaurant system. After analysis of the project's goal and research direction, a set of objectives were established, as specified in §1.3. All the activities done during the project were attempts to realize these objectives. At the end of the project, the developed prototype software has fulfilled these objectives by the following means:

- Objective #1 was satisfied by implementing the prototype system with three-tier architecture and presentation separation pattern.
- Objective #2 was addressed by utilizing Hierarchy Task Analysis (HTA) to model user interface's presentation and behaviour.
- Objective #3 was satisfied by integrating Responsive Web Design (RWD) to allow mobile friendly access to the system.
- Objective #4 was satisfied by adopting Remote Procedure Call and Server Push technology for real-time communication between client and server.
- Objective #5 was addressed with various testing approaches to ensure the prototype system is as robust as possible.

Employing the agile development method also proved useful in managing the software development process. The software prototype was constantly evolving thanks to the incremental and iterative development cycle. It was tested at every iteration, hence most defects were addressed early on in the project. Besides, it reduces the scope of implementation and allows the author to manage the development activities effectively. The project management techniques are discussed in §3.3 which also demonstrates their values for time and workload management. The project plan was constantly revised to reflect the progress and capability of the author based on these techniques.

The project was ambitious and time-consuming, implementing as many features as possible within the very limited timeframe. It has successfully satisfied the Functional Requirements (FR) from FR1 to FR14 and all Non-functional Requirements (NFR) of the system. These requirements have top priority and reflect the most needed features required by stakeholders. FR 15 through 19 are not implemented due to time constraints. However, they are the lower priority features that are pleasant – but not paramount. Their absence would not result in major operational issues in restaurant order processing. As the system was designed to be easily extendable, these features could be implemented in the future. At the bottom line, the system is useable in term of the stakeholder's need and operational concerns. It satisfied the basic goal of replacing the paper-based system in the restaurant operations.

7.2 Future Improvement

In addition to the unfinished requirements, there are other possibilities of further improving the project. The improvements may include:

1. Presenting graphical floor plan for table management and reservation;
2. Support food order delivery and driver tracking;
3. Extension of pricing methods for individual or multiple recipes;
4. Advanced inventory control with material storage and expiry information; and
5. Managing customer loyalty membership and discount voucher.

Another interesting possibility is to host the entire system on Cloud-based services. If the restaurant business model expanded to multiple outlets, the restaurant manager could access the data of different restaurants to view their performance reports or order materials from suppliers.

7.3 Concluding Remarks

This chapter has concluded the report of this project. The project successfully implemented a working complex prototype of a Smart-Resto System. The implemented prototype software has been fully tested throughout the project phases and it demonstrated acceptable performance. Overall, the project enabled the author to completed most of the high priority requirements. This report also documented all the relevant research details and decision-

makings processes. If future extensions of the system are undertaken, the report will be helpful in assisting the completion of the remaining requirements and future improvements that might be involved. In summary, the project has satisfied its objectives and fulfilled its purpose to assist restaurant operations.

References

[1] POSitive Technologies INC. (2013). *The Past, Present and Future of POS* [Online]. Available: http://www.positivetech.com/2013/02/27/the-past-present-and-future-of-pos/ (Last Accessed: 20/4/2013).

[2] touchPOS.net. *History of POS Systems* [Online]. Available: http://www.touchpos.net/page.html?chapter=10&id=9 (Last Accessed: 20/4/2013).

[3] G. Bisson. (1998) Getting Down To Business : Using The ST In An IBM World. *STart*.

[4] T. Shimmura, T. Takenaka, and M. Akamatsu, "Real-Time Process Management System in a Restaurant by Sharing Food Order Information," in *Soft Computing and Pattern Recognition, 2009. SOCPAR '09. International Conference of*, 2009, pp. 703-706.

[5] K. J. Patel, U. Patel, and A. Obersnel, "PDA-based wireless food ordering system for hospitality industry — A case atudy of Box Hill Institute," in *Wireless Telecommunications Symposium, 2007. WTS 2007*, 2007, pp. 1-8.

[6] Y. H. Huo, "Information technology and the performance of the restaurant firms," *Journal of Hospitality & Tourism Research*, vol. 22, pp. 239-251, 1998.

[7] R. Leung and R. Law, "Evaluation of Hotel Information Technologies and EDI Adoption: The Perspective of Hotel IT Managers in Hong Kong," *Cornell Hospitality Quarterly*, vol. 54, pp. 25-37, February 1, 2013 2013.

[8] T. Tan-Hsu, C. Ching-Su, and C. Yung-Fu, "Developing an Intelligent e-Restaurant With a Menu Recommender for Customer-Centric Service," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 42, pp. 775-787, 2012.

[9] C. Soon Nyean, C. Wei Wing, and Y. Wen Jiun, "Design and development of Multi-touchable E-restaurant Management System," in *Science and Social Research (CSSR), 2010 International Conference on*, 2010, pp. 680-685.

[10] E. W. T. Ngai, F. F. C. Suk, and S. Y. Y. Lo, "Development of an RFID-based sushi management system: The case of a conveyor-belt sushi restaurant," *International Journal of Production Economics*, vol. 112, pp. 630-645, 4// 2008.

[11] T.-H. Chen, H.-H. Lin, and Y.-D. Yen, "Mojo iCuisine: The Design and Implementation of an Interactive Restaurant Tabletop Menu," in *Human-Computer Interaction. Towards Mobile and Intelligent Interaction Environments*. vol. 6763, J. Jacko, Ed., ed: Springer Berlin Heidelberg, 2011, pp. 185-194.

[12] C. Rich, "Building Task-Based User Interfaces with ANSI/CEA-2018," *Computer*, vol. 42, pp. 20-27, 2009.

[13] D. Ansel and C. Dyer, "A Framework for Restaurant Information Technology," *Cornell Hotel and Restaurant Administration Quarterly*, vol. 40, pp. 74-84, June 1, 1999 1999.

[14] C. R. Oronsky and P. K. Chathoth, "An exploratory study examining information technology adoption and implementation in full-service restaurant firms," *International Journal of Hospitality Management*, vol. 26, pp. 941-956, 2007.

[15] M. D. Olsen and D. J. Connolly, "Experience-based Travel How Technology Is Changing the Hospitality Industry," *Cornell Hotel and Restaurant Administration Quarterly*, vol. 41, pp. 30-40, 2000.

[16] J. Lukkari, J. Korhonen, and T. Ojala, "SmartRestaurant: mobile payments in context-aware environment," in *Proceedings of the 6th international conference on Electronic commerce*, 2004, pp. 575-582.

[17] H. Xu, B. Tang, and W. Song, "Wireless Food Ordering System Based on Web Services," in *Intelligent Computation Technology and Automation, 2009. ICICTA '09. Second International Conference on*, 2009, pp. 475-478.

[18] H. W. Gellersen and M. Gaedke, "Object-oriented Web application development," *Internet Computing, IEEE*, vol. 3, pp. 60-68, 1999.