ABSTRACT

Overhauling Legacy Enterprise Software Applications with a
Concept Refinement Process Model

by

Daniel P. Knight

January 2013

Director of Thesis: Dr. Nasseh Tabrizi

Major Department: Computer Science

Currently, there are many legacy enterprise software applications in active deployment that are outdated. These large legacy applications are rapidly becoming less practical for both the organizations they service, and for the organizations responsible for servicing them. Due to this problem, organizations utilizing legacy enterprise software applications are looking for feasible methods for overhauling them. This thesis establishes a process model for refining the initial concept associated with overhauling legacy enterprise software applications, and examines a case study of that process as applied to a real-world legacy software system.

Overhauling Legacy Enterprise Software Applications with a Concept Refinement Process Model

A Thesis

Presented to the Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Software Engineering

by

Daniel P. Knight

January 2013

Copyright © 2013

Daniel P. Knight

Overhauling Legacy Enterprise Software Applications with a
Concept Refinement Process Model

by

Daniel P. Knight

APPROVED BY:

DIRECTOR OF THESIS: ____

M. H. Nassehzadeh Tabrizi, PhD

COMMITTEE MEMBER: ____

Junhua Ding, PhD

COMMITTEE MEMBER: ____

Sergiy Vilkomir, PhD

CHAIR OF THE DEPARTMENT OF COMPUTER SCIENCE:

____

Karl Abrahamson, PhD

DEAN OF THE GRADUATE SCHOOL:

____

Paul J. Gemperline, PhD

TABLE OF CONTENTS

List of Figures viii

List of Tables x

Chapter 1: Introduction 1

Chapter 2: The Overhaul Concept Refinement Process Model 3

2.1: Understand the Existing Issues 6

2.2: Research Software Development Trends and Technology 9

2.3: Identify Remedies for Existing Issues 11

2.4: Collectively Analyze and Conceptualize Solutions 12

2.5: Select a Solution to Refine 13

2.6: Refine Solution with Rapid Prototyping 15

2.7: Final Preparation for Software Development 16

Chapter 3: Case Study 18

3.1: Understanding the Existing Issues 19

3.2: Researching Software Development Trends and Technology 25

3.3: Identifying Remedies for Existing Issues 26

3.4: Collectively Analyzing and Conceptualizing Solutions 28

3.5: Selecting a Solution to Refine 30

3.6: Refining Solution with Rapid Prototyping 31

3.7: Final Preparation for Software Development 39

3.8: Projected Benefits 39

Chapter 4: Overhaul Concept Refinement Process Model Analysis 41

4.1: Domain 44

4.2: Flow 48

4.3: Sub Processes..52

4.4: Artifacts 55

4.5: Usability..57

4.6: Repeatability 57

4.7: Limitations..58

4.8: Related Works..59

4.9: Conclusion 62

References..63

Appendix A: Background Information 66

A.1: Cloud Computing..66

A.2: Service Oriented Architecture..66

A.3: Representational State Transfer 67

A.4: Mobile Web 69

A.5: Hypertext Markup Language 5 70

A.6: JavaScript Object Notation 71

A.7: Extreme Programming..72

A.8: Classic Active Server Page Technology 75

Appendix B: Service Central's Client-Side JavaScript Object Constructor.............77

Appendix C: Service Central's Server-Side VBScript URI Controller 79

Appendix D: Service Central's Collection Level HTTP Request 87

Appendix E: Service Central's Element Level HTTP Request 91

Appendix F: Service Central's Mobile App. Screenshots 93

Appendix G: Service Central's Friendly Web Service Interface....101

Appendix H: Service Central's URI Collections Hierarchy....104

LIST OF FIGURES

Figure 1: Overhaul Concept Refinement Process Model 5

Figure 2: Issue Discovery Process Model..8

Figure 3: Research Planning Process Model 10

Figure 4: Remedy Discovery Process Model..12

Figure 5: Conception Process Model..13

Figure 6: Solution Selection Process Model...14

Figure 7: Rapid Prototyping Process Model...15

Figure 8: Prototype Evaluation Process Model 17

Figure 9: Legacy Architecture 20

Figure 10: Symptoms Document 21

Figure 11: Issues Document..22

Figure 12: Cross Reference Document 24

Figure 13: Research Plan Document...26

Figure 14: Remedies Document..27

Figure 15: New High-Level Architectural Concept..28

Figure 16: New High-Level Development Process Concept...................................29

Figure 17: Prioritized Solutions Document 31

Figure 18: Sample User Stories - Login 32

Figure 19: Sample User Stories - Landing..33

Figure 20: Sample CRC Card - Service..34

Figure 21: Helicon Configuration File..34

Figure 22: Sample Unit Test – Parse URI’s Collections & Elements 35

Figure 23: Sample Unit Test – Parse URI's Query String....................................35
Figure 24: Integration Test 1....................................36
Figure 25: Integration Test 2....................................36
Figure 26: System Test 1....................................37
Figure 27: Code Snippet – AllRegs.html....................................37
Figure 28: System Test 2....................................38
Figure 29: Code Snippet – Reg10.html....................................38
Figure 30: Enterprise Software Lifecycle Model....................................48
Figure 31: Hierarchical Command Structure of the United States Marine Corps.....53
Figure 32: Hierarchical Structure of Strategies and Tactics....................................54
Figure 33: Pragmatic REST Constraints....................................69
Figure 34: Sample JSON Structure....................................71
Figure 35: User Login Screen....................................93
Figure 36: Main Menu Screen....................................94
Figure 37: RMA (Return Material Authorization) Screen – RMA# 382....................................95
Figure 38: RMA Screen, Entities Section – Two Entities for RMA# 382....................................96
Figure 39: RMA Notes Screen, Notes Section - Modal Dialog for Select Note.......97
Figure 40: RMA Note Screen, Notes Section – "Add Note" Button....................................98
Figure 41: RMA Add Note Screen....................................99
Figure 42: RMA Note Screen, Notes Section – New Note Added....................................100
Figure 43: FWSI Architecture....................................103
Figure 44: URI Hierarchy....................................104

LIST OF TABLES

Table 1: Service Central's Projected Customer-Side Stakeholder Benefits 40

Table 2: Service Central's Projected Development-Side Stakeholder Benefits 40

Table 3: Aspects of Legacy Software Applications 41

Table 4: Customer-Side Stakeholder Benefits 42

Table 5: Development-Side Stakeholder Benefits 42

Table 6: Risks in Overhauling Legacy Software Applications 43

Table 7: Risks Mitigated / Overhaul Concept Refinement Process Model Phase 44

CHAPTER 1: INTRODUCTION

Currently, there are many legacy enterprise software applications in active deployment that were initially designed and built over a decade ago. Over time, many of these software applications have grown to the point where they are too difficult to maintain, integrate with other software applications, and configure to effectively meet customer requirements. According to an article in the *Journal of Systems and Software*:

> In the last decade, we have seen an increasing use of both the object-oriented paradigm and distributed systems. As a result, there is increasing interest in migrating and reengineering legacy systems to these new hardware technologies and software development paradigms [1].

Legacy enterprise software applications, in many cases, cannot be easily replaced by modern software applications because the organizations using them have become locked-in to them. One reason for this is because they provide business critical functionality. According to an article in *Information and Software Technology*, "Legacy systems typically form the backbone of the information flow within organizations and are the main driver to consolidate information on their business." [2]. Additionally, organizations may become locked-in to legacy enterprise software applications because migration to new applications is not feasible or possible [3]. This is because data migrations are often highly complex, time consuming, error prone, and incomplete. Unfortunately, an organization that is locked-in to a legacy enterprise software application is ill suited to remain competitive because their software solution is built upon technology that has become outdated. However, despite the aforementioned issues, legacy enterprise software applications do continue to offer a limited amount of value because they perform necessary business functions, even if they do not perform these functions as well or as diversely as they could, or as they necessarily should.

According to John R. Leary, “Even when precedents can be used to foster common understanding of objectives and methods, large-scale systems pose exponentially greater difficulty in communication than is the case in smaller systems.” [4]. Therefore, when a legacy enterprise software application exceeds a certain age, size, and complexity threshold, the need to migrate the software application to a different or more modern software architecture, built upon modern technologies, may become a challenging necessity. This type of migration can be considered an overhaul of the legacy enterprise software application.

In order to successfully overhaul a legacy enterprise software application, a process must be executed [5]. According to researchers at the University of Sannio Palazzo Bosco Lucarelli, “Making a decision about how to evolve a legacy system cannot be made spontaneously; rather, it requires a decisional framework that takes into account several factors including software value, risk analysis, and cost estimation” [6]. The focus of this thesis is to present an *Overhaul Concept Refinement Process Model* that can be used to help accomplish the overhaul of legacy enterprise software applications.

This thesis is organized as follows. First, we will present the *Overhaul Concept Refinement Process Model* graphically in the form of a flowchart. Each node in the flowchart will be sufficiently detailed in its own subsection in Chapter 2. Next, we will examine a case study that exemplifies the utilization of the proposed process model in a real-word context. Finally, this thesis will be concluded with a chapter committed to analyzing the *Overhaul Concept Refinement Process Model*. In this chapter, subsections will be specifically devoted to analyzing the model’s domain, organizational flow, and artifacts.

CHAPTER 2: THE OVERHAUL CONCEPT REFINEMENT PROCESS MODEL

Unique problems require unique solutions. For example, the Chief Technology Officer (CTO) of an organization that initially developed, and now maintains, a legacy enterprise software application may determine that the organization's legacy software application needs to be overhauled to remain competitive. However, the CTO is not a technical expert on the legacy software application, and therefore does not truly understand the full extent of the existing issues. This renders him incapable of selecting a new *Software Architecture* and *Software Development Process* that is appropriate (and necessary) to achieve his overhaul initiative. Therefore, the CTO calls a meeting and announces to the software development, deployment, and maintenance staff that they need to overhaul their legacy software application in order to remain competitive. However, before they begin the full-scale software development effort necessary to attain the overhauled software product, he needs his staff to provide him with confidence that the right software product will be constructed via the most appropriate software development process, and that it can be feasibly accomplished within a reasonable amount of time. In this example, the CTO challenged his staff to refine his overhaul concept of their organization's legacy enterprise software application to determine what needs to be built, how it needs to be developed, and whether or not it can be feasibly accomplished. This is the type of unique problem that the *Overhaul Concept Refinement Process Model* intends to help solve.

Every software application lifecycle begins with a conception phase, in which the most high-level concept of a software application is conceived. The concept is then refined to a necessary level, where it becomes a worthy prospect for realization via a software development process [5]. It is within the conception phase of the software application's lifecycle that our

proposed *Overhaul Concept Refinement Process Model* exists. The purpose of the process model is to facilitate the refinement of a specific type of initial concept that is often encountered by software development organizations that support legacy enterprise software applications, and to evolve that initial concept to a level of maturity that makes it worthy of realization. The specific initial concept that we are referring to is the perception that a particular legacy software application need be overhauled. That is, some large business critical software application that is currently servicing an organization has become outdated and needs to be overhauled to remain competitive.

A typical software lifecycle model consists of the following high-level phases:

1. Conception
2. Requirements
3. Design
4. Implementation
5. Testing
6. Deployment
7. Maintenance
8. Retirement

A software lifecycle model that includes an overhaul may have the following phases [5]:

Initial:

1. Conception
2. Requirements
3. Design
4. Implementation
5. Testing
6. Deployment
7. Maintenance

Overhaul:

8. Conception
9. Requirements
10. Design
11. Implementation
12. Testing
13. Deployment

14. Maintenance

The primary difference between phases 1 and 8, in the above list, is that phase 1 is extensively focused on the functions of a software application (i.e., what the software will do), while phase 8 should be focused on enhancing non-functional elements of the existing software application and how it can perform its present functions more effectively. When overhauling an existing software application verses developing a new software application, much less effort must be expended gathering and identifying all necessary functional requirements via communication with the end-users, customers, and other stakeholders. This is because during an overhaul, the existing software application acts as roadmap for the identification and documentation of the functional requirements. Therefore, the general idea is to expend a suitable amount of effort establishing a workable concept from which a high quality software product can be created. Overhauling is not an overall restoration effort; overhauling is an overall quality improvement effort that aims to ensure the product's ongoing value. Figure 1 illustrates the high-level phases and flow of the *Overhaul Concept Refinement Process Model*. It is not a *Software Development Process*. It precedes the execution of a *Software Development Process* and enables the confident selection of a practical *Software Architecture* and *Software Development Process* that is best suited to the problem and concept.

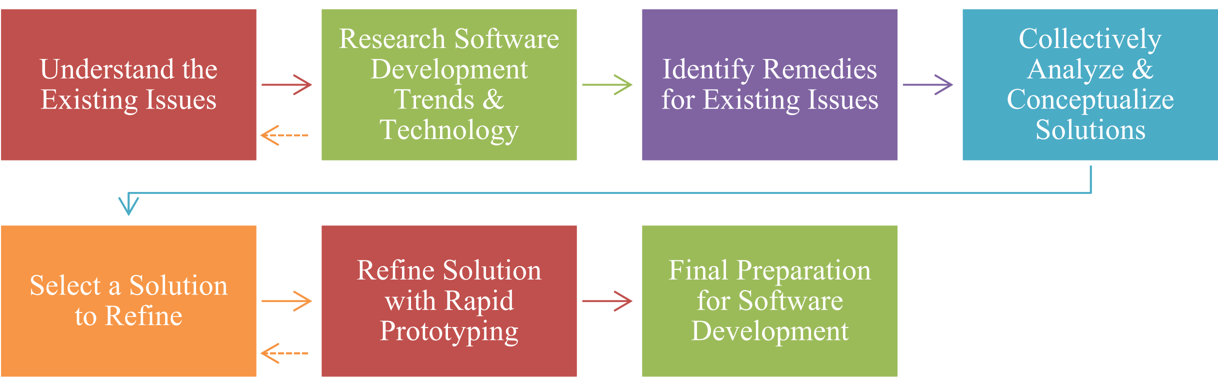

Figure 1: *Overhaul Concept Refinement Process Model*

The next seven sections (2.1 – 2.7) will each examine a phase of the *Overhaul Concept Refinement Process Model.*

2.1 Understand the Existing Issues

The first phase in the *Overhaul Concept Refinement Process Model* involves thoroughly comprehending the legacy software application's existing issues as a team. Issues are problematic elements of a software application and its lifecycle. While certain intrinsically obvious issues will always be the compelling force behind the initiation of the overhaul process, many issues that must be understood by the entire team should be discovered through collaborative investigation. Issues are not limited to a legacy software application's functional capabilities. Issues are found in both the functional and non-functional elements of the legacy software application [7].

Functional elements of a software application are those that directly translate to its features. For example, many software applications have a login feature that authenticates and then authorizes an end-user to additional features. Non-functional elements are those that do not directly translate to features. For example, usability, reliability, supportability, availability, scalability, portability and performance are all non-functional elements of a software application. Unfortunately, many issues can be found within these difficult to remedy, non-functional elements, which are primarily responsible for determining the overall quality of the software application [8].

A software application's functional elements only determine a certain amount of its value; its non-functional elements are also a factor in determining value. For example, a particular software application performs a specific function, and this function has a certain value

to a specific organization, but the software application and its function are not easily scalable (i.e., the software application cannot feasibly be scaled up if the organization grows or scaled down if the organization shrinks). Therefore, the software application's overall value to the organization is limited by the organization's size. As the organization's size changes over time, the software application's value can diminish because it is no longer suitable for the new size of the organization. Due to the high impact of non-functional elements on a software application's value, it is important to identify and remedy those issues. Also, remedying issues within the non-functional category will increase the software application's overall quality and value.

Due to the size and complexity of legacy software applications, it is often impractical to exhaustively identify and document every issue, and attempting to do so is not the objective of this phase [7]. It is, however, important to collaboratively identify and document the major issues. To do this, use the *Issue Discovery Process Model* illustrated in Figure 2.

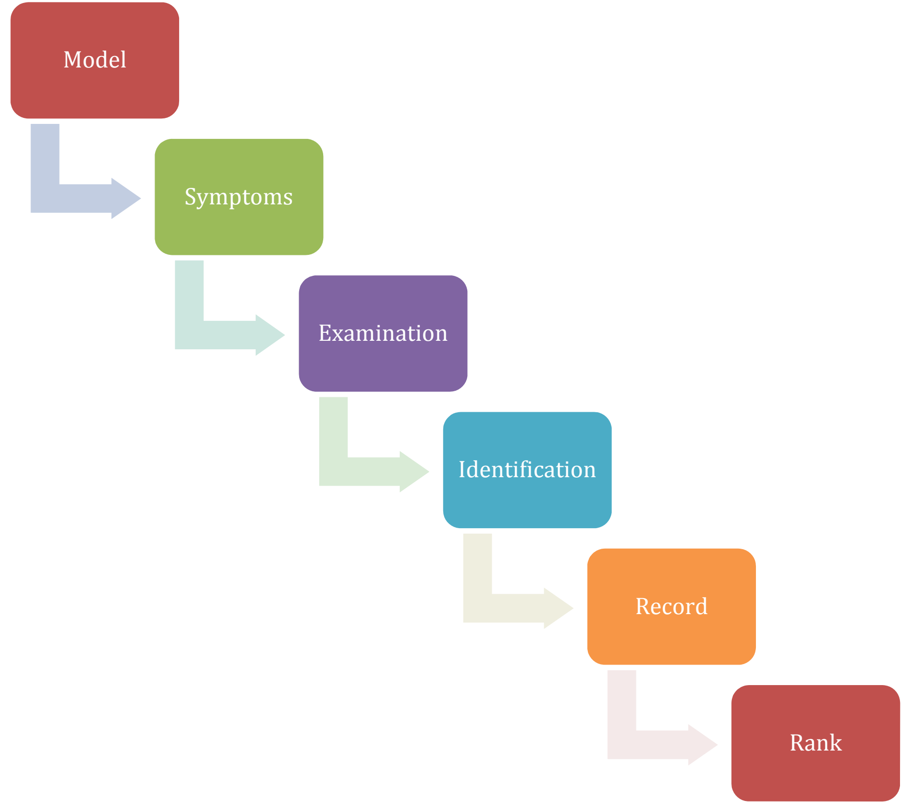

Figure 2: *Issue Discovery Process Model*

The steps in this model are:

1. **Model** – Construct architectural views of the critical architectural elements.
2. **Symptoms** – List symptoms gathered from stakeholders.
3. **Examination** – Investigate the architectural views and symptoms to find issues.
4. **Identification** – Make the determination that an element is problematic for a documentable reason.
5. **Record** – Write a short description of the issue, briefly explain why it is an issue, and list each symptom it is causing.
6. **Rank** – Review all recorded issues and assign a severity ranking to each.

The output of this process should be:

1. A set of models that describe critical architectural elements from certain viewpoints.
2. A *Symptoms* document listing each reported symptom, and stakeholder group that reported the symptom.
3. An *Issues* document listing and describing each identified issue.
4. A *Symptoms/Issues* cross reference document, linking symptoms to issues.

The goal is not to exhaustively identify and document all issues; this would be infeasible in most contexts. The goal is to understand the high-level issues with the existing architecture, technology, ongoing development process, and all relevant support processes. The above process should ideally identify no more than 25 - 50 high-level issues. If the list gets too long, the elements under examination may be too low-level. The issues list should be short enough that a single person could reasonable comprehend the list and attain an overall understanding of what the issues are without having to refer to long lists of low-level issues.

All of the documents output by this phase of the *Overhaul Concept Refinement Process Model* will be used as inputs to subsequent phases of the process, but the most important documents are the *Issues* document and the models that depict critical architectural views. These documents substantiate the necessity of overhauling the legacy software application under process.

2.2 Research Software Development Trends and Technology

The second phase in the *Overhaul Concept Refinement Process Model* is to research software development trends and the capabilities of newer, more current technology. This phase is critical because it empowers the construction of a new design concept for the software application being overhauled. Without a firm grasp on current software development trends and modern technology, the development of a competitive modern software application is essentially impossible. Additionally, research may further enable identification and documentation of existing issues (Phase 1 of the *Overhaul Concept Refinement Process Model*) by revealing issues that were previously unrealized.

While researching software development trends and modern technology, it is always important to avoid jumping to an early conclusion. For example, an appealing new software architecture may grab the attention of a developer, but it would be unadvisable for the developer to prematurely conclude the research process because they think they have just found the single solution that will resolve all or most of the existing issues. Be thorough. Understand the major driving forces behind the current software development trends, and understand the actual capabilities that modern technologies bring to the table. However, avoid spending an exorbitant amount of time on a single research item because this may result in missed opportunities to explore other research items. Researching modern technology can be an open-ended process, therefore, controlling the scope of the research and executing it in a timely manner is the only feasible option.

To define an adequate domain of research and to promote identification of research items, use the *Research Planning Process Model* (see Figure 3).

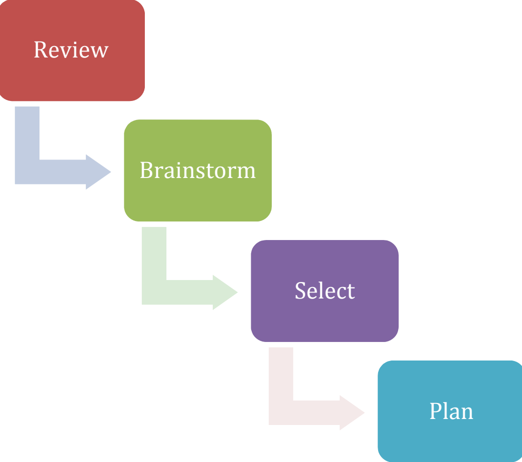

Figure 3: *Research Planning Process Model*

The steps in this model are:

1. **Review** – Analyze all documents created in Phase 1 of the *Overhaul Concept Refinement Process Model*.

2. **Brainstorm** – Consider the existing system's architectural constraints and documented issues, and then begin to brainstorm areas of research interest.
3. **Select** – Identify and document pertinent research items.
4. **Plan** – Allocate a specific amount of time to each selected research item and establish an overall deadline for completing the research. Additionally, determine how the research will be documented.

The output of this process should be a *Research Plan* document listing research items each designated with a time allocation, and an ultimate deadline for the overall research effort.

When executing the *Research Plan*, stick to the plan. However, while executing research, it is possible to stumble across relevant subject matter that is worthy of further investigation, but was not included in the *Research Plan*. If this happens, make note of the subject matter and then proceed with the research per the *Research Plan,* unless the subject matter is exceptional and worthy of altering the *Research Plan* immediately to accommodate. Situations worthy of altering the *Research Plan* should be handled carefully; the decision to alter the *Research Plan* should be made quickly via a collaborative effort by the appropriate project stakeholders, but the decision should not be made lightly.

2.3 Identify Remedies for Existing Issues

The third phase in the *Overhaul Concept Refinement Process Model* builds on the knowledge and documentation accumulated during the preceding phases. The objective of this phase is to identify and document potential remedies for each issue that was documented in Phase 1. Remedies are solutions capable of mitigating issues.

In order to identify potential remedies, it is necessary first to understand the existing issues, and research modern software development trends and technologies (Phases 1 and 2 of the *Overhaul Concept Refinement Process Model*). To identify potential remedies for the documented issues, use the *Remedy Discovery Process Model* (see Figure 4).

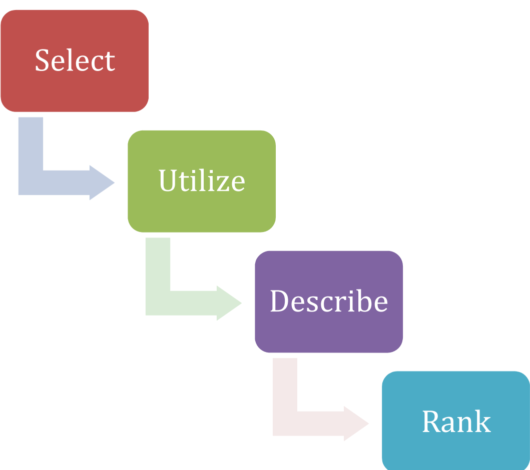

Figure 4: *Remedy Discovery Process Model*

The steps in this process are:

1. **Select** an issue from the *Issues* document.
2. **Utilize** knowledge gleaned from your research to list potential remedies for the selected issue.
3. Briefly **describe** how each potential remedy will eliminate or mitigate the issue.
4. **Rank** each remedy's level of difficultly, cost, and time to implement (high, medium, or low).

The output of this process is a document that associates ranked potential remedies with the documented issues.

2.4 Collectively Analyze and Conceptualize Solutions

The fourth phase in the *Overhaul Concept Refinement Process Model* is to collectively analyze all documents created in the preceding steps and begin to formulate overall architectural design concepts and development process concepts. It is not the goal of this phase to form completely architected solutions, but rather it is about conceptualizing potential high-level architectures and development processes, given a well formed understanding of the existing issues, modern development trends, and potential remedies. Primarily, the goal of this phase is to

conceptualize what the existing software could become and how it could be achieved. To produce these outputs, use the *Conception Process Model* (see Figure 5).

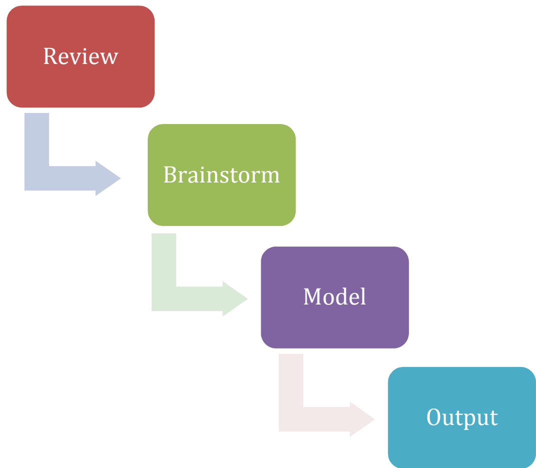

Figure 5: *Conception Process Model*

The steps in this process are:

1. **Review** all documentation from previous phases.
2. **Brainstorm** design concepts with team members.
3. Collaboratively draft high-level graphical **models** of potential software architectures and/or software development processes.
4. **Output** all initial drafts to next overall phase in the *Overhaul Concept Refinement Process Model*.

The potential outputs of this process are:

1. High-level graphical models of potential Software Architectures.
2. High-level graphical models of potential Software Development Processes.

2.5 Select a Solution to Refine

The fifth phase in the *Overhaul Concept Refinement Process Model* is to select a solution to refine. The preceding phase of the *Overhaul Concept Refinement Process Model* may have resulted in:

1. One Architectural Concept and One Development Process Concept
2. One Architectural Concept and Multiple Development Process Concepts
3. Multiple Architectural Concepts and One Development Process Concept

4. Multiple Architectural Concepts and Multiple Development Concepts

At this stage, selection of a single architectural concept coupled with a single development process concept, may be impractical. It may be necessary to select multiple pairings of architectural and development process concepts for further refinement via rapid prototyping, in order to discover the best single pairing of architectural and development process concepts.

Keep in mind, the primary goal of the *Overhaul Concept Refinement Process Model* is to arrive at a well formed architectural and development process concept that is mature enough for successful implementation. Many software projects are canceled because the initial concept was poor, but the realization of its poorness was not discovered until well into implementation. To select a solution(s) to refine, use the *Solution Selection Process Model* (see Figure 6).

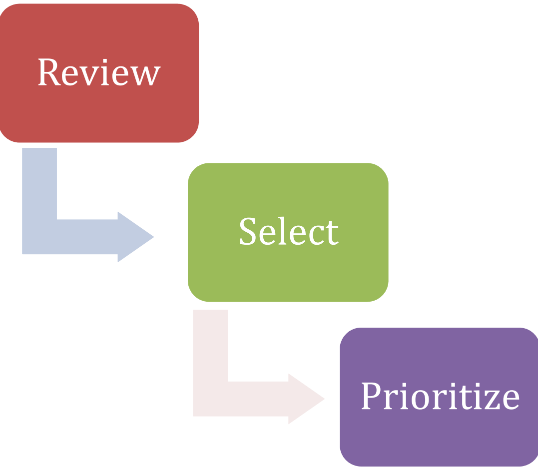

Figure 6: *Solution Selection Process Model*

The steps in the process are:

1. Collaboratively **review** each draft and determine if it is worthy of further investigation.
2. **Select** and list the drafted models that are worthy of further investigation.
3. **Prioritize** the list.

The output of this phase is a prioritized list of architectural concepts paired with a development process.

2.6 Refine Solution with Rapid Prototyping

The sixth phase in the *Overhaul Concept Refinement Process Model* is to refine potential solutions via rapid prototyping. Multiple solutions may be selected for refinement via rapid prototyping, if time and budget allows. A prototype is a preliminary model of a software application that implements a subset of the end product's features, and enables customers and developers to examine critical aspects of the proposed software application [5]. Furthermore, prototyping helps validate whether or not a proposed software application can be successfully developed, given the selected architectural concept and development process concept.

Prototyping should not be restricted to the software product, but should include the development process itself. The process of developing software is critical and must be considered with great care. Therefore, prototyping a development process tailored specifically to the new architectural design concept is beneficial. Ideally, the prototype process should be implemented during the development of the prototype software product. The quality of the process used to develop a software product impacts the quality of the product.

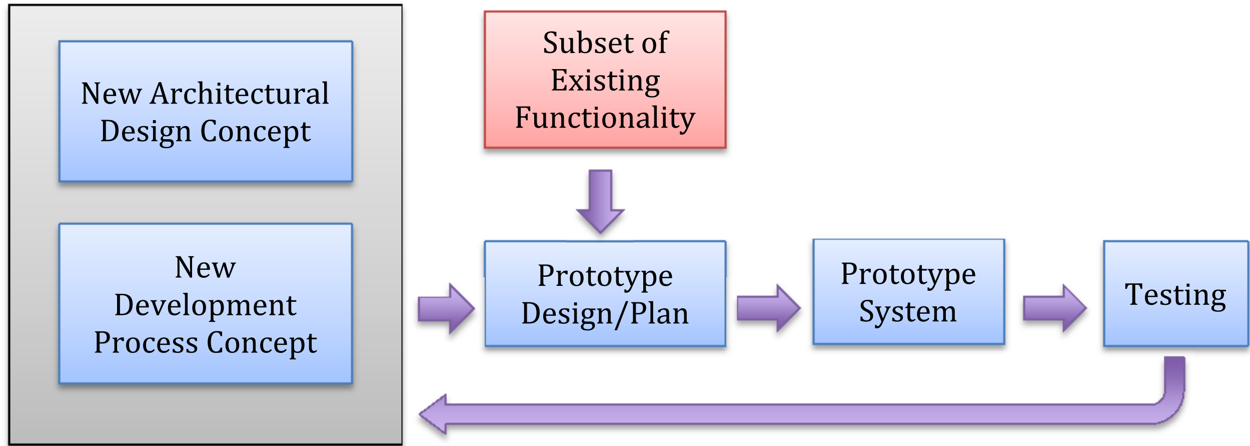

Figure 7: *Rapid Prototyping Process Model*

The *Rapid Prototyping Process Model* (see Figure 7) illustrates the usage of the new architectural design concept, new development process concept, and a subset of the existing software application's functionality to develop prototypes of the software product and its development process. Furthermore, the model illustrates a feedback loop to enable iterative refinement of the prototypes. Prototyping is complete when the prototypes (i.e., development process prototype and software product prototype) adequately substantiate (or fail to substantiate) the new architectural design concept and development process concept. If the prototypes do not substantiate the new design concepts, creation of new design concepts may be necessary.

During prototyping, do not make compromises in order to get the prototype working more quickly. For example, do not use an inappropriate programming language, operating system, or inefficient algorithm because of familiarity or simplicity of demonstration. Over time these types of compromises become inappropriately familiar to developers, which may lead to them becoming an integral part of the system's new design. It is imperative to construct prototypes without compromising the original design intent [5].

2.7 Final Preparation for Software Development

The seventh phase in the *Overhaul Concept Refinement Process Model* is to evaluate the final prototyped development process and software product, and determine if the project can be feasibly pursued. To evaluate the prototypes, use the *Prototype Evaluation Process Model* (see Figure 8).

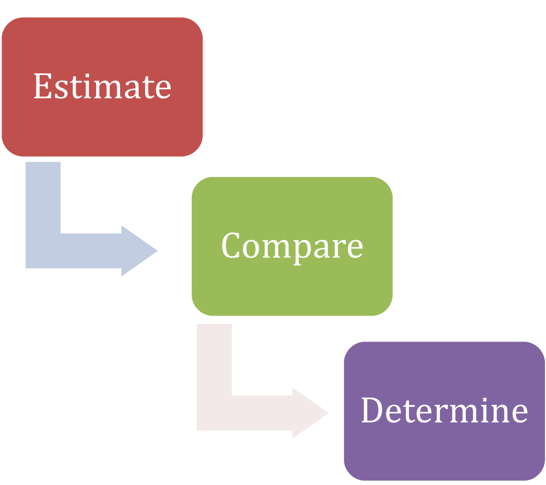

Figure 8: *Prototype Evaluation Process Model*

The steps in this process are:

1. **Estimate** the resources and cost necessary to implement the full-scale software overhaul effort.
2. **Compare** the estimated resources and cost to the available resources and budget.
3. **Determine** if the project can continue (or in what limited capacity it could continue).

If it is determined that the project will continue, begin the process of gathering the necessary resources and organizing the project. Answer these questions using the prototyped development process as a basis [9, 10]:

1. How will the overhaul project team be structured?
2. What tools will be used to enable project management?
3. How will the overall project plan be documented, and how will it be used to measure the project's overall progress and ensure the project remains within budget?

CHAPTER 3: CASE STUDY

Service Central® is an enterprise-level post-sales service and support software application developed via loosely implemented Extreme Programming methodologies. It is a client-server software application that is primarily built upon the following Web technologies, programming languages, and markup technologies:

- Microsoft's classic ASP (Active Server Page) Web technology implemented in VBScript for server-side computation
- JavaScript for client-side computation
- HTML
- XML

Additionally, Service Central uses Microsoft's SQL Server for its "system database" and IBM's DB2 (typically on an IBM AS/400 midrange) for its "application database".

Service Central is a highly functional (but poorly documented) software application that has evolved via continuous development efforts over the past decade. It consists of millions of lines of code spanning multiple programming languages, and hundreds of complex database tables (and views) that are spread across two types of database management systems, each of which implement a unique version of SQL.

Extreme Programming techniques have facilitated a certain degree of Service Central's success, in terms of helping to flexibly meeting customer requirements. However, loosely defined Extreme Programming methodologies have not been conducive to producing adequate system documentation; nor has it brought a great degree of scalability, maintainability, portability, or performance to Service Central.

Unfortunately, as evident by Service Central, Extreme Programming can easily become too extreme, if it is not appropriately defined and implemented from the beginning of the

software lifecyle model. That is, documentation (including source code comments) becomes nearly non-existent, and Unit Testing and Acceptance Testing become the only levels of software testing. This exposes too many Integration level and System level defects to customers and end-users during acceptance testing and production usage of the system. However, Extreme Programming can be successful if it is strictly defined and implemented to the established definition.

The subsequent sections in this chapter will examine the details of each phase in the *Overhaul Concept Refinement Process Model* as they have been applied to Service Central by its overhaul concept refinement team. Each section in this chapter will provide examples of the artifacts that are output from each phase of the *Overhaul Concept Refinement Process Model*. Additionally, code snippets and screenshots from the prototype produced by the execution of the overhaul process on Service Central are included in the appendices. Appendix G defines a new type of architectural style that was created during the execution of the overhaul process on Service Central. The next seven sections (3.1 – 3.7) will each examine a phase of the *Overhaul Concept Refinement Process Model* as it was applied to Service Central.

3.1 Understanding the Existing Issues

The first phase of the *Overhaul Concept Refinement Process Model* is to understand the existing issues. Service Central's overhaul concept refinement team executed the *Issue Discovery Process Model* against Service Central and produced the following artifacts:

1. A model that illustrates the critical architectural elements of the existing state of Service Central.
2. A *Symptoms* document listing each reported symptom, and stakeholder group that reported the symptom.
3. An *Issues* document listing and describing each identified issue.
4. A *Symptoms/Issues* cross reference document, linking symptoms to issues.

The first step of the *Issue Discovery Process Model* requires the creation of an architectural model that illustrates the important features of Service Central's existing architecture. In accordance with this requirement, Service Central's overhaul concept refinement team analyzed the existing architectural components and created an architectural model that illustrated Service Central's architectural state from a bird's eye view (see Figure 9).

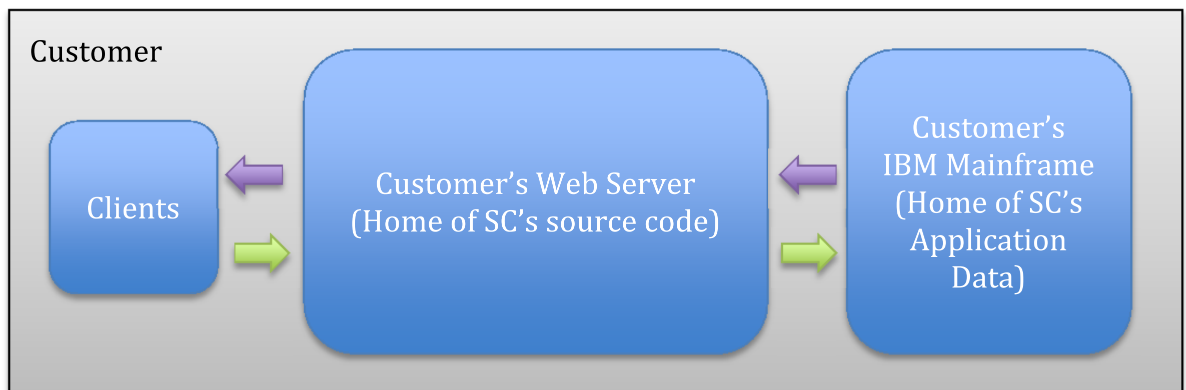

Figure 9: *Legacy Architecture*

Service Central's *Legacy Architecture* is cumbersome to deploy to new customers because every customer requires their own dedicated Web server and Application Database. Additionally, the legacy architecture results in Service Central's source code being deployed to many different locations and platforms. This results in various customers being at different release levels of the software and introduces many other undesirable situations. For example, as Service Central's customer base grows, delivery of upgrades becomes more difficult because the upgrades must be distributed to more systems on a wider variety of platforms.

After creating the architectural model of the *Legacy Architecture*, Service Central's overhaul concept refinement team proceeded to the second step of the *Issue Discover Process Model*. The second step resulted in the creation of a *Symptoms Document* (see Figure 10). The *Symptoms Document* was very simple to construct and only listed five major symptoms as gathered from

various project stakeholders. During the process of gathering symptoms, it was discovered that many stakeholders would actually report issues (instead of symptoms) from which the symptoms could be derived/abstracted by the overhaul concept refinement team. For example, end-users would report that Service Central's graphical user interface was unattractive and that its response time was slow. These reported issues had to be translated to their actual symptoms. For example, "unattractive" translated to "difficult to sale" and "difficult to use". Also, Service Central's overhaul concept refinement team determined to only include the major symptoms. In fact, the symptoms that were listed in the *Symptoms Document* were umbrella symptoms, capable of suitably covering the lower level symptoms that were reported by stakeholders.

Service Central - Symptoms Document

	Symptom	Stakeholder Group
1	Difficult to modify source code.	Developers
2	Difficult to install and upgrade.	Developers
3	Slow to load screen and data.	End-Users
4	Difficult to use.	End-Users
5	Difficult to sale.	Marketing

Figure 10: *Symptoms Document*

Once the *Symptoms Document* was complete, Service Central's overhaul concept refinement team proceeded with steps 3-6 of the *Issue Discover Process Model*, by first investigating the *Legacy Architecture* and the symptoms listed in the *Symptoms Document*, in an

attempt to discover issues within Service Central. When issues were discovered, they were recorded in an *Issues Document*. Finally, once all issues were listed, they were analyzed by Service Central's overhaul concept refinement team, and each was assigned a severity ranking that was also recorded in the *Issues Document*.

As with the *Symptoms Document*, the *Issues Document* only captured the major issues (see Figures 10 & 11). The goal was not to exhaustively identify and rank all existing issues, but to identify and rank the high-level issues.

Service Central - Issues Document

	Issue	Severity Ranking
1	Spaghetti code throughout code modules	High
2	Distributed source code across customers	Medium
3	Unnecessary data being loaded	Medium
4	Poor documentation	Medium
5	Unattractive user interface	High
6	Poor development process	High
7	Old coding technology	Medium
8	Not user configurable	Low
9	Lacking open APIs	Medium

Figure 11: *Issues Document*

Finally, Service Central's overhaul concept refinement team constructed a *Cross Reference Document* (see Figure 12). In order to create this document, Service Central's overhaul concept refinement team had to analyze the *Symptoms Document* and the *Issues Document* and determine the issues that could be categorized under each symptom. The *Cross Reference Document* captured each symptom from the *Symptoms Document* and linked it with all of its underlying issues from the *Issues Document*.

Service Central – Cross Reference Document

	Symptom No.	Issue No.
1	Symptom 1	Issue1
2	Symptom 1	Issue 4
3	Symptom 1	Issue 6
4	Symptom 1	Issue 7
5	Symptom 2	Issue 2
6	Symptom 2	Issue 4
7	Symptom 2	Issue 7
8	Symptom 2	Issue 8
9	Symptom 3	Issue 1
10	Symptom 3	Issue 3
11	Symptom 3	Issue 7
12	Symptom 4	Issue 4
13	Symptom 4	Issue 5
14	Symptom 4	Issue 8
15	Symptom 4	Issue 9
16	Symptom 5	Issue 4
17	Symptom 5	Issue 5

Figure 12: *Cross Reference Document*

3.2 Researching Software Development Trends and Technology

Upon completion of the first phase of the *Overhaul Concept Refinement Process Model*, Service Central's overhaul concept refinement team proceeded to the second phase. The objective of the second phase was to create and execute a *Research Plan Document* (see Figure 13). In order to accomplish these objectives, Service Central's overhaul concept refinement team, executed the *Research Planning Process Model*.

First, Service Central's overhaul concept refinement team reviewed the four documents created during the previous phase and brainstormed over these documents to identify pertinent research items. Upon identification of a research item, it was added to the *Research Plan Document*. After all of the pertinent research items were listed, each was considered distinctly and a specific amount of research time was allocated. Finally, the time allocations of each research item were summed and the total was recorded in the *Research Plan Document*.

Once the *Research Plan Document* was completed, Service Central's overhaul concept refinement team executed research per its specifications. During the research phase, Service Central's overhaul concept refinement team not only gleaned knowledge on the research items specified in the plan, but also acquired an extended understanding of the documented symptoms and issues. This additional understanding of the symptoms and issues would later prove beneficial when attempting to conceptualize new potential architectures for Service Central.

Service Central – Research Plan Document

	Research Item	Time Allocation (hours)
1	Cloud Computing	16
2	HTML5	10
3	Service Oriented Architectures (SOA)	20
4	JavaScript Object Notation (JSON)	10
5	Server-Side Programming Languages	20
6	Agile Software Development Models	40
7	Mobile Web Development	20
8	User Interface Design	10
9	Databases	16

Total = 162 Man-hours of research

Figure 13: *Research Plan Document*

3.3 Identifying Remedies for Existing Issues

Upon completion of the second phase of the *Overhaul Concept Refinement Process Model*, Service Central's overhaul concept refinement team proceeded to the third phase. The objective of the third phase was to create a *Remedies Document* (see Figure 14). In order to accomplish this objective, Service Central's overhaul concept refinement team, executed the *Remedy Discovery Process Model*.

Service Central's overhaul concept refinement team implemented the *Remedy Discovery Process Model* by first selecting issues, one at a time, from the *Issues Document* and utilizing knowledge gleaned from the second phase to identify potential remedies for each issue. Each remedy identified was listed in the *Remedies Document*. Once all remedies were identified, they were ranked based on importance.

Service Central - Remedies Document

	Remedies	Issue No.	Ranking
1	Implement Service Orient Architecture	Issue 1	High
2	Migrate Service Central to the Cloud	Issue 2	Medium
3	Use JSON to pass only required data	Issue 3	Medium
4	Establish an Agile documentation process	Issue 4	Medium
5	Use HTML5 and the JQuery library	Issue 5	High
6	Implement Scrum or Extreme Programming	Issue 6	High
7	Incrementally migrate services coded in C#	Issue 7	Medium
8	Decouple front-end from back-end via SOA	Issue 8	Low
9	Use SOA and create a RESTful interface	Issue 9	Medium

Figure 14: *Remedies Document*

3.4 Collectively Analyzing and Conceptualizing Solutions

Upon completion of the third phase of the *Overhaul Concept Refinement Process Model*, Service Central's overhaul concept refinement team proceeded to the fourth phase. The objectives of the fourth phase were to collectively analyze all of the documents created during preceding phases, begin to formulate overall architectural design concepts and development process concepts, and to output high-level graphical models of these concepts.

In order to accomplish these objectives, Service Central's overhaul concept refinement team, executed the *Conception Process Model*. During the execution of that process, Service Central's overhaul concept refinement team first collaboratively drafted a *New High-Level Architectural Concept* in the form of a graphical model (see Figure 15). Once the *New High-Level Architectural Concept* was created, a *New High-Level Development Process Concept* was derived (see Figure 16) and based on the *New High-Level Architectural Concept*.

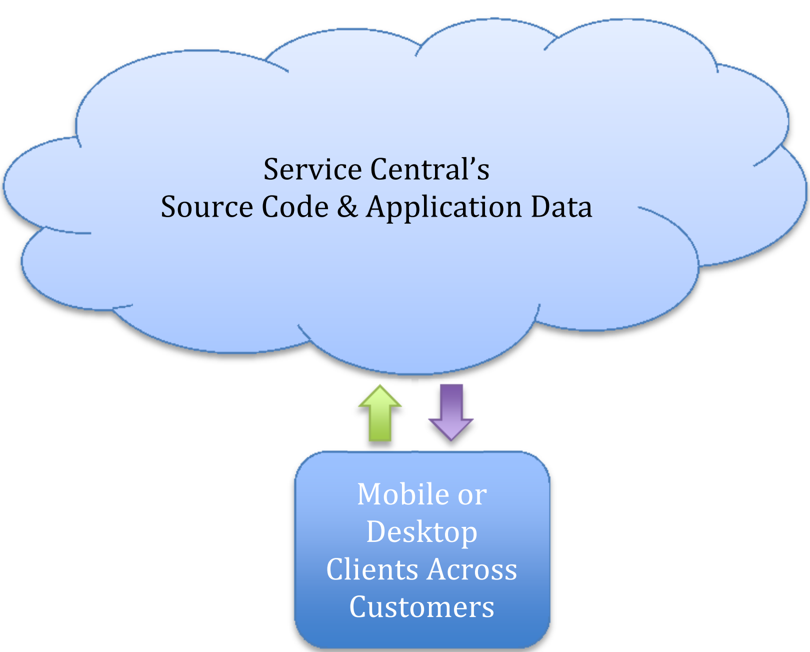

Figure 15: *New High-Level Architectural Concept*

The *New High-Level Architectural Concept* was a cloud-based service oriented architecture [11, 12]. It was established because it is a popular style being successfully implemented by comparable modern software products, and it best facilitated the remedies listed in the *Remedies Document*. For example, cloud-based service orient architectures are well suited for being incrementally developed and offer the potential to support massive scalability [13, 14]. Furthermore, they resolve the issue of having the source code distributed across many customers; this greatly reduces the difficulty of upgrading all Service Central customers simultaneously.

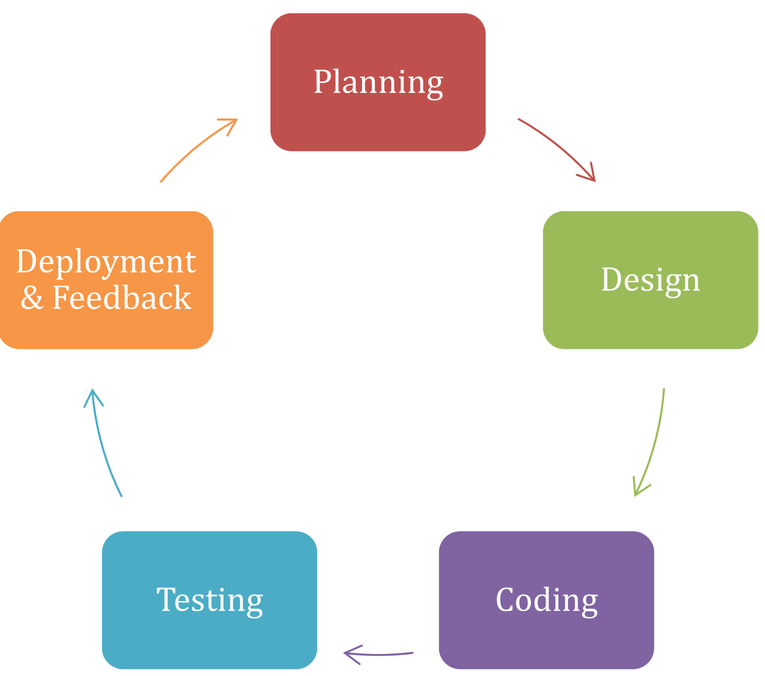

Figure 16: *New High-Level Development Process Concept*

The *New High-Level Development Process Concept* was established based on Extreme Programming methodologies. Service Central's overhaul concept refinement team drafted this development process model because Extreme Programming methodologies align well with a product that must be built incrementally over time [15, 16].

3.5 Selecting a Solution to Refine

Upon completion of the fourth phase of the *Overhaul Concept Refinement Process Model*, Service Central's overhaul concept refinement team proceeded to the fifth phase. The objective of the fifth phase was to create a *Prioritized Solutions Document* that listed the potential architectural concepts associated with their development process concept (see Figure 17). Moreover, this prioritized list identifies the solution to select for further refinement via rapid prototyping by establishing a solution as being highest in priority [5].

Service Central's overhaul concept refinement team had only created a single *New High-Level Architectural Concept* and *New High-Level Development Process Concept* during the preceding phase. However, upon creation of the *Prioritized Solutions Document*, it was determined that the single architectural concept could be teased apart into similar concepts, one being explicitly for mobile clients and the other for desktop clients [17]. The determination to identify two separate architectural concepts should technically have been accomplished in the preceding phase, but happened during the fifth phase simply by chance. The idea was conceived during the fifth phase, and Service Central's overhaul concept refinement team collaboratively determined to tease apart the single architectural concept into two concepts, documenting them as separate solutions in the *Prioritized Solutions Document*.

Service Central - Solutions Document

	Architectural Concept	Development Process
1	Mobile Web App via SOA	Extreme Programming
2	Desktop App via SOA	Extreme Programming

Figure 17: *Prioritized Solutions Document*

3.6 Refining Solution with Rapid Prototyping

During the sixth phase of the *Overhaul Concept Refinement Process Model*, a prototype development process and a prototype mobile Web application were created for the highest priority solution identified in the *Prioritized Solutions Document*. First, the development process was defined as follows [15]:

1. The Software product will be created iteratively; each iteration will produce deliverable software. (*No software will be delivered to the customer during development of prototype product. An internal review between developers will occur at the end of each iteration.*)
2. Each iteration will consist of five phases:
    a. Planning
        i. Express requirements as short unambiguous user stories, and derive the user stories from use cases that exist within the current product. Additionally, each use case will be examined for efficiency before being translated into user stories.
        ii. Assess user stories.
        iii. Group user stories.
        iv. Use project velocity to commit to iteration specific delivery date.
    b. Design
        i. K.I.S.S. principle (*Keep It Simple Stupid*).
        ii. Use *Class Collaborator Cards* and graphical models to express user stories in design terms.
        iii. Create unit tests.
    c. Coding
        i. Code to the unit tests.
        ii. Execute unit tests regularly.

iii. Comment source code sufficiently to enable all developers to easily understand the code.
iv. Create documentation of configurable code modules and complex integration interfaces.

d. Testing
   i. Execute integration level tests.
   ii. Execute system level tests.
   iii. Have end-users perform acceptance tests (*No acceptance test during development of prototype product*).

e. Deployment & Feedback (*Deployment will not occur during development of the prototype product.*)
   i. Deliver new functionality to the customer's production environment.
   ii. Measure the project's velocity.
   iii. Document the lessons learned during the iteration.
   iv. Document all functionality added during the iteration.

Once the prototype development process was defined, it was implemented on Service Central in order to develop a prototype mobile Web application. Figures 18 and 19 are examples of user stories that were created during the planning phase of the first development iteration.

Login

End-users must be presented with a login screen that requires the entry of predefined security credentials. Upon subsequent arrivals at the login screen (via re-launching the application), the End-User's previously authenticated credentials should automatically populate the input fields, requiring the end-user to only press the "submit" button to login.

Figure 18: *Sample User Stories - Login*

Landing (Main Menu)

Upon authentication of the End-user login credentials, the end-user should arrive at a common menu that allows quick selection of the desired Service Central document type or service. There should be menu items for: Tickets, Work Orders, RMAs, Quotes, Product Registrations, and User's Messages.

Figure 19: *Sample User Stories - Landing*

During the design phase of the first development iteration, an architectural style was defined (see Appendix G). In adherence to the newly defined architectural style, the following technologies were selected for the various layers of the style:

1. **Client Layer**
   i. HTML5 & JQuery Mobile (*for creation of the graphical user interface*).
2. **Façade Layer**
   i. Helicon (software that *bolts onto Microsoft's IIS Server 2003 to intercept HTTP requests and rewrite the request to the correct URI controller*).
   ii. VBScript URI controller (*it examines the HTTP requests directed to it by Helicon and determines the correct service to invoke in order to generate the appropriate HTTP response*).
3. **Service Layer**
   i. VBScript Services (*they instantiate the necessary objects that are used for performing CRUD against database objects*).
4. **Class Layer**
   i. VBScript Public Classes (*for interaction with particular database objects*).
5. **Database Layer**
   i. Microsoft's SQL Server 2008 (*for storing system and application level data*).

Furthermore, CRC cards were created for some initial classes (see Figure 20).

Service	
Security Credentials URI JSON Version Error Message Make HTTP GET Request Make HTTP POST Request Make HTTP PUT Request Make HTTP DELETE Request	Security

Figure 20: *Sample CRC Card - Service*

Next, during the coding phase of the first development iteration, Helicon® was installed and configured to intercept HTTP requests and appropriately redirect those requests to a URI controller. Here is how Helicon's *httpd.config* file was implemented in order to rewrite HTTP requests to the URI controller:

```
# Helicon ISAPI_Rewrite configuration file
# Version 3.1.0.95

RewriteEngine On

# Don't apply files/directories that actually exist at the host location
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

# Prefix with FURI {ServiceDomain} "/sc" (see both instances of sc below)
RewriteRule ^/sc/([a-z0-9./]+)$ /sc/URIController.asp [L,NC]
```

Figure 21: *Helicon Configuration File*

Additionally, during the coding phase, an initial URI controller was constructed (see Appendix C for the implemented source code of the URI controller). The initial URI controller directly embedded the *service* and *class* layers of the architectural style. The goal of the first

development iteration was to validate the concept of the URI controller. In alignment with that goal, a client-side JavaScript object constructor that is responsible for instantiating specific *services* (that exist on the server-side) and establishing a common means of interfacing with *services* from within the client-side code, was created (see Appendix B for the implemented source code of the client-side JavaScript object constructor) [18, 19, 20, 21].

However, before any code was written, unit tests were always defined (see Figures 22 and 23).

Unit Test: Parse URI's *Collections* & *Elements*

Parse URI into an array type VBScript variable that sequentially stores the URI's collections and elements. Unit test of parse URI http://testdomain.com/sc/regs/1/notes/2 should result in the URI's collections and elements being storied in an array where the array's index of:

0 = "regs"
1 = "1"
2 = "notes"
3 = "2"

Figure 22: *Sample Unit Test – Parse URI's Collections & Elements*

Unit Test: Parse URI's Query String

Parse URI's query string into an array type VBScript variable that sequentially stores the URI's query string's key/value pairs. Unit test of parse URI's query string http://testdomain.com/sc/regs/?SysId=S1&UserId=Smith should result in the URI's query string elements being storied in an array where the array's index of:

0 = "SysId=S1"
1 = "UserId=Smith"

Figure 23: *Sample Unit Test – Parse URI's Query String*

These types of unit tests were continuously created, coded to, and tested against during the iteration's coding phase.

During the testing phase of the first development iteration, a set of integration and system level tests were executed that validated the system implemented functioned as anticipated. The tests were executed as follows [7, 22]:

Integration Level Tests

Figure 24 and Figure 25 exemplify the integration level tests executed.

Test 1

Procedure: Browse to http://5.221.208.53/sc/regs?Sysid=S1 in Internet Explorer.

Expected Result: An Internet Explorer window containing a JSON structure that contains an array of all the Registration Requests from the REGSREQS database file that is located in the SQL Server database named ServCentral.

Actual Result: The actual result matched the description of the expected result. (See Appendix D for details of GET request to URI http://5.221.208.53/sc/regs?Sysid=S1).

Figure 24: *Integration Test 1*

Test 2

Procedure: Browse to http://5.221.208.53/sc/regs/10?Sysid=S1 in Internet Explorer.

Expected Result: An Internet Explorer window containing a JSON structure that contains the Registration Request # 10 from the REGSREQS database file that is located in the SQL Server database named ServCentral.

Actual Result: The actual result matched the description of the expected result. (See Appendix E for details of GET request to URI http://5.221.208.53/sc/regs/10?Sysid=S1).

Figure 25: *Integration Test 2*

System Level Tests

Figure 26 and Figure 28 exemplify the system level tests executed. Figure 27 and Figure 29 show the HTML source code of the URLs browsed to in the system level tests.

Test 1

Procedure: Browse to http://testdomain.com/AllRegs.html in Internet Explorer.

Expected Result: An Internet Explorer window titled "All Registrations" containing a JSON structure that contains an array of all the Registration Request from the REGSREQS database file that is located in the SQL Server database named ServCentral.

Actual Result: The actual result matched the description of the expected result. (See Appendix D for HTTP response body of GET requests to URI http://5.221.208.53/sc/regs?Sysid=S1).

Figure 26: *System Test 1*

```
<html>
  <head>
    <title>All Registrations</title>
    <script src="service.js"></script>
    <script>
      //Create instant of Service object
      var objAllRegs  = new Service("http://5.221.208.53/sc/regs?Sysid=S1");

      //Make HTTP GET request
      objAllRegs.get();

      //Print to screen the returned JSON string
      document.write(objAllRegs.JSONString);
    </script>
  </head>
    <body> </body>
</html>
```

Figure 27: *Code Snippet – AllRegs.html*

Test 2

Procedure: Browse to http://testdomain.com/Reg10.html in Internet Explorer

Expected Result: An Internet Explorer window titled "Registration # 10" containing a JSON structure that contains the Registration Request # 10 from the REGSREQS database file that is located in the SQL Server database named ServCentral.

Actual Result: The actual result matched the description of the expected result. (See Appendix E for HTTP response body of GET requests to URI http://5.221.208.53/sc/regs/10?Sysid=S1).

Figure 28: *System Test 2*

```
<html>
  <head>
     <title> Registration # 10</title>
     <script src="service.js"></script>
     <script>
        //Create instant of Service object
        var objAllRegs  = new Service("http://5.221.208.53/sc/regs/10?Sysid=S1");

         //Make HTTP GET request
         objAllRegs.get();

         //Print to screen the returned JSON string
          document.write(objAllRegs.JSONString);
     </script>
  </head>
     <body> </body>
</html>
```

Figure 29: *Code Snippet – Reg10.html*

Finally, during the feedback phase of the first development iteration, it was noted that the source code needed to contain more descriptive commenting and that all user stories, CRC cards, and unit test should be captured on a digital medium instead of physical paper cards. Also, it was documented that the URI controller would need to be refactored and extended to include a hierarchical data structure that captures all of Service Central's *collections*, and that a mechanism

would need to be created to traverse the hierarchical structure of *collections* in order to invoke the appropriate external *service* (see Appendix H for a diagram of the initial *collections* hierarchy) [18, 21, 22].

The results of the first development iteration were extremely positive and indicative of a well selected architectural concept and development process. Subsequent development iterations further validated the architectural concept and development process, and ultimately sufficient prototypes for both were completed (see Appendix F for screenshots of the final prototype).

3.7 Final Preparation for Software Development

During the seventh phase of the *Overhaul Concept Refinement Process Model*, Service Central's overhaul concept refinement team evaluated the prototyped development process and prototyped software product, and determined that the overhaul project would continue. However, due to a lack of necessary resources (i.e., developers and tools), it was determined that the continuation of the overhaul project would require a three month period to acquire the appropriate resources for the project, and to create a fully defined high-level project plan.

3.8 Projected Benefits

Service Central's overhaul concept refinement team has projected significant benefits to customer-side stakeholders and development-side stakeholders (see Tables 1 and 2). These projected benefits are the direct results of the *Overhaul Concept Refinement Process Model*.

	Benefit
1	Service technicians will be able to collect digital signatures in the field.
2	Service technicians will be able to record labor hours in the field.
3	Maintenance costs will be reduced due to higher quality software.
4	Response time to incidents will be reduced.
5	Upgrades will happen automatically.
6	Initial installation process will be eliminated because the software will reside in the cloud.

Table 1: *Service Central's Projected Customer-Side Stakeholder Benefits*

	Benefit
1	Maintenance costs will be reduced due to the higher degree of maintainability of the source code and its overall quality.
2	Deployment effort will be greatly reduced because the software will reside in the cloud.
3	Sales will increase because of new functionality, and higher quality.
4	Strategic development will be increased due to freeing up the development resources previously used to deploy, upgrade, and maintain the legacy system.
5	Its new SOA will simplify the enhancement process by reducing the degree of coupling between components.
6	Service Central will remain competitive with newer software applications and continue to support existing customers more successfully.

Table 2: *Service Central's Projected Development-Side Stakeholder Benefits*

CHAPTER 4: OVERHAUL CONCEPT REFINEMENT PROCESS MODEL ANALYSIS

Overhauling a legacy software application can potentially enhance two aspects of the legacy software application: the functional and non-functional aspects (refer to Table 3).

	Aspect	Description
1	Functional	The features made available to its end-users.
2	Non-Functional (Quality)	The quality of its features and non-function aspects. For example, quality characteristic include: usability, reliability, supportability, availability, scalability, portability, performance, maintainability and etc.

Table 3: *Aspects of Legacy Software Applications*

The aim of an overhaul is to enhance these two aspects of the legacy software application by adding new functionality and/or removing obsolete functionality, and by improving its quality characteristics. In doing so, the customer-side stakeholders and the development-side stakeholders will benefit in substantial ways (refer to Tables 4 & 5). However, overhauling a legacy enterprise software application can be risky and difficult (refer to Table 6). In order to successfully overhaul a legacy software application, the risks must be mitigated. The *Overhaul Concept Refinement Process Model* aims to mitigate risks involved, thereby enabling the realization of the benefits by the stakeholders (refer to Table 7).

	Benefit	Description
1	Increased Functionality	Additional features end-users can utilize to perform more functions.
2	Increased Quality	The functions can be executed more quickly and with increased accuracy. The software application is more learnable.
3	Lower Maintenance Cost	Maintenance costs are reduced due to higher quality software.
4	Quicker Response Time to Incidents	Due to an increased degree of maintainability, customers can more quickly receive responses to incidents reported to software support.
5	Easier to Upgrade	Manual upgrade processes can be replaced by automated upgrades, making upgrades easier, more consistent, and less expensive.
6	Easier to Implement	Initial setup and installation processes can be simplified, resulting in shorter and less expensive initial implementations.

Table 4: *Customer-Side Stakeholder Benefits*

	Benefit	Description
1	Lower Maintenance Cost	Maintenance costs are reduced due to the higher degree of maintainability of the source code and its overall quality.
2	Easier to Deploy	Manual deployment to new customers may be automated, stimulating growth.
3	Easier to Sell	More functionality, higher quality, and modern technology can increase sales.
4	Enables More Strategic Development	Less time spent maintaining, deploying, and upgrading can free up time to continuously improve and optimize the software application via continuous strategic development.
5	Easier to Enhance	New component architecture can simplify the enhancement process.
6	More Competitive	The product can remain competitive with new software applications and continue to support existing customers.

Table 5: *Development-Side Stakeholder Benefits*

	Risk	Description
1	Incomplete Understanding of Existing Issue	The development team may have an incomplete or inaccurate understanding of the legacy system’s issues, resulting in poor decisions regarding the overhaul process.
2	Insufficient Knowledge of Technology	The development team may lack the knowledge necessary to successfully overhaul the legacy system, and select or design an appropriate development process.
3	ἀnappropriate Architecture	A poor architecture may have a severe impact on the legacy system’s functionality and quality characteristics, and negatively impact the overall success of the overhaul project.
4	Ineffective Development Process	An ineffective development process may have a severe impact on the legacy system’s functionality and quality characteristics, and negatively impact the overall success of the overhaul project.
5	Incorrect Toolset	The wrong tools may cause the overhaul project to be canceled or exceed budget and schedule constraints.
6	Incomplete Functionality	The legacy system may lose functionality during the process of being overhauled.
7	Premature Selection	The premature selection of a programming language, system architecture, development process, or toolset may have a severe impact on the overhaul project’s overall success.
8	Inaccurate Timeline	An inaccurate project timeline could ruin the project’s and legacy system’s success.
9	Reduced Quality	The legacy system’s quality may be reduced by a poorly designed and implemented architecture or development process.

Table 6: *Risks in Overhauling Legacy Software Applications*

	Risks Mitigated (from Table 6)	Description
Phase 1	1, 6	Involves thoroughly comprehending the legacy software application's existing issues as a team [31, 32].
Phase 2	2, 5	Involves researching software development trends and the capabilities of newer, more current technology [32].
Phase 3	1, 2, 3, 4, 7	Involves the identification and documentation of potential remedies for each issue that was documented in Phase 1 [31].
Phase 4	3, 4, 6, 7	Involves collectively analyzing all documents created in the preceding steps and beginning to formulate overall architectural design concepts and development process concepts [33, 34].
Phase 5	3, 4, 7	Involves selecting the best potential solution to refine. [32, 34]
Phase 6	2, 3, 4, 5, 6, 7, 8, 9	Involves refining potential solutions via rapid prototyping of the development process and architectural concepts [5, 32, 34].
Phase 7	5, 6, 7, 8, 9	Involves evaluating the final prototyped development process and software product to determine if the project can be feasibly pursued [5, 26, 32, 34].

Table 7: *Risks Mitigated / Overhaul Concept Refinement Process Model Phase*

4.1 Domain

The *Overhaul Concept Refinement Process Model* was created for legacy enterprise software applications. Enterprise software applications are business-oriented software applications that typically perform business functions such as customer information management, employee information management, order processing, inventory management, and etc. They typically reside on servers or mainframes and provide services to many end-users simultaneously. In this thesis, the phrase "legacy enterprise software applications" refers to enterprise software applications that where originally built and deployed more than a decade ago and have been subjected to maintenance, but not regular overall product evolution or upgrade. Maintenance, in this context, refers to miscellaneous repairs or modifications made to the software application, post-deployment, without the intent of altering the overall architecture, technology, or quality of the software application. Enterprise software applications are not

single-user software applications, such as Microsoft Word 2010, Microsoft Windows 8, Adobe Reader, or Google SketchUp. Single-user software applications like these are typically executed on personal computers and undergo regular and planned overall product evolution. For example, every few years Microsoft releases a new version of the Windows operating system and Google releases the latest version of SketchUp.

A typical software lifecycle model has the following phases: Conception, Requirements, Design, Implementation, Testing, Deployment, Maintenance, and Retirement. A typical definition of a *Software Development Process* includes the following phases: Conception, Requirements, Design, Implementation, Testing and Deployment. This thesis argues that all software application lifecycles must begin with conception of the software product, but *Software Development Processes* are only required after conception and therefore should not include a Conception phase (see Figure 30). First there must be a concept, and then a *Software Development Process*, in order to develop (realize) the concept. It is inappropriate for a *Software Development Process* to include a Conception phase because *Software Development Processes* do not enable the conception of a software product. They enable the construction of a software product based on a concept that was conceived during the Conception phase of the software lifecycle model. For the purposes of this thesis, the Conception phase will be understood as being separate from and preceding the *Software Development Process* and belonging to the software lifecycle model (see Figure 30).

Software Development Processes aim to ensure software applications are built the right way. The *Overhaul Concept Refinement Process Model* aims to mitigate the risk of building the incorrect software product via an inappropriate *Software Development Process*. In other words, it aims to ensure the software concept is valid before using a *Software Development Process* to

correctly build the software product. Some *Software Development Processes* attempt to guide the selection of architectural styles, but this is inappropriate. This does not mean that *Software Development Processes* should not explicitly state the type of *Software Architecture* they are best suited to develop (e.g. IMB's Rational Unified Process recommends and is best suited for component style architectures). The selection of *Software Architecture* should precede the selection of the *Software Development Process*. The *Software Development Process* should not be used to determine the architectural style of a software application. The problem/concept being solved/realized should dictate the architectural style, and then the combination of problem/concept and architectural style should dictate the *Software Development Process*. This is because particular combinations of problem/concept and architectural style are most effectively realized via certain *Software Development Processes*. For example, problem/concept "X" is best solved/realized through implementing architectural style "Y" which is best developed by implementing *Software Development Process* "Z". Consider building a bridge to span two land masses. The physical conditions (the problem) are most suitable for a suspension bridge (the architectural style) which requires a specific building process (the development process) created for suspension bridge construction. The specific nature of the problem must dictate the architectural style required; and these taken together determine the development process employed.

There is not a single *Software Development Process* that is capable of optimally handling every software development situation. Every situation is unique and must be handled accordingly. For example, software development situation "A" requires the development of a safety critical software application. Therefore, the *Cleanroom Software Development Process*

that focuses on defect prevention rather than defect removal may be a more appropriate *Software Development Process* than the *Extreme Programming Software Development Process*.

There are software development organizations that mandate the use of the *Rational Unified Process* (RUP) to develop software, but RUP is not always suitable for every software development situation. Software development organizations that have predetermined the *Software Development Process* they will use on all of their software development projects are at risk of using an ineffective *Software Development Process* on certain software development projects. Therefore, software development organizations need a process model that can help them refine the concept of the software product and guide them in the selection of the architectural style and *Software Development Process*, case by case. The *Overhaul Concept Refinement Process Model* is that type of process model, designed specifically for Overhaul concepts of legacy enterprise software applications.

Figure 30 outlines the hierarchical Software Lifecycle Model of enterprise software applications for the purpose of this thesis. It is read from left to right and from top to bottom, and illustrates that the "Software Lifecycle" has 5 potential high-level phases: Conception, Development, Maintenance, Overhaul, and Retirement. Each second level phase implies a strategy to be implemented via tactics illustrated in the third level of the hierarchical structure. Although "Concept Refinement" (shown in the third level) is a tactic (child) of "Conception" (shown in the second level), it is the all-embracing strategy of the *Overhaul Concept Refinement Process Model*, which in turn implements its own tactics to achieve "Concept Refinement" and ultimately "Conception". Therefore, "Concept Refinement" is the domain of the *Overhaul Concept Refinement Process Model*. The clouds depicted to the far right of Figure 30 specify which third level tactics must be planned, monitored, and controlled (managed) to be effective.

They are depicted as clouds because the specifics of the management activities are beyond the scope of the Software Lifecycle Model.

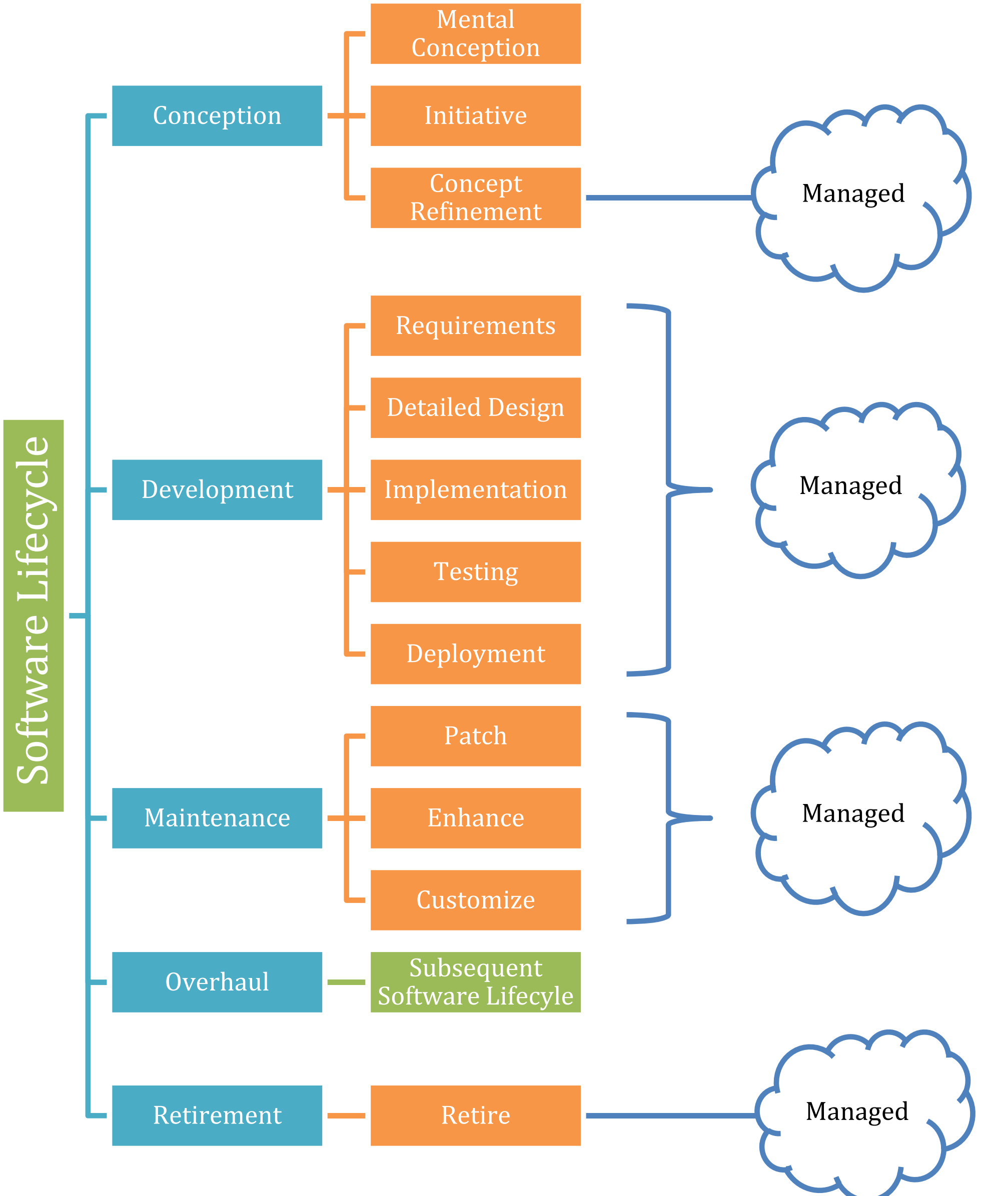

Figure 30: *Enterprise Software Lifecycle Model*

4.2 Flow

This section analyzes the rationale of the organizational flow of the *Overhaul Concept Refinement Process Model*. The *Overhaul Concept Refinement Process Model* is a high-level

process model that can be used to refine a specific type of abstract concept. Its means of concept refinement and its ultimate output are necessarily high-level (i.e., more abstract than *Software Development Processes*, but less abstract than the initial concept). It is comprised of seven phases that are executed sequentially. Listed below (in order of execution) are the seven phases of the *Overhaul Concept Refinement Process Model*:

1. Understand the Existing Issues
2. Research Software Development Trends and Technology
3. Identify Remedies for Existing Issues
4. Collectively Analyze and Conceptualize Solutions
5. Select a Solution to Refine
6. Refine Solution with Rapid Prototyping
7. Final Preparation for Software Development

Figure 1 illustrates the flow listed above. Furthermore, it illustrates the potential feedback loops that exist between Phase 2 and Phase 1, and Phase 6 and Phase 5. In other words, the result of Phase 2 may further enable activities in Phase 1, and the result of Phase 6 may require Phase 5 to be repeated.

The initial concept that leads to the overhauling of a legacy enterprise software application is typically born from the realization of critical issues (problems) present in the existing software system. Therefore, the objective of Phase 1 is to understand the existing issues (problems) of the software application as a team. In order to effectively solve a problem, the problem must first be understood. For example, consider a simple math problem where some positive integers summed equal some greater positive integer. The solution (the greater positive integer) cannot be reasonably determined if both lesser positive integers are unknown to the problem solver. This rationale applies to refining an initial overhaul concept. Before a solution can be determined, the problem must be understood.

Phase 2 of the *Overhaul Concept Refinement Process Model* focuses on researching items that apply to the problem being solved. For example, consider a medical doctor treating a patient's condition. The particular circumstances relating to the patient's condition may suggest a variety of types of antibiotics for treatment. As new drugs become available on a regular basis, the treating physician may need to research the benefits and drawbacks of each potential antibiotic in order to provide the best treatment. The rationale illustrated in this example is similar to that of the *Overhaul Concept Refinement Process Model.* Phase 2 (Research) is required to aid in the later determination of the best possible solution. Research brings new knowledge to the team, and in turn, enables better understanding of the problem, which further enables problem solving. Moreover, research may uncover existing issues that were previously unknown. Phase 2 must follow Phase 1 because the output of Phase 1provides the guidance necessary to select research items in Phase 2.

Phases 1 and 2 of the *Overhaul Concept Refinement Process Model* brought a common knowledge of the existing issues to the team, and armed them with the necessary knowledge to begin Phase 3 (identifying potential high-level remedies for the issues discovered and documented in Phase 1). The order of these Phases is important. The identification of potential remedies for issues (Phase 3) cannot precede Phase 1. This is because remedies for issues cannot be determined before the issues themselves are identified. Moreover, Phase 3 cannot precede Phase 2 because Phase 2 provides the means by which remedies can be assigned to an issue. Therefore, it can be concluded that Phase1 must precede Phase 2, and Phase 2 must precede Phase 3.

Phase 4 of the *Overhaul Concept Refinement Process Model* examines the documentation created in Phase 3 and uses the information in that documentation to begin to effectively

conceptualize overall solutions to the problem. Therefore, Phase 3 must precede Phase 4. The overall solutions conceptualized in Phase 4 must consist of an architectural style that is appropriate for the problem, as well as a *Software Development Process* that is optimal for the combination of problem and architectural style.

The overall conceptualized solutions conceived in Phase 4 are the input required by Phase 5 of the *Overhaul Concept Refinement Process Model*. Therefore, Phase 4 must precede Phase 5. Phase 5 assesses the overall conceptualized solutions created in Phase 4 and determines the order in which they would most likely solve the overall problem. This enables the selection of the best possible overall solution to further refine via Phase 6.

Phase 6 of the *Overhaul Concept Refinement Process Model* implements the *Software Development Process* selected in Phase 5 to rapidly construct a prototype in the selected architectural style. To do this, a subset of the existing legacy software system's functionality is recreated via the *Software Development Process* and is implemented in the new architectural style. Phase 5 must precede Phase 6 because Phase 6 implements the selected overall conceptualized solution as determined in Phase 5. If Phase 6 fails to substantiate the selected overall conceptualized solution (architectural style and Software Development Process) then Phase 5 can be repeated and another overall conceptualized solution can be selected for further refinement via Phase 6. Moreover, it is important to note that the *Rapid Prototyping Process Model* is iterative in nature and can be executed multiple times to further refine and optimize the architectural style and *Software Development Process* as necessary.

Phase 7 of the *Overhaul Concept Refinement Process Model* uses the knowledge gleaned from executing Phases 1 – 6 to scope and plan the overhaul software development project. Therefore, Phases 1 – 6 must precede Phase 7 because Phase 7 cannot be executed until Phase 6

has substantiated the practicality of the refined overhaul concept. Furthermore, Phase 7 is the final phase in the *Overhaul Concept Refinement Process Model* and is intended to be followed by the execution of a well-planned *Software Development Process* that transitions the concept into a concrete reality (i.e., a successfully overhauled enterprise software application).

4.3 Sub Processes

This section will analyze the rationale of the sub processes of the *Overhaul Concept Refinement Process Model*. Bear in mind, the sub processes are tactical guidelines that are intended to help achieve the overall strategy of the specific *Overhaul Concept Refinement Process Model* phase within which it resides. Each sub process is comprised of steps, which in turn are detailed via descriptions. The steps are the tactics of the sub process and the steps' descriptions are the tactics of the distinct steps. The tactics necessary to appropriately implement each step, per its description, must be determined collaboratively by the overhaul concept refinement team. These objectively established low-level tactics are therefore objective only to the particular concept refinement team that established them.

Consider the hierarchical command structure of the United States Marine Corps shown in Figure 31. Here, "Commander and Chief (President)" is the highest level and "Captain" is the lowest level. During a war, the strategies and tactics implemented to win the war are also hierarchical and are realized differently at different levels of the command structure. For example, consider the hierarchical structure illustrated by Figure 32. The Commander and Chief's tactic level item for being a successful leader is "Win Country A". However, "Win Country A" is the strategy level item to the Marine Corps Generals. The tactic level items for the Marine Corps Generals are "Win Region A" and "Win Region B". Therefore, the "Win Region"

level items are the strategy level items to the Marine Corps Colonels, and in turn, the tactic level items for the Marine Corps Colonels are the "Win City" level items. Therefore, "Win City" level items are the strategy level items to the Marine Corps Captains which make their tactic level items the "Win Field" level items. This example illustrates how strategies are implemented through tactics, and how tactics at one level become the strategies for the levels below.

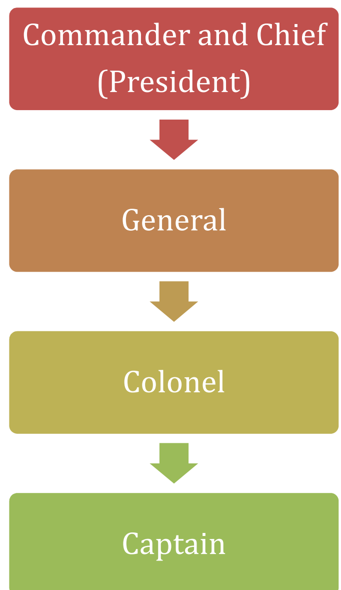

Figure 31: *Hierarchical Command Structure of the United States Marine Corps*

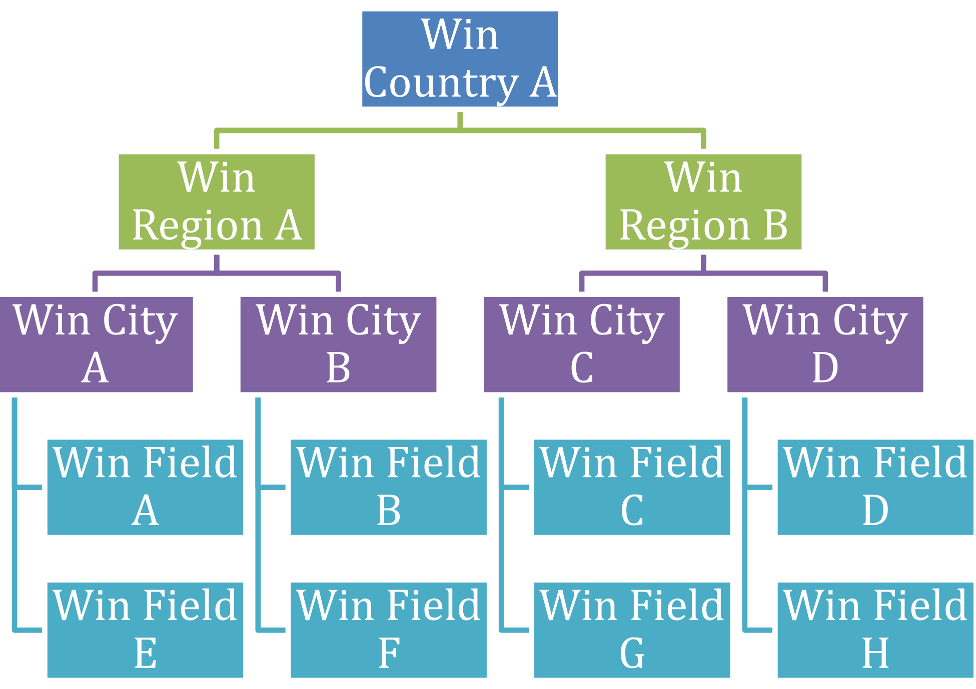

Figure 32: *Hierarchical Structure of Strategies and Tactics*

In the *Enterprise Software Lifecycle Model*, "Concept Refinement" is a tactic of "Conception". This thesis defines the *Overhaul Concept Refinement Process Model* as being a tactic for "Concept Refinement" of a legacy enterprise software application that is being overhauled. The tactics of the *Overhaul Concept Refinement Process Model* are its seven phases (e.g., *Understand the Existing Issues*). The tactics of the phases are the sub processes that exist within the distinct phases (e.g., *Issue Discovery Process Model*). The tactics of the sub processes are the steps of the sub processes (e.g., *Model*). The tactics of the steps are their individual descriptions (e.g., "Construct architectural views of the critical architectural elements."). Objectively establishing and executing the tactics of the sub processes' steps' descriptions is the responsibility of the overhaul concept refinement team. Outlining, examining, and exemplifying the tactics of these descriptions is beyond the scope of this

thesis. It is not the objective of this thesis to develop or detail activities such as *Modeling*, *Brainstorming*, *Analyzing*, *Selecting*, *Reviewing*, *Ranking*, or *Recording*, because the tactics involved in these low-level activities (as compared to higher level "Concept Refinement" activities) are well examined and outlined in other published works, and can be considered to belong to the common knowledge of the software engineering discipline.

4.4 Artifacts

This section analyzes the general rationale behind the artifacts of the *Overhaul Concept Refinement Process Model*. The artifacts of the *Overhaul Concept Refinement Process Model* are listed below.

1. Models that describe critical architectural elements from certain viewpoints.
2. A *Symptoms* document listing each reported symptom, and stakeholder group that reported the symptom.
3. An *Issues* document listing and describing each identified issue.
4. A *Symptoms/Issues* cross reference document, linking symptoms to issues.
5. A *Research Plan* document listing research items each designated with a time allocation, and an ultimate deadline for the overall research effort.
6. A document that associates ranked potential remedies with the documented issues.
7. High-level graphical models of potential Software Architectures.
8. High-level graphical models of potential Software Development Processes.
9. A prioritized list of architectural concepts paired with a development process.
10. A prototype software product.

The artifacts of the *Overhaul Concept Refinement Process Model* are necessarily defined at a level of abstraction consistent with the sub process responsible for creating them. Objectively establishing and executing the tactics of the sub processes' steps' descriptions is the responsibility of the overhaul concept refinement team. Therefore, a lower level description and detailed implementation process of the artifacts created by those sub processes should only be defined by the overhaul concept refinement team. Moreover, it is their responsibility to design and construct the artifacts in such a way as to be objective to

their team and the high-level strategy behind them. For example, consider the *Issues* document created in the case study found in Chapter 3. To an outsider (i.e., someone that was not a member of Service Central's overhaul concept refinement team) the wording used to describe the issues in their *Issues* document may appear to be subjective (e.g., "Spaghetti code throughout code modules"). However, the issues' descriptions are not subjective because the entire team held a common understanding of the detailed meaning behind each issue's description. Something is subjective if it comes forth from a single individual's opinion or interpretation [35]. Objective is the exact opposite of subjective [35]. Therefore, by definition, it can be concluded that the wording used in the case study's *Issues* document is objective because each issue's description held a common understanding among the entire overhaul concept refinement team. Furthermore, members of the overhaul concept refinement team were the only people responsible for analyzing or working with the *Issues* document (and all other artifacts) during the execution of the *Overhaul Concept Refinement Process Model*.

The *Issues* document created in the case study exemplifies the efficiency and effectiveness of using objectively established high-level documentation. For example, consider high-level computer programming languages (e.g., C++). They bring efficiency and effectiveness to the construction of software applications that pure low-level binary code cannot. This rationale applies to artifacts created during the execution of the *Overhaul Concept Refinement Process Model*, but only if the abstraction of the details are objectively established.

4.4 Usability

This section briefly examines usability, which is an important quality attribute of the *Overhaul Concept Refinement Process Model*. Usability refers to how easily a process model can be learned and used by an organization. The case study found in Chapter 3 is concrete evidence that the *Overhaul Concept Refinement Process Model* is usable. Furthermore, it exemplifies the usability of the *Overhaul Concept Refinement Process Model* by illustrating the potential simplicity of its artifacts (i.e., *Symptoms* document, *Issues* document, and etc.). Service Central's overhaul concept refinement team needed only a few hours to study the process model before implementing it against Service Central, which speaks to its usability. However, the degree to which the *Overhaul Concept Refinement Process Model* is usable cannot be objectively stated without an explicitly defined, objectively established metric. The establishment of this metrics is beyond the scope of this thesis, but that fact the *Overhaul Concept Refinement Process Model* has been used and only required a few hours to learn is notable [23, 24, 25].

4.5 Repeatability

This section briefly examines repeatability, which is an important quality attribute of the *Overhaul Concept Refinement Process Model*. Repeatability describes the degree to which a process can be repeated. The extensive documentation of the *Overhaul Concept Refinement Process Model* found in Chapter 2 is concrete evidence that the *Overhaul Concept Refinement Process Model* is repeatable because it is documented. To further substantiate this claim, upon completion of the of the initial execution of the *Overhaul Concept Refinement Process Model*, Service Central's overhaul concept refinement team determined (based on their experience and process documentation) that the process model could be repeated on Service Central in the

future, if it ever needed to be overhauled again. However, the degree to which the *Overhaul Concept Refinement Process Model* is repeatable cannot be objectively stated without an explicitly defined, objectively established metric. The establishment of this metrics is beyond the scope of this thesis, but that fact the *Overhaul Concept Refinement Process Model* is documented, has been used, and has been determined to be repeatable by a software development organization is notable [23, 24, 25].

4.6 Limitations

The *Overhaul Concept Refinement Process Model* requires personnel with technical expertise on the existing legacy software application. Without these individuals, the ability to effectively use the *Overhaul Concept Refinement Process Model* is greatly diminished. For example, Service Central's overhaul concept refinement team included two experts on Service Central's technical implementation. These two engineers brought the necessary depth of understanding on many of Service Central's existing issues to the overhaul concept refinement team. Technical experts with adequate technical knowledge are necessary to sufficiently bring understanding of the legacy system's issues to the entire overhaul concept refinement team because the *Overhaul Concept Refinement Process Model* basis many of its processes off of the assumption that the entire team has an accurate understanding of the existing issues (Phase 1).

In addition, the *Overhaul Concept Refinement Process Model* is intended to focus on a single legacy software application at a time. If an organization has many legacy systems that need to be overhauled simultaneously, a separate instance of the *Overhaul Concept Refinement Process Model* must be implemented for each legacy system (refer to Figure 30). This requires a

well-defined management approach in order to adequately support the parallel executions of the *Overhaul Concept Refinement Process Model.*

4.7 Related Works

The following sections briefly examine process models related to overhauling software applications. Each section will illustrate how the *Overhaul Concept Refinement Process Model* integrates with the related process and how it is different.

IBM's Rational Unified Process

The Rational Unified Process (RUP®) created by IBM attempts to be a comprehensive framework for developing software. IBM claims that RUP improves software development project performance with proven, adaptable processes [36]. IBM promotes RUP with the following highlights [36]:

- "Enhances productivity with industry-proven configurable techniques and practices to fit individual project needs."
- "Supports team collaboration and individual practitioners with context-sensitive guidance across geographies and functions."
- "Enables early risk mitigation using iterative processes centered around business priorities and stakeholder needs."
- "Promotes organizational transformation with comprehensive education services and an extensive consultant and partner ecosystem."

RUP is widely recognized as being a *Software Development Process* because its 4 high-level process phases (i.e., Inception, Elaboration, Construction, and Transition) approximately map to the high-level phases of the widely known *Waterfall* development process model. The *Waterfall* process model's phases are Conception, Requirements, Design, Implementation, Testing, and Deployment.

The *Overhaul Concept Refinement Process Model,* unlike RUP, is not a comprehensive framework for developing software. Furthermore, it does not aim to be a "one size fits all" type of process model like RUP. The *Overhaul Concept Refinement Process Model* has a much more specific domain than that of RUP and its goals and strategies are different. The goal of RUP is to enable a software development organization to design, construct, test, and deploy a successful software product effectively. The goal of the *Overhaul Concept Refinement Process Model* is to refine a specific type of concept, conceived by a specific type of organization, for a specific category of software, to a point that enables realization of the concept by means of a managed *Software Development Process*. Even though RUP and the *Overhaul Concept Refinement Process Model's* sub processes' steps may share similar tactics (e.g., modeling), this does not mean these tactics will be implemented in the exact same way or for the same reason. At their strategy levels, comparing RUP to the *Overhaul Concept Refinement Process Model* is analogous to comparing an apple to an orange. However, referring to Figure 30, it is possible for the *Overhaul Concept Refinement Process Model* to have been executed during the "Conception" phase of an enterprise software application's lifecycle and RUP to have been executed during the "Development" phase.

TmaxSoft's Re-architecting Process

According to TmaxSoft, "Re-architecting is the process of integrating your legacy applications into SOA-based (service oriented architecture) open system deployments." TmaxSoft's "Re-architecting" process model is advertised to be a 5-step "legacy modernization pathway". The 5 steps of TmaxSoft's Re-architecting process are listed below [37].

1. **Analyze the target legacy system** – “Experienced TmaxSoft re-architecting engineers work with the administrators and developers of the target legacy system to fully map out the target system and determine the architecture requirements of the new system.”
2. **Design the architecture/logic for the new system** – “TmaxSoft re-architecting engineers draw up plans for the architecture of the new system. Then the business logic within the target system is further analyzed and the logic is separated into two groups: that which can simply be replaced by the common logic modules provided by ProFrame framework, and that which needs to be redeveloped.”
3. **Develop the new system** – “Following the plans determined in Step 2, ProFrame is installed as the framework for the new system, providing a base upon which the re-architected business logic will operate. ProFactory is then installed and used to redevelop the business logic from the legacy system as a series of independent, reusable modules. ProFactory provides a range of tools for building these logic modules in an intuitive, graphic manner. This greatly reduces the need for coding in business logic re-development. ProFactory is then used to link together the redeveloped business modules and the modules provided by the ProFrame framework. This recreates the logic flow from the original legacy system. This is again a visual process in which GUI tools are used to link together the various modules. The new system will be fully grounded in SOA concepts, making it far more flexible and transparent than the original legacy system.”
4. **Test the new system** – “The new system should be tested and tuned. These include business logic tests, service reliability tests, performance tests, consistency tests, etc.”
5. **Train the local developers/users** – “Training is actually undertaken during all steps of the re-architecting process. TmaxSoft engineers teah the client's developers and system users how to best use the ProFrame framework and develop/modify business logic using the GUI provided.”

The advertised benefits of TmaxSoft's “Re-architecting” process model are listed below [37].

1. “Re-architecting connects legacy business logic with modern technologies and concepts.”
2. “Re-architecting can evolve legacy applications into SOA-based deployments.”
3. “The new system will require less time spent coding when modifying or developing logic.”
4. “By being based on SOA concepts and built on an advanced framework, the new system will be flexible, transparent, and reliable.”
5. “The new system will be expandable without the danger of a 'spaghetti architecture' emerging.”

TmaxSoft’s Re-architecting process model is a proprietary process of TmaxSoft and is intended to be executed by TmaxSoft engineers. Its goal is to overhaul legacy enterprise software applications by re-architecting them into SOA-based software deployments. It is a type of *Software Development Process* specifically designed for overhauling legacy enterprise software application in a predetermined architectural style (i.e., SOA).

The *Overhaul Concept Refinement Process Model* could (in a certain situation) drive the selection of TmaxSoft's Re-architecting process as the most appropriate development solution for a software development organization that has an initiative to overhaul their legacy enterprise software application. Referring to Figure 30, it is possible for the *Overhaul Concept Refinement Process Model* to be executed during the "Conception" phase of an enterprise software application's lifecycle and TmaxSoft's Re-architecting process to be executed during the "Development" phase.

4.8 Conclusion

In conclusion, the *Overhaul Concept Refinement Process Model* is usable and repeatable [26]. Moreover, as illustrated in Chapter 3, it can be used to refine the new overall architectural and software development concepts associated with overhauling a legacy enterprise software application. If an idea or concept is poor, it "fails fast" in the rapid prototyping phase, which prevents resources being wasted on developing a product based on a poor concept. The *Overhaul Concept Refinement Process Model* extends the typical conceptualization phase of an enterprise software lifecycle model of a typical legacy system by providing an essential mechanism for refining the overhaul concept of the legacy system.

REFERENCES

[1] M. A. Serrano, D. L. Carver, and C. M. Oca. “Reengineering legacy systems for distributed environments.” *The Journal of Systems and Software*. vol. 64, pp. 37-55, 2002.

[2] M. Colosimo, A. Lucia, G. Scanniello, G. Tortora. “Evaluating legacy system migration technologies through empirical studies.” *Information and Software Technology*. vol. 51, pp. 433-447, 2009.

[3] K. Jamsa, “Securing the Cloud,” in *Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile Security, and More*, Burlington, MA, Jones & Bartlett Learning, 2012, ch. 9, pp. 127-139.

[4] J. R. Leary. "An Architectural Basis for Evolving Software Systems." *J. Systems Software*. vol. 30, pp. 27-43, 1995.

[5] E. J. Braude and M. E. Bernstein, “Software Process,” in *Software Engineering: Modern Approaches*, 2nd ed. John Wiley & Sons, Inc., 2011, ch. 3, pp. 32-62.

[6] C. Pahl. "Adaptive development and maintenance of user-centric software systems." *Information and Software Technology*. vol. 46, pp. 973-986, 2004.

[7] D. Graham, E. V. Veenendaal, I. Evans, R. Black, “Testing Throughout the Software Life Cycle,” in *Foundations of Software Testing*, Revised ed. Prentice Hall, C&C Offset, China, 2009, ch. 2, pp. 35-56.

[8] E. J. Braude and M. E. Bernstein, “Principles of Requirements Analysis,” in *Software Engineering: Modern Approaches*, 2nd ed. John Wiley & Sons, Inc., 2011, ch. 10, pp. 230-244.

[9] E. J. Braude and M. E. Bernstein, “Project Management,” in *Software Engineering: Modern Approaches*, 2nd ed. John Wiley & Sons, Inc., 2011, ch. 7, pp. 140-167.

[10] J. Henry, “Write Your Plan,” in *Software Project Management: A Real-World Guide to Success*, Pearson Education, Inc., 2004, ch. 8, pp. 157-174.

[11] K. Jamsa, “Introducing Cloud Computing,” in *Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile Security, and More*, Burlington, MA, Jones & Bartlett Learning, 2012, ch. 1, pp. 1-16.

[12] K. Jamsa, “Mobile Cloud Computing,” in *Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile Security, and More*, Burlington, MA, Jones & Bartlett Learning, 2012, ch. 14, pp. 203-216.

[13] R. Daigneau and I. Robinson, "From Objects to Web Services," in *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services*, Pearson Education, Inc., 2012, ch. 1, pp. 1-11.

[14] K. Jamsa, "Service-Oriented Architecture," in *Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile Security, and More*, Burlington, MA, Jones & Bartlett Learning, 2012, ch. 11, pp. 161-176.

[15] S. Tariq, M. Nazir, and F. Saleemi. (2012, Feb.) "Enhancement of XP for Cloud Application Development." *Journal of Emerging Trends in Computing and Information Sciences*. [on-line] Available: http://www.cisjournal.org [Nov. 04, 2012].

[16] S. Mitchell, J. Owen, and K. Warr. (2004, Aug.) "An adventure in Extreme Programming." *IBM WebSphere Developer Technical Journal*. [on-line] Available: http://www.ibm.com/developerworks/websphere/techjournal/0408_mitchell/0408_mitchell.html [Nov. 04, 2012].

[17] G. R. Frederick and R. Lal, "Introduction to Mobile Web Development," in *Beginning Smartphone Web Development: Building JavaScript, CSS, HTML and Ajax-based Applications for iPhone, Android, Palm Pre, BlackBerry, Windows Mobile, and Nokia S60*, Apress, 2009, ch. 1, pp. 1-14.

[18] T. Erl, B. Carlyle, C. Pautasso, and R. Balasubramanian, "Introduction to Services," in *SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST*, Prentice Hall, 2011, ch. 3, pp. 23-30.

[19] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures." Ph.D. dissertation, Computer Science Dept., Univ. of California, Irvine, CA, 2000.

[20] T. Erl, B. Carlyle, C. Pautasso, and R. Balasubramanian, "REST Constraints and Goals," in *SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST*, Prentice Hall, 2011, ch. 5, pp. 51-66.

[21] T. Erl, B. Carlyle, C. Pautasso, and R. Balasubramanian, "Service-Orientation with REST," in *SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST*, Prentice Hall, 2011, ch. 7, pp. 93-126.

[22] G. R. Frederick and R. Lal, "Testing a Mobile Web Site," in *Beginning Smartphone Web Development: Building JavaScript, CSS, HTML and Ajax-based Applications for iPhone, Android, Palm Pre, BlackBerry, Windows Mobile, and Nokia S60*, Apress, 2009, ch. 10, pp. 259-272.

[23] E. J. Braude and M. E. Bernstein, "Software Architecture," in *Software Engineering: Modern Approaches*, 2nd ed. John Wiley & Sons, Inc., 2011, ch. 18, pp. 438-475.

[24] K. Jamsa, "Application Scalability," in *Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile Security, and More*, Burlington, MA, Jones & Bartlett Learning, 2012, ch. 19, pp. 277-290.

[25] A. Terazzi, A. Giordano, G. Minuco. "How can usability measurement affect the re-engineering process of clinical software procedures?" *International Journal of Medical Informatics*. vol. 52, pp. 229-234, 1998.

[26] K. Tahera, R.N. Ibrahim, P.B. Lochert. "A fuzzy logic approach for dealing with qualitative quality characteristics of a process." *Expert Systems with Applications*. vol. 34, pp. 2630-2638, 2008.

[27] G. R. Frederick and R. Lal, "The Future of the Mobile Web," in *Beginning Smartphone Web Development: Building JavaScript, CSS, HTML and Ajax-based Applications for iPhone, Android, Palm Pre, BlackBerry, Windows Mobile, and Nokia S60*, Apress, 2009, ch. 13, pp. 303-314.

[28] A. Freeman, "Putting HTML5 in Context," in *The Definitive Guide to HTML5*, Apress, 2011, ch. 1, pp. 1-2.

[29] *Introducing JSON*. [on-line] Available: http://www.json.org/ [Nov. 12, 2012].

[30] E. J. Braude and M. E. Bernstein, "Agile Software Processes," in *Software Engineering: Modern Approaches*, 2nd ed. John Wiley & Sons, Inc., 2011, ch. 4, pp. 63-79.

[31] S. Dong, M. Johar, R. Kumar. "Understanding key issues in designing and using knowledge flow networks: An optimization-based managerial benchmarking approach." *Decision Support Systems*. vol. 53, pp. 646-659, 2012.

[32] J. A. Robinson, "Researching possible solutions," in *Software Design for Engineers and Scientists*, Newnes, 2004, ch. 11, pp. 255-275.

[33] J. A. Robinson, "Design methodology," in *Software Design for Engineers and Scientists*, Newnes, 2004, ch. 9, pp. 235-243.

[34] J. A. Robinson, "Detailed design and implementation," in *Software Design for Engineers and Scientists*, Newnes, 2004, ch. 13, pp. 288-297.

[35] *Subjective.* [on-line] Available: http://dictionary.reference.com/browse/subjective [Dec. 26, 2012].

[36] *IBM Rational Unified Process (RUP).* [on-line] Available: http://www-01.ibm.com/software/awdtools/rup/ [Dec. 26, 2012].

[37] *Re-architecting*. [on-line] Available: http://us.tmaxsoft.com/ [Dec. 26, 2012].

APPENDIX A: BACKGROUND INFORMATION

This appendix provides background information on various topics that were touched upon in the case study presented in Chapter 3.

A.1 Cloud Computing

Currently, cloud computing is a phenomenon in the computer science, business, and personnel computing realms. The phrase “cloud computing” refers to the abstraction of Web-based computing resources and services that enable software developers to create, deploy, and maintain complex distributed hypermedia applications on virtualized remote resources. In a nutshell, cloud computing removes the necessity of having data, software applications, and computing resources on local devices (i.e., laptops, smartphones, and desktops). Instead of being local to client devices, these resources exist within the “cloud” and are accessed via the Internet. The massive Internet infrastructure that has evolved over recent decades is the primary mechanism that has made cloud computing possible. As a result, cloud computing is enabling people and software systems to access, store, and process data on a massive scale [11, 12].

A.2 Service Oriented Architecture

A Service Oriented Architecture (SOA) is an architectural style that is prevalent among modern software applications. SOA aims to deliver software applications that are scalable, interoperable, maintainable, and highly performant. A *service* is a software program that makes available its functionality through a technical interface. A *service's* technical interface is

typically called a *service contract*, and is comprised of the *service capabilities*. For example, a Work Order *service* may publish a *service contract* that expresses *service capabilities* like:

- Get Existing Work Order
- Create New Work Order
- Update Existing Work Order
- Delete Existing Work Order

A *service consumer* is a software program that accesses and invokes a *service* and consumes the *service's* capabilities. A SOA establishes and constrains the characteristics of how and for what purpose *services*, *service contracts*, *service capabilities*, and *service consumers* can be implemented and how they relate to each other [13, 14, 18].

A.3 Representational State Transfer

Representational State Transfer (REST) is an architectural style for distributed hypermedia systems (e.g., Web applications). The REST style architecture was introduced and defined in 2000 by Dr. Roy Fielding in his doctoral dissertation [19]. Although REST has become a buzzword among API (application program interface) developers, very few software applications have been built that fully comply with Fielding's definition of REST. According to Fielding:

> The name "Representational State Transfer" is intended to evoke an image of how a well-designed Web application behaves: a network of Web pages (a virtual state-machine), where the user progresses through the application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use [19].

Fielding's REST constraints are:

1. Client-server
2. Stateless server
3. Cache
4. Uniform interface

   a. Identification of resources (resource identifier identifies the specific resource involved in an interaction between components)
   b. Manipulation of resources through representations (resource representation represents the state of a resource for transfer between components)
   c. Self-descriptive messages contain all the information necessary to complete transformations
   d. Hypermedia as the engine of application state
5. Layered system
6. Code-On-Demand (optional)

Many software developers believe that Fielding's explicit definition of REST is too difficult to adhere to in real-world software development. As a result, the term "pragmatic REST" has become popular for describing software applications that implement a subset of Fielding's constraints. While Fielding's definition of REST has driven the development of many software applications that partially comply with the formal constraints of the REST style, only a small percentage of software applications fully comply with all of Fielding's constraints [19, 20, 21].

The first three constraints as illustrated in Figure 33 are the three constraints many Web API developers adhere to and refer to as pragmatic REST.

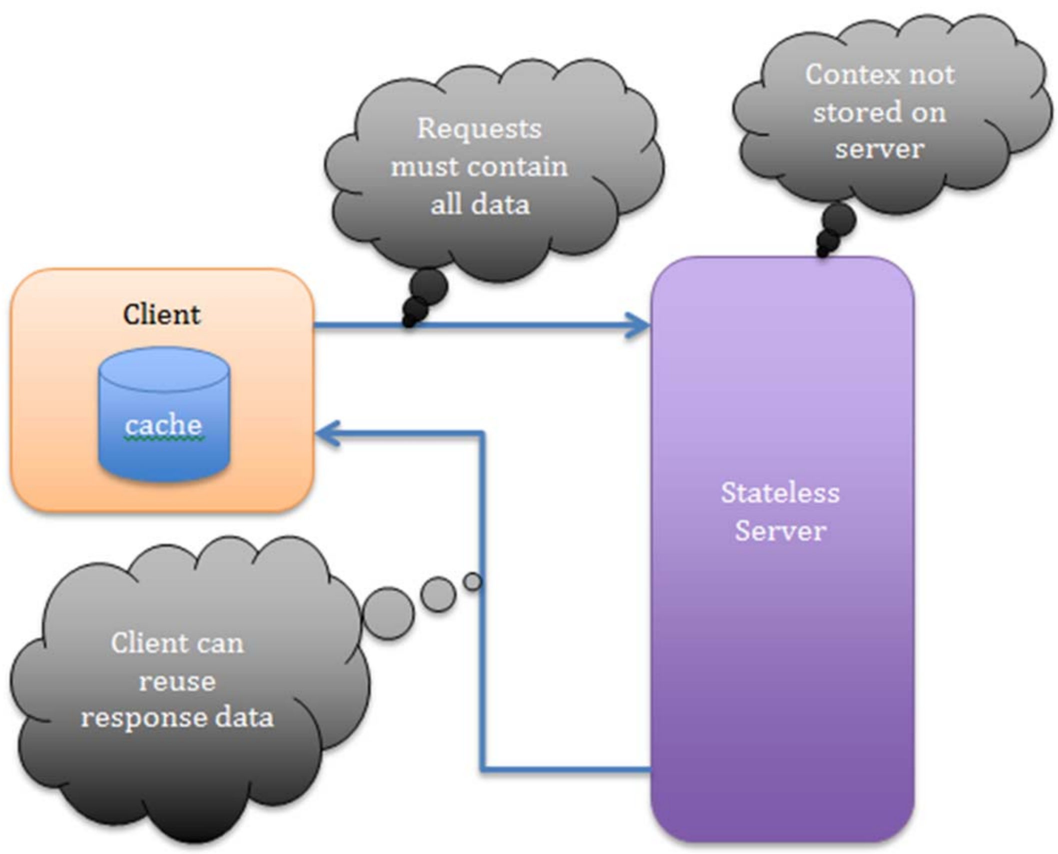

Figure 33: *Pragmatic REST Constraints*

A.4 Mobile Web

The Mobile Web is a rapidly growing digital medium. It can be thought of as being the traditional Internet translated for use by mobile devices (i.e., smart phones, feature phones, tablets, etc.). Developing software for the Mobile Web has a unique set of best practices; this is why a distinction is made between the Desktop Web and the Mobile Web [17].

The popularity of the Mobile Web is obvious to nearly everyone who owns a mobile device. The Mobile Web is a phenomenon sweeping the planet; people are using mobile devices to perform tasks that would traditionally require a desktop computer with a high bandwidth Internet connection. The emergence of widespread accessibility to information is partially due to

the Mobile Web. Its realization is facilitating the development of new ways to do business, communicate, play, and many other activities [17].

While software development standards for the Mobile Web are immature, they are emerging and will undoubtedly become well established in the coming years. Moreover, tools and tool kits are surfacing that make software development for the Mobile Web more readily achievable by mainstream software developers. For example, JQuery Mobile is a popular JavaScript library that expedites client-side software development of Mobile Web applications [17].

The Mobile Web puts data in people's hands while they are on the go. It is new, but rapidly maturing. Adoption of device standards by manufacturers and software development standards by software developers looks promising. The Mobile Web is becoming the predominate digital medium for information sharing among the planet's general population [27].

A.5 Hypertext Markup Language 5

HTML is an acronym for Hypertext Markup Language. HTML originated in the early 1990s and has become the primary method for marking up data for interpretation by mainstream Web browsers. That is, Web browsers translate HTML documents into the graphical user interfaces (GUIs) presented to end-users of Websites [28].

HTML5 is more than just the latest HTML specification. HTML5 is used by many software developers to refer to a set of related Web technologies: HTML, CSS (Cascading Style Sheets), and JavaScript. The HTML5 specification strongly relies on the existence of CSS and JavaScript, and therefore cannot be teased apart from them. Each of the fundamental core technologies behind HTML5 plays a specific role. The HTML is the semantic markup of a

Webpage’s content, CSS applies presentation characteristics to the semantic HTML document, and JavaScript is the client-side programming language that can dynamically alter the HTML and style on the fly [28].

Moreover, HTML5 intrinsically embraces multimedia content unlike its previous versions. With HTML5, video and audio clips are supported natively in the browser and do not require third-party plugins (e.g., Adobe Flash). HTML5 also introduces user defined attributes for HTML elements and many new input types. These powerful new features enable highly dynamic Webpages to be constructed more quickly and with fewer defects than ever before [28].

A.6 JavaScript Object Notation

JavaScript Object Notation (JSON) is a data-interchange format that is based on a subset of the JavaScript programming language. It has become widely popular among many API developers because it is very easy to read, generate, and parse. In many cases, JSON is being used instead of more heavyweight data-interchange technologies like XML [29]. Figure 34 is a snippet of JSON.


```
{
  "Results":[
    {
      "Success":"Ok",
      "TicketNo":"580",
      "RmaNo":"432",
      "NumberOfEntityLines":"2 of 2",
      "NumberOfNoteLines":"3 of 3",
      "ErrorMessage":""
    }
}
```


Figure 34: *Sample JSON Structure*

A.7 Extreme Programming

Extreme Programming, introduced by Kent Beck in the mid-90s, is a type of Agile software development model that has many strengths and some substantial weaknesses [30]. Agile software development models have become popular among many software developers and organizations. Unfortunately, due to the rapid wide-scale adoption of Agile software development models, many software developers and organizations have implemented Agile methodologies poorly; others have implemented Agile models with success, helping to define paths to successful implementations of Agile models.

Agile software development methods, like Extreme Programming, are considered to be "lightweight" methods, when compared with their predecessors [15, 16, 30]. The birth of Agile software development can be attributed to the flaws inherent in traditional, "heavyweight", software development models (e.g., the Waterfall model). This is because, with respect to documentation, traditional software development models are heavily focused on comprehensive documentation, whereas Agile software development models often strive to position the implementation and deployment of working software over the comprehensive documentation of the software. That is, when using traditional software development models, large amounts of project resources are consumed creating, managing, and adhering to documentation artifacts. Agile software development models, on the other hand, focus on working continuously with customers and end-users to quickly and adequately manage complex changing requirements, and strive to continuously deliver working software to the end-users as frequently as possible.

Extreme Programming places specific emphasis on four general Agile software development characteristics [30]:

1. Communication – the continuous communication between developers and customers.

2. Simplicity – the constant selection of the simplest design.
3. Courage – the courage to commit to rapid delivery of functionality.
4. Feedback – feedback loops incorporated throughout development activities.

It is from these four core Agile software development characteristics that the Extreme Programming model is derived. The Extreme Programming process model has four primary phases that are executed iteratively [15, 16]:

1. Planning
2. Design
3. Coding
4. Testing

Additionally, each iteration ends in new completed functionality delivered to the customer. Extreme Programming's planning phase follows this process [15, 16]:

8. Create user stories.
9. Assess user stories.
10. Group user stories.
11. Commit to iteration specific delivery date.
12. Use project velocity to determine delivery dates for subsequent iterations.

User stories are the primary focus of the planning phase of an iteration. A user story is a system requirement expressed in as few unambiguous sentences as possible, and is written by the customer from the perspective of the end-user. Furthermore, user stories are informal statements of the requirements that are prioritized by the customer. Often, user stories are written on small paper cards and then stacked in order of priority. Once user stories have been established they are assessed and grouped for development in distinct development iterations. User stories are grouped based on likeness, effort, priority, and etc. Additionally, project velocity, calculated based on results of previous iterations, can be used to establish delivery dates of the iteration.

The design phase of the Extreme Programming model emphasizes the K.I.S.S. principle (Keep It Simple Stupid). This principle is encouraged by the use of CRC (Class Responsibility Collaborator) cards. These cards are divided into 3 sections: one section for the class's name, one

section for the class's responsibilities, one section for the class's collaborators. A class represents entities of similarity and can be named (e.g., employee). A responsibility is something the class knows or does (e.g., name, salary, address, eats lunch). A collaborator is another class, interacted with by the class being defined, and is necessary to fulfill the class's responsibilities (e.g., lunch). CRC cards are a simple tool that can be used to quickly create an object-oriented design. However, when a design has a high complexity level, a spike solution may be used in the design phase to refine the design. The phrase "spike solution" is Extreme Programming's proprietary name for prototype [15].

Extreme Programming's coding phase involves 3 key tasks [15]:

1. Creating unit tests before coding.
2. Coding to the unit tests.
3. Executing unit tests regularly.

The unit tests created serve as detailed requirements for the programmers during code implementation. Generally speaking, programmers will execute multiple unit tests against new or refactored code on a daily basis. When a programmer's code successfully passes its associated unit test, the programmer moves on to the next task. Additionally, to further ensure high quality software, the Extreme Programming model recommends "pair programing". Pair programing is a software programing technique that requires two programmers to simultaneously work on the same code at the same time from a single work station. In practice, one programmer types the code while the other programmer monitors and makes recommendations. Also, the programmers are required to switch roles routinely; each programmer performs each role 50% of the time.

Although unit tests are routinely executed during the coding phase, Extreme Programming dedicates a phase strictly to testing. During the testing phase, unit tests, integration tests, system tests, and acceptance tests are all possibilities. However, typical implementations of the Extreme Programming model focus the majority of the testing phase on acceptance testing

performed by the customer. The customer performs tests against the software to determine whether the requirements were sufficiently implemented in the new iteration. If the customer's acceptance testing uncovers issues, feedback is provided to the development team immediately, enabling the development team to address the issues as quickly as possible [22, 30].

All in all, the Extreme Programming model embraces these fundamental principles and practices: customer involvement, incremental delivery, people over process, embrace change, maintain simplicity, incremental planning, small releases, simple design, test-first development, refactoring, pair programming, collective ownership, continuous integration, sustainable pace, on-site customer [15, 16, 30].

A.8 Classic Active Server Page Technology

Classic Active Server Page (ASP) technology was Microsoft's first server-side scripting engine technology for dynamic generation of Web pages. Software applications built with classic ASP technology are often frighteningly difficult to maintain because their source code is typically a jumbled mess (i.e., spaghetti code). Spaghetti code is a derogatory term for source code having an overly complex and/or tangled control flow structure. Additionally, many classic ASP code modules are a mish-mash (i.e., a confused mixture) of programming languages. For example, a typical classic ASP code module may contain:

- Client-side code (JavaScript)
- Server-side code (VB Script)
- CSS (Cascading Style Sheets for style definition of Graphical User Interface elements)
- HTML

The various types of components found in classic ASP code modules are often highly dependent on each other. That is, they are contingent on or determined by each other. This high level of dependency between components results in source code that is highly brittle and difficult

to maintain. A seemingly simple change to one small piece of the source code is often infeasible because that one small piece of source code can directly impact many dependent components. Therefore, the number of man-hours required to implement a change in a classic ASP code module is often greatly amplified because changes must be implemented throughout the source code to counteract or support a change made in another area. While classic ASP code modules of a limited scope can be quite powerful and easy to maintain, they quickly become unmanageable once they exceed a certain size/complexity threshold [23].

APPENDIX B: SERVICE CENTRAL'S CLIENT-SIDE JAVASCRIPT OBJECT CONSTRUCTOR

```
function Service(strURI, objJSON, strJSON, intVersion){
  //Properties
  this.URI = strURI;
  this.JSON = objJSON;
  this.JSONString = strJSON;
  this.Version = intVersion;
  this.Error = "";

  //Methods

  //get
  this.get = function (){
    var http = new HTTPRequest();

    var getData = "JSON=" + encodeURIComponent( this.JSONString )

    http.open( "GET", this.URI, false );
    http.setRequestHeader("Content-type", "application/json");
    http.setRequestHeader("Content-length", getData.length);
    http.setRequestHeader("Connection", "close");
    http.send( getData );

    var strResponseMsg = http.responseText;

    try{
      this.JSON = eval( '(' + strResponseMsg + ')' );
      this.JSONString = strResponseMsg;
    }
    catch(err){
      this.Error = {"Msg" : err, "ResponseText" : strResponseMsg };
    }
  }

  //post
  this.post = function (){
    var http = new HTTPRequest();

    var postData = "JSON=" + encodeURIComponent( this.JSONString );

    http.open( "POST", this.URI, false );
    http.setRequestHeader("Content-type", "application/json");
    http.setRequestHeader("Content-length", postData.length);
    http.setRequestHeader("Connection", "close");
    http.send( postData );

    var strResponseMsg = http.responseText;

    try{
      this.JSON = eval( '(' + strResponseMsg + ')' );
      this.JSONString = strResponseMsg;
    }
    catch(err){
```

```
            this.Error = {"Msg" : err, "ResponseText" : strResponseMsg };
        }

    }

    //put
    this.put = function (){
        var http = new HTTPRequest();

        var putData = "JSON=" + encodeURIComponent( this.JSONString );

        http.open( "PUT", this.URI, false );
        http.setRequestHeader("Content-type", "application/json");
        http.setRequestHeader("Content-length", putData.length);
        http.setRequestHeader("Connection", "close");
        http.send( putData );

        var strResponseMsg = http.responseText;

        try{
            this.JSON = eval( '(' + strResponseMsg + ')' );
            this.JSONString = strResponseMsg;
        }
        catch(err){
            this.Error = {"Msg" : err, "ResponseText" : strResponseMsg };
        }
    }

    //delete
    this.delete = function (){
        var http = new HTTPRequest();

        http.open( "DELETE", this.URI, false );
        http.setRequestHeader("Content-type", "application/json");
        http.setRequestHeader("Connection", "close");
        http.send();

        var strResponseMsg = http.responseText;

        try{
            this.JSON = eval( '(' + strResponseMsg + ')' );
            this.JSONString = strResponseMsg;
        }
        catch(err){
            this.Error = {"Msg" : err, "ResponseText" : strResponseMsg };
        }
    }
}
```

APPENDIX C: SERVICE CENTRAL'S SERVER-SIDE VBSCRIPT URI CONTROLLER

```
<%
option explicit

Server.ScriptTimeout = 1800     ' Set Script timeout to 30 minutes

'============================
'     Variables
'============================

'-- Common

Dim strMySysDBName  ' System ODBC Datasource Name
Dim strMyAppDBName  ' Application ODBC Datasource Name
Dim strMyAdmDBName  ' Administration ODBC Datasource Name

Dim objConnSys      ' System Database Connection
Dim objConnApp      ' Application Database Connection

Dim objCommand      ' Database Command
Dim objRS           ' Result Set
Dim strSQL          ' SQL Statement
Dim intNoOfRecords  ' Number of records affected

Dim strSysId
Dim strLangId
Dim bFirstTime

%>

 <!-- #INCLUDE FILE="dbname.inc" -->
 <!-- #INCLUDE FILE="common.inc" -->
 <!-- METADATA TYPE="typelib" FILE="C:\Program Files\Common Files\System\ado\msado15.dll" -->

<%

'============================================
' Get Passed Data
'============================================

strSysId  = Request.Form( "SysId" )

If SysIsEmpty( strSysId ) Then

 strSysId = Request.Querystring( "SysId" )

End If

strLangId = Request.QueryString( "LangId" )

If SysIsEmpty( strLangId ) Then

 strLangId = SysGetLangId()

End If
```

```
'============================================
' Connect to database
'============================================

Call SysConnectSystemDB()

SysSetSessionSysId( strSysId )

Session.Contents( "LangId" ) = strLangId

Call SysConnectApplicationDB()

Dim strRequestMethod
Dim astrURI

strRequestMethod = UCase( Request.ServerVariables("REQUEST_METHOD") )

astrURI = Split( LCase( CStr( Request.ServerVariables("HTTP_X_REWRITE_URL") ) ), "?", 2)

Dim astrURIBase
Dim intURIBaseLength

astrURIBase = Split( LCase( CStr( astrURI(0) ) ), "/")
intURIBaseLength = UBound( astrURIBase )

Dim i
Dim bBeyondSc
Dim bAtCollection
Dim strUriBaseElemType
Dim aCollections(100)

Dim c
c = 0

bBeyondSc     = False
bAtCollection = False

For i = 0 To intURIBaseLength

 If bBeyondSc And bAtCollection Then

  bAtCollection  = False
  '--strUriBaseElemType = "Collection: "

 Else

  bAtCollection  = True
  '--strUriBaseElemType = "Element: "

 End If

 If bBeyondSc And Len( CStr( astrURIBase(i) ) ) > 0 Then

  '--Response.Write( strUriBaseElemType & astrURIBase(i) & "<br>" )
  aCollections(c) = astrURIBase(i)
  c = c + 1
```

```
  End If

  If astrURIBase(i) = "sc" Then

    bBeyondSc     = True
    bAtCollection = True

  End If

Next

If strRequestMethod = "POST" Or strRequestMethod = "PUT" Then

  Function BytesToStr(bytes)

    If Request.TotalBytes >  0 Then

        Dim Stream
        Set Stream = Server.CreateObject("Adodb.Stream")
            Stream.Type = 1 'adTypeBinary
            Stream.Open
            Stream.Write bytes
            Stream.Position = 0
            Stream.Type = 2 'adTypeText
            Stream.Charset = "iso-8859-1"
            BytesToStr = Stream.ReadText
            Stream.Close
        Set Stream = Nothing

    Else

      BytesToStr = ""

    End If

  End Function

  '---Response.Write("<br/>Post Data:<br/>")

  Dim strRequestBody
  Dim astrRequestBody
  Dim intRequestBodyLength

  strRequestBody = BytesToStr(Request.BinaryRead(Request.TotalBytes))

  astrRequestBody = Split( LCase( CStr( strRequestBody ) ), "&")
  intRequestBodyLength = UBound( astrRequestBody )

  For i = 0 To intRequestBodyLength

    If Len( CStr( strRequestBody ) ) > 0 Then

      Dim astrRequestBodyElements

      astrRequestBodyElements = Split( LCase( CStr( astrRequestBody(i) ) ), "=")

      Response.Write( "RequestBody Key: " & astrRequestBodyElements(0) & "<br>" )
```

```
      If CLng( UBound( astrRequestBodyElements ) ) > CLng( 0 ) Then

        Response.Write( "RequestBody Value: " & unescape( astrRequestBodyElements(1) ) & "<br>" )

      End If

    End If

  Next

End If

Dim aProdRegReqsCollections(4)

aProdRegReqsCollections(0) = "REGS"
aProdRegReqsCollections(1) = "Integer"
aProdRegReqsCollections(2) = "NOTES"
aProdRegReqsCollections(3) = "Integer"

Dim intCollectionsLength

intCollectionsLength = UBound( aCollections )
bAtCollection        = True

For i = 0 To 3

  If Len( CStr( aCollections(i) ) ) > 0 Then

    If bAtCollection Then

      bAtCollection = False

      If UCASE( aCollections(i) ) = aProdRegReqsCollections(i) And i = c-1 Then

        If i = 0 Then

          RegistrationResource strRequestMethod, ""

        Else

          Response.Write( "{""JSON for Registration # " & aCollections(1) & "'s collection of notes""}<br/>" )

        End If

      End If

    Else

      bAtCollection = True

      If IsNumeric( aCollections(i) ) Then

        aCollections(i) = CInt( aCollections(i) )

      End If

      If UCase( CStr( TypeName( aCollections(i) ) ) ) = UCase( aProdRegReqsCollections(i) ) And i = c-1 Then
```

```
          If i = 1 Then

            RegistrationResource strRequestMethod, aCollections(i)

          Else

            Response.Write( "{""JSON for Registration # " & aCollections(1) & "'s note"":" & aCollections(i) & "}<br/>" )

          End If

        End If

      End If

    End If

  Next

  Sub RegistrationResource(strRequestMethod,lngRequestNo)

    Select Case strRequestMethod

       Case "GET"

      %>
       {
         "ProdRegReqData":
           [
             <%
             '============================================
             ' Build SQL Statement
             '============================================

             strSQL = "Select "
             strSQL = strSQL & " REQUESTNO "
             strSQL = strSQL & " ,SYSID "
             strSQL = strSQL & " ,STATUS "
             strSQL = strSQL & " ,STATUSTEXT "
             strSQL = strSQL & " ,CRTDATE "
             strSQL = strSQL & " ,STSDATE "
             strSQL = strSQL & " ,PURDATE "
             strSQL = strSQL & " ,PRODUCTID "
             strSQL = strSQL & " ,SERIALNO "
             strSQL = strSQL & " ,UNIQSERIAL "
             strSQL = strSQL & " ,METER "
             strSQL = strSQL & " ,USERID "
             strSQL = strSQL & " ,ROLEID "
             strSQL = strSQL & " ,RCONO "
             strSQL = strSQL & " ,RCUSTNO "
             strSQL = strSQL & " ,RCUSTTYPE "
             strSQL = strSQL & " ,RCUSTTYPEN "
             strSQL = strSQL & " ,REMAIL "
             strSQL = strSQL & " ,OWNNAME "
             strSQL = strSQL & " ,OWNADD1 "
             strSQL = strSQL & " ,OWNADD2 "
             strSQL = strSQL & " ,OWNADD3 "
             strSQL = strSQL & " ,OWNCITY "
```

```
strSQL = strSQL & " ,OWNSTATE "
strSQL = strSQL & " ,OWNZIP "
strSQL = strSQL & " ,OWNCTRYID "
strSQL = strSQL & " ,OWNCONTACT "
strSQL = strSQL & " ,OWNPHONE "
strSQL = strSQL & " ,OWNFAX "
strSQL = strSQL & " ,OWNEMAIL "
strSQL = strSQL & " ,CONSUMERNO "
strSQL = strSQL & " from REGSREQS where 1=1 "


If Not SysIsEmpty( lngRequestNo ) and SysIsNumeric( lngRequestNo ) Then

  strSQL = strSQL & " and REGSREQS.REQUESTNO=" & lngRequestNo

End If

  strSQL = strSQL & " order by REQUESTNO"

'==============================================
' Execute SQL
'==============================================

Set objRS = objConnApp.Execute( strSQL )

Dim lngProdRegReqRequestNo
Dim strProdRegReqSysId
Dim strProdRegReqStatus
Dim strProdRegReqStatusText
Dim strProdRegReqCrtDate
Dim strProdRegReqStsDate
Dim strProdRegReqPurDate
Dim strProdRegReqProductId
Dim strProdRegReqSerialNo
Dim strProdRegReqUniqSerial
Dim lngProdRegReqMeter
Dim strProdRegReqUserId
Dim intProdRegReqRCoNo
Dim intProdRegReqRCustNo
Dim strProdRegReqRCustType
Dim intProdRegReqRCustTypeN
Dim strProdRegReqREmail
Dim strProdRegReqOwnName
Dim strProdRegReqOwnAdd1
Dim strProdRegReqOwnAdd2
Dim strProdRegReqOwnAdd3
Dim strProdRegReqOwnCity
Dim strProdRegReqOwnState
Dim strProdRegReqOwnZip
Dim strProdRegReqOwnCtryId
Dim strProdRegReqOwnContact
Dim strProdRegReqOwnPhone
Dim strProdRegReqOwnFax
Dim strProdRegReqOwnEmail
Dim intProdRegReqConsumerNo
Dim strProdRegReqDescription

bFirstTime = true
```

```
Do While Not objRS.EOF

 lngProdRegReqRequestNo  = SysXMLReplaceSpecialChars( objRs( "REQUESTNO"  ) )
 strProdRegReqSysId      = SysXMLReplaceSpecialChars( objRs( "SYSID"      ) )
 strProdRegReqStatus     = SysXMLReplaceSpecialChars( objRs( "STATUS"     ) )
 strProdRegReqStatusText = SysXMLReplaceSpecialChars( objRs( "STATUSTEXT" ) )
 strProdRegReqCrtDate    = SysXMLReplaceSpecialChars( objRs( "CRTDATE"    ) )
 strProdRegReqStsDate    = SysXMLReplaceSpecialChars( objRs( "STSDATE"    ) )
 strProdRegReqPurDate    = SysXMLReplaceSpecialChars( objRs( "PURDATE"    ) )
 strProdRegReqProductId  = SysXMLReplaceSpecialChars( objRs( "PRODUCTID"  ) )
 strProdRegReqSerialNo   = SysXMLReplaceSpecialChars( objRs( "SERIALNO"   ) )
 strProdRegReqUniqSerial = SysXMLReplaceSpecialChars( objRs( "UNIQSERIAL" ) )
 lngProdRegReqMeter      = SysXMLReplaceSpecialChars( objRs( "METER"      ) )
 strProdRegReqUserId     = SysXMLReplaceSpecialChars( objRs( "USERID"     ) )
 intProdRegReqRCoNo      = SysXMLReplaceSpecialChars( objRs( "RCONO"      ) )
 intProdRegReqRCustNo    = SysXMLReplaceSpecialChars( objRs( "RCUSTNO"    ) )
 strProdRegReqRCustType  = SysXMLReplaceSpecialChars( objRs( "RCUSTTYPE"  ) )
 intProdRegReqRCustTypeN = SysXMLReplaceSpecialChars( objRs( "RCUSTTYPEN" ) )
 strProdRegReqREmail     = SysXMLReplaceSpecialChars( objRs( "REMAIL"     ) )
 strProdRegReqOwnName    = SysXMLReplaceSpecialChars( objRs( "OWNNAME"    ) )
 strProdRegReqOwnAdd1    = SysXMLReplaceSpecialChars( objRs( "OWNADD1"    ) )
 strProdRegReqOwnAdd2    = SysXMLReplaceSpecialChars( objRs( "OWNADD2"    ) )
 strProdRegReqOwnAdd3    = SysXMLReplaceSpecialChars( objRs( "OWNADD3"    ) )
 strProdRegReqOwnCity    = SysXMLReplaceSpecialChars( objRs( "OWNCITY"    ) )
 strProdRegReqOwnState   = SysXMLReplaceSpecialChars( objRs( "OWNSTATE"   ) )
 strProdRegReqOwnZip     = SysXMLReplaceSpecialChars( objRs( "OWNZIP"     ) )
 strProdRegReqOwnCtryId  = SysXMLReplaceSpecialChars( objRs( "OWNCTRYID"  ) )
 strProdRegReqOwnContact = SysXMLReplaceSpecialChars( objRs( "OWNCONTACT" ) )
 strProdRegReqOwnPhone   = SysXMLReplaceSpecialChars( objRs( "OWNPHONE"   ) )
 strProdRegReqOwnFax     = SysXMLReplaceSpecialChars( objRs( "OWNFAX"     ) )
 strProdRegReqOwnEmail   = SysXMLReplaceSpecialChars( objRs( "OWNEMAIL"   ) )
 intProdRegReqConsumerNo = SysXMLReplaceSpecialChars( objRs( "CONSUMERNO" ) )
 strProdRegReqDescription=SysXMLReplaceSpecialChars(SysGetMEMOField(5468, lngProdRegReqRequestNo,
0, 0, 0, 0, 0, 0, 0 ) )

 If Not bFirstTime Then

   Response.Write ","

 End If
 %>

 {
   "RequestNo"   : "<%= lngProdRegReqRequestNo   %>",
   "SysId"       : "<%= strProdRegReqSysId       %>",
   "Status"      : "<%= strProdRegReqStatus      %>",
   "StatusText"  : "<%= strProdRegReqStatusText  %>",
   "CrtDate"     : "<%= strProdRegReqCrtDate     %>",
   "StsDate"     : "<%= strProdRegReqStsDate     %>",
   "PurDate"     : "<%= strProdRegReqPurDate     %>",
   "ProductId"   : "<%= strProdRegReqProductId   %>",
   "SerialNo"    : "<%= strProdRegReqSerialNo    %>",
   "UniqSerial"  : "<%= strProdRegReqUniqSerial  %>",
   "Meter"       : "<%= lngProdRegReqMeter       %>",
   "UserId"      : "<%= strProdRegReqUserId      %>",
   "RCoNo"       : "<%= intProdRegReqRCoNo       %>",
   "RCustNo"     : "<%= intProdRegReqRCustNo     %>",
```

```
            "RCustType"   : "<%= strProdRegReqRCustType   %>",
            "RCustTypeN"  : "<%= intProdRegReqRCustTypeN  %>",
            "REmail"      : "<%= strProdRegReqREmail      %>",
            "OwnName"     : "<%= strProdRegReqOwnName     %>",
            "OwnAdd1"     : "<%= strProdRegReqOwnAdd1     %>",
            "OwnAdd2"     : "<%= strProdRegReqOwnAdd2     %>",
            "OwnAdd3"     : "<%= strProdRegReqOwnAdd3     %>",
            "OwnCity"     : "<%= strProdRegReqOwnCity     %>",
            "OwnState"    : "<%= strProdRegReqOwnState    %>",
            "OwnZip"      : "<%= strProdRegReqOwnZip      %>",
            "OwnCtryId"   : "<%= strProdRegReqOwnCtryId   %>",
            "OwnContact"  : "<%= strProdRegReqOwnContact  %>",
            "OwnPhone"    : "<%= strProdRegReqOwnPhone    %>",
            "OwnFax"      : "<%= strProdRegReqOwnFax      %>",
            "OwnEmail"    : "<%= strProdRegReqOwnEmail    %>",
            "ConsumerNo"  : "<%= intProdRegReqConsumerNo  %>",
            "Description" : "<%= strProdRegReqDescription %>"
           }

           <%
           bFirstTime = false

           objRS.MoveNext

          Loop

          objRS.Close
          objRS = ""

        %>
        ]

      }

     <%

     Case "POST"

       '-- Code here to create new product registration request

     Case "PUT"

       '-- Code here to update an existing product registration request

     Case "DELETE"

       '-- Code here to delete an existing product registration request

     Case else

   End Select

  End Sub
  %>
```

APPENDIX D: SERVICE CENTRAL'S COLLECTION LEVEL HTTP REQUEST

HTTP request headers for GET request of http://5.221.208.53/sc/regs?SysId=S1:

Request: GET /sc/regs?sysid=S1 HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)
Accept-Encoding: gzip, deflate
Host: 5.221.208.53
Connection: Keep-Alive

HTTP response headers for GET request of http://5.221.208.53/sc/regs?SysId=S1:

Response: HTTP/1.1 200 OK
Date: Tue, 06 Nov 2012 22:25:17 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Content-Length: 23987
Content-Type: text/html
Cache-contro: private

HTTP response body for GET request of http://5.221.208.53/sc/regs?SysId=S1:


```
{
  "ProdRegReqData":[
    {
      "RequestNo":"1",
      "SysId":"S1",
      "Status":"0",
      "StatusText":"",
      "CrtDate":"11/21/2007 11:24:12 AM",
      "StsDate":"11/21/2007 11:24:12 AM",
      "PurDate":"11/1/2007",
      "ProductId":"P-0001",
      "SerialNo":"1234567",
      "UniqSerial":"1234567",
      "Meter":"0",
      "UserId":"MICKEY",
      "RCoNo":"1",
      "RCustNo":"300",
      "RCustType":"1",
      "RCustTypeN":"4",
      "REmail":"mary.higgins@hotmail.com",
      "OwnName":"Joe Owner",
      "OwnAdd1":"123 Blueberry Lane",
      "OwnAdd2":"",
      "OwnAdd3":"",
      "OwnCity":"Lenoir",
      "OwnState":"NC",
      "OwnZip":"28645",
      "OwnCtryId":"USA",
      "OwnContact":"Frank Smith",
      "OwnPhone":"(111)222-3333",
```

```
      "OwnFax":"(111)222-3334",
      "OwnEmail":"frank.smith@anywhere.com",
      "ConsumerNo":"59",
      "Description":""
    },
    {
      "RequestNo":"4",
      "SysId":"S1",
      "Status":"0",
      "StatusText":"test",
      "CrtDate":"",
      "StsDate":"2/23/2012",
      "PurDate":"",
      "ProductId":"P-0001",
      "SerialNo":"1234567",
      "UniqSerial":"",
      "Meter":"0",
      "UserId":"DONALD",
      "RCoNo":"1",
      "RCustNo":"0",
      "RCustType":"1",
      "RCustTypeN":"0",
      "REmail":"",
      "OwnName":"sally",
      "OwnAdd1":"123",
      "OwnAdd2":"",
      "OwnAdd3":"",
      "OwnCity":"grand",
      "OwnState":"MI",
      "OwnZip":"49417",
      "OwnCtryId":"USA",
      "OwnContact":"joe",
      "OwnPhone":"1234567",
      "OwnFax":"",
      "OwnEmail":"Joe@test.com",
      "ConsumerNo":"78",
      "Description":""
    },
    {
      "RequestNo":"5",
      "SysId":"S1",
      "Status":"1",
      "StatusText":"",
      "CrtDate":"",
      "StsDate":"2/24/2012",
      "PurDate":"",
      "ProductId":"P-0004",
      "SerialNo":"777-888-999",
      "UniqSerial":"",
      "Meter":"0",
      "UserId":"GKNIGHT",
      "RCoNo":"1",
      "RCustNo":"300",
      "RCustType":"1",
      "RCustTypeN":"4",
      "REmail":"",
      "OwnName":"joe blow",
      "OwnAdd1":"12345",
```

```
    "OwnAdd2":"",
    "OwnAdd3":"",
    "OwnCity":"holland",
    "OwnState":"MI",
    "OwnZip":"49417",
    "OwnCtryId":"USA",
    "OwnContact":"joe",
    "OwnPhone":"123",
    "OwnFax":"",
    "OwnEmail":"",
    "ConsumerNo":"79",
    "Description":""
  },
  {
    "RequestNo":"6",
    "SysId":"S1",
    "Status":"1",
    "StatusText":"",
    "CrtDate":"5/10/2012",
    "StsDate":"5/16/2012",
    "PurDate":"5/10/2012",
    "ProductId":"1600980-005",
    "SerialNo":"393",
    "UniqSerial":"393",
    "Meter":"0",
    "UserId":"",
    "RCoNo":"1",
    "RCustNo":"400",
    "RCustType":"1",
    "RCustTypeN":"1",
    "REmail":"bestbuy@bestnet.com",
    "OwnName":"Tom Jones",
    "OwnAdd1":"13503 Toms Street",
    "OwnAdd2":"",
    "OwnAdd3":"",
    "OwnCity":"Grand Haven",
    "OwnState":"MI",
    "OwnZip":"49417",
    "OwnCtryId":"USA",
    "OwnContact":"Joe",
    "OwnPhone":"616 844 2121",
    "OwnFax":"",
    "OwnEmail":"crow@rmb.com",
    "ConsumerNo":"0",
    "Description":""
  },
  {
    "RequestNo":"10",
    "SysId":"S1",
    "Status":"1",
    "StatusText":"",
    "CrtDate":"5/16/2012",
    "StsDate":"5/18/2012",
    "PurDate":"5/18/2012",
    "ProductId":"101388",
    "SerialNo":"876",
    "UniqSerial":"876",
    "Meter":"0",
```

```
    "UserId":"",
    "RCoNo":"0",
    "RCustNo":"0",
    "RCustType":"",
    "RCustTypeN":"0",
    "REmail":"0",
    "OwnName":"Hartford Enterprises",
    "OwnAdd1":"1740 Celia Creek Rd.",
    "OwnAdd2":"",
    "OwnAdd3":"",
    "OwnCity":"New York",
    "OwnState":"NY",
    "OwnZip":"10019",
    "OwnCtryId":"USA",
    "OwnContact":"John Wilkins",
    "OwnPhone":"(111) 222-3333",
    "OwnFax":"",
    "OwnEmail":"JohnW@hartfordent.net",
    "ConsumerNo":"1",
    "Description":""
  }
```

APPENDIX E: SERVICE CENTRAL'S ELEMENT LEVEL HTTP REQUEST

HTTP request headers for GET request of http://5.221.208.53/sc/regs/10?SysId=S1:

Request: GET /sc/regs/10?sysid=S1 HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)
Accept-Encoding: gzip, deflate
Host: 5.221.208.53
Connection: Keep-Alive

HTTP response headers for GET request of http://5.221.208.53/sc/regs/10?SysId=S1:

Response: HTTP/1.1 200 OK
Date: Tue, 06 Nov 2012 22:08:26 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Content-Length: 1281
Content-Type: text/html
Cache-contro: private

HTTP response body for GET request of http://5.221.208.53/sc/regs/10?SysId=S1:


```
{
  "ProdRegReqData":[
    {
      "RequestNo":"10",
      "SysId":"S1",
      "Status":"1",
      "StatusText":"",
      "CrtDate":"5/16/2012",
      "StsDate":"5/18/2012",
      "PurDate":"5/18/2012",
      "ProductId":"101388",
      "SerialNo":"876",
      "UniqSerial":"876",
      "Meter":"0",
      "UserId":"",
      "RCoNo":"0",
      "RCustNo":"0",
      "RCustType":"",
      "RCustTypeN":"0",
      "REmail":"0",
      "OwnName":"Hartford Enterprises",
      "OwnAdd1":"1740 Celia Creek Rd.",
      "OwnAdd2":"",
      "OwnAdd3":"",
      "OwnCity":"New York",
      "OwnState":"NY",
      "OwnZip":"10019",
      "OwnCtryId":"USA",
      "OwnContact":"John Wilkins",
      "OwnPhone":"(111) 222-3333",
```

```
        "OwnFax":"",
        "OwnEmail":"JohnW@hartfordent.net",
        "ConsumerNo":"1",
        "Description":""
      }
    ]
  }
```

APPENDIX F: SERVICE CENTRAL'S MOBILE APP. SCREENSHOTS

Figures 35 – 42 are screenshots of Service Central's prototyped mobile application.

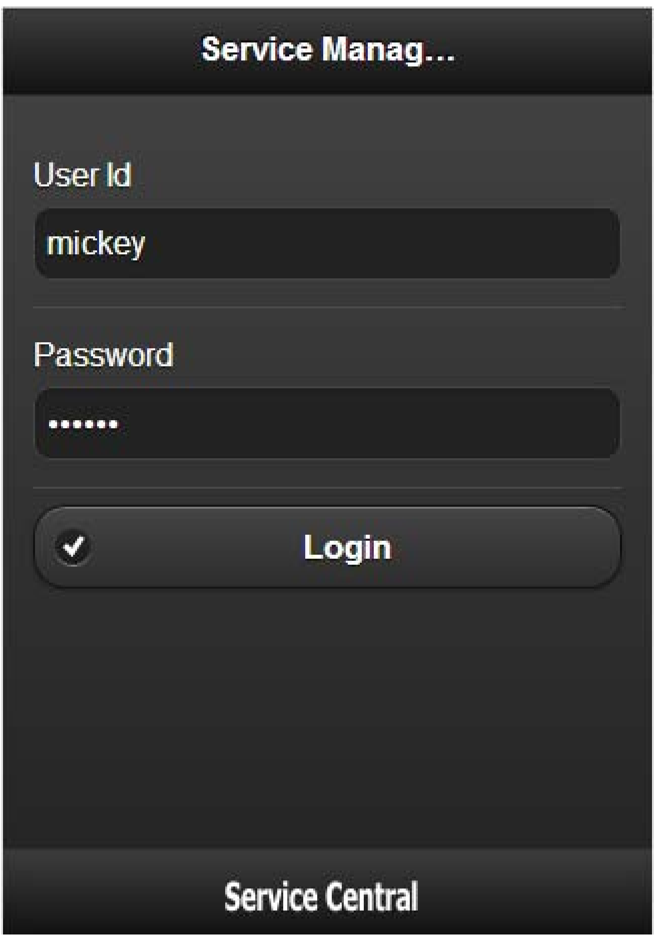

Figure 35: *User Login Screen*

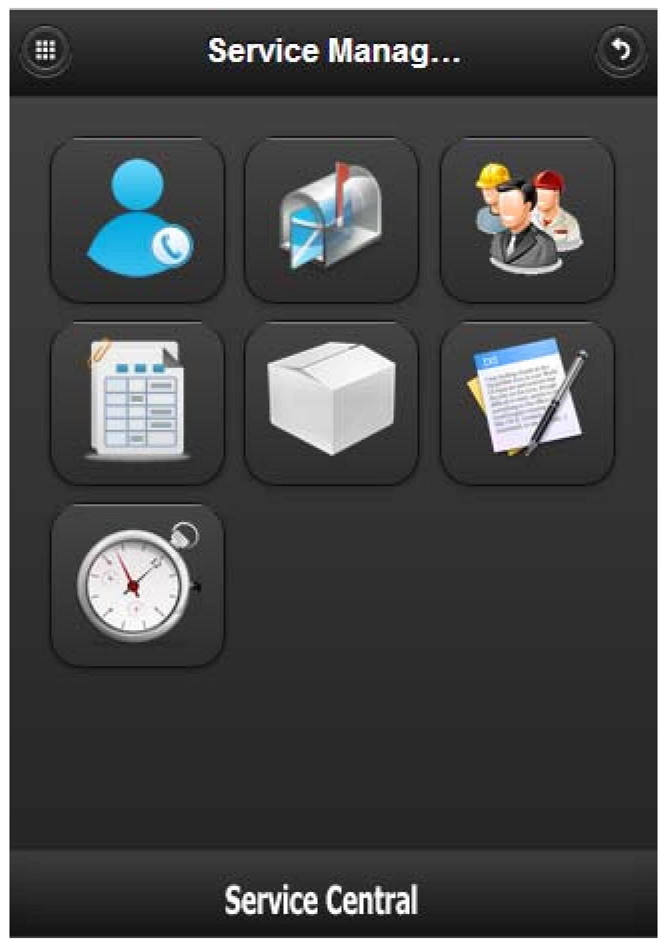

Figure 36: *Main Menu Screen*

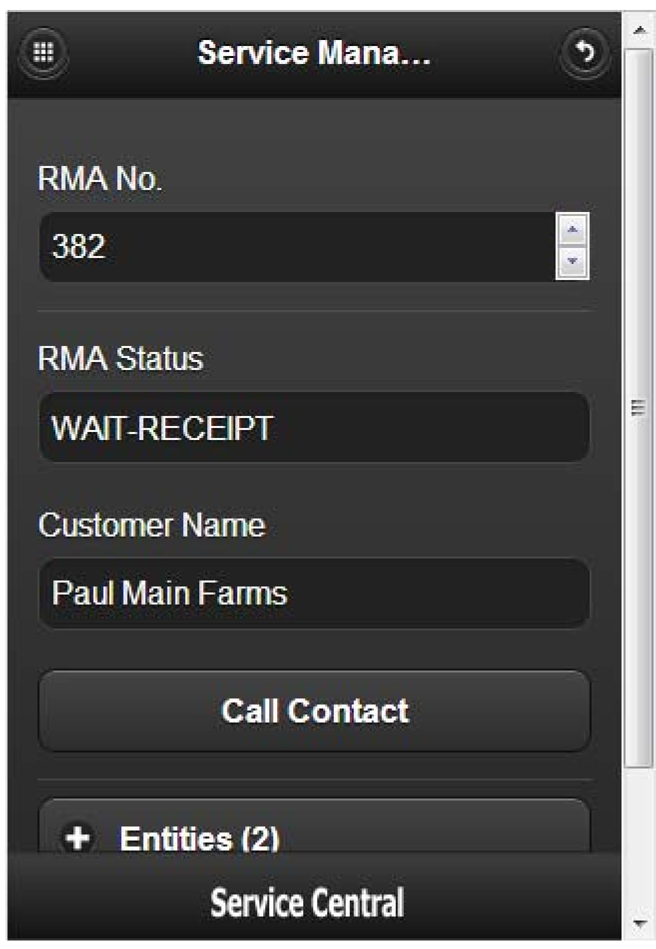

Figure 37: *RMA (Return Material Authorization) Screen – RMA# 382*

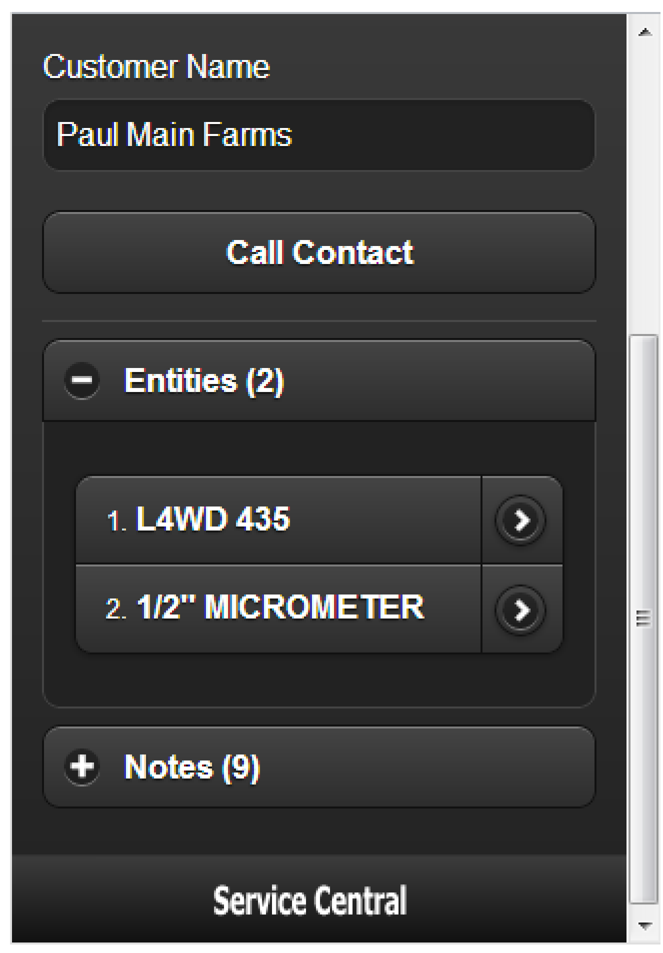

Figure 38: *RMA Screen, Entities Section – Two Entities for RMA# 382*

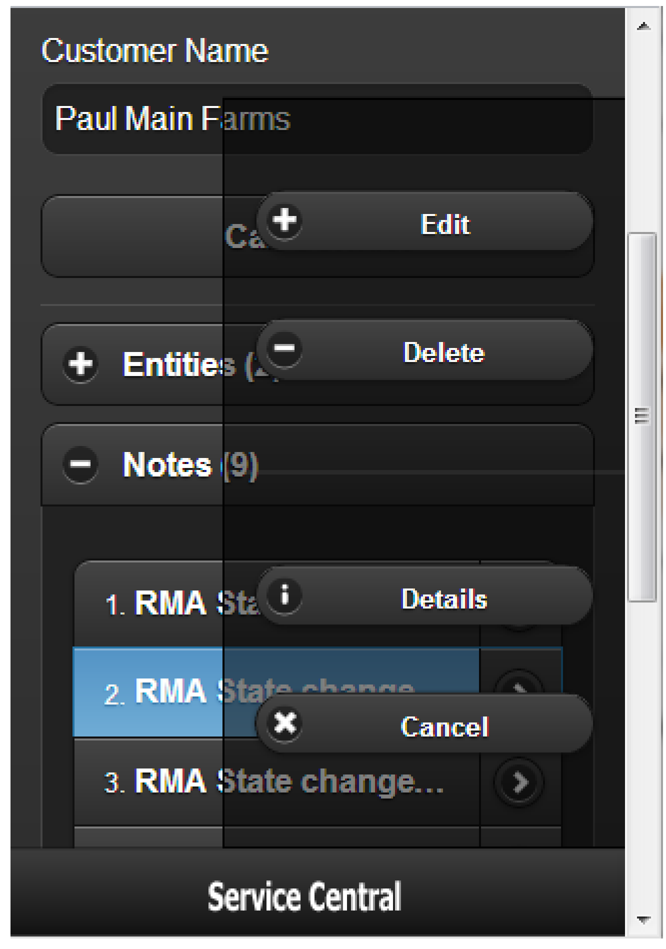

Figure 39: *RMA Notes Screen, Notes Section - Modal Dialog for Select Note*

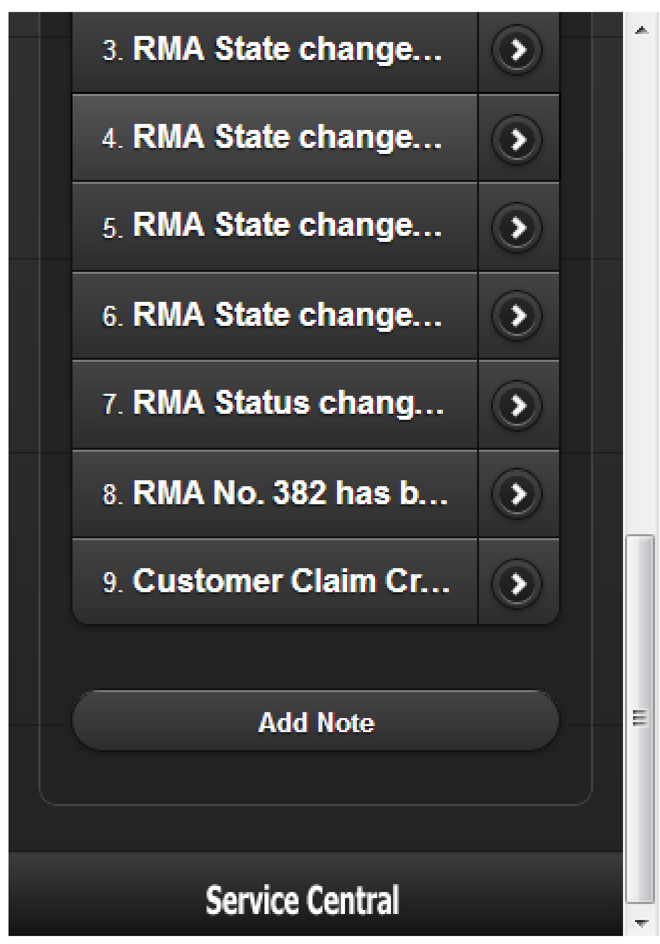

Figure 40: *RMA Note Screen, Notes Section – "Add Note" Button*

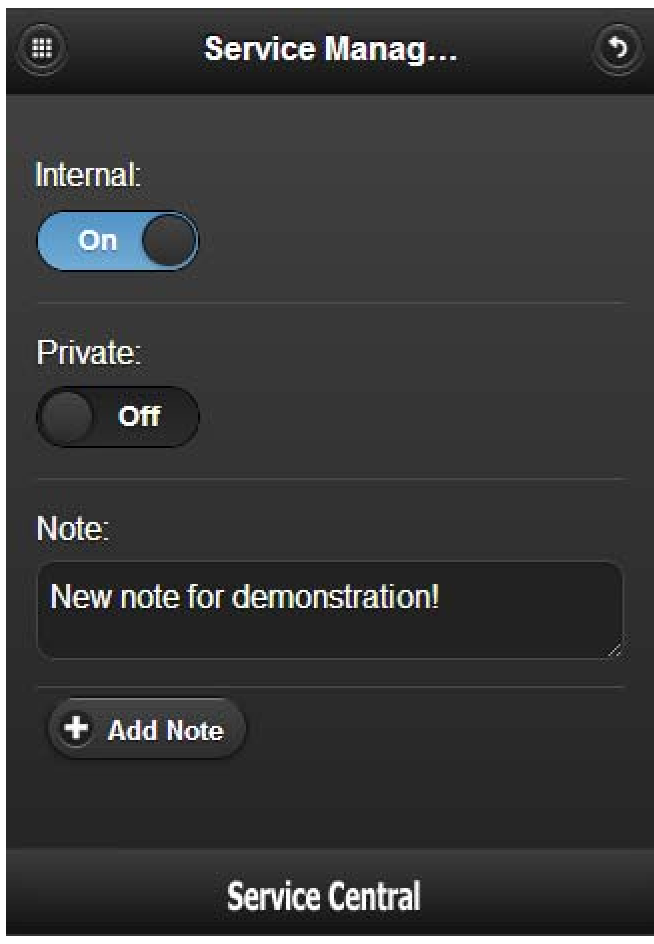

Figure 41: *RMA Add Note Screen*

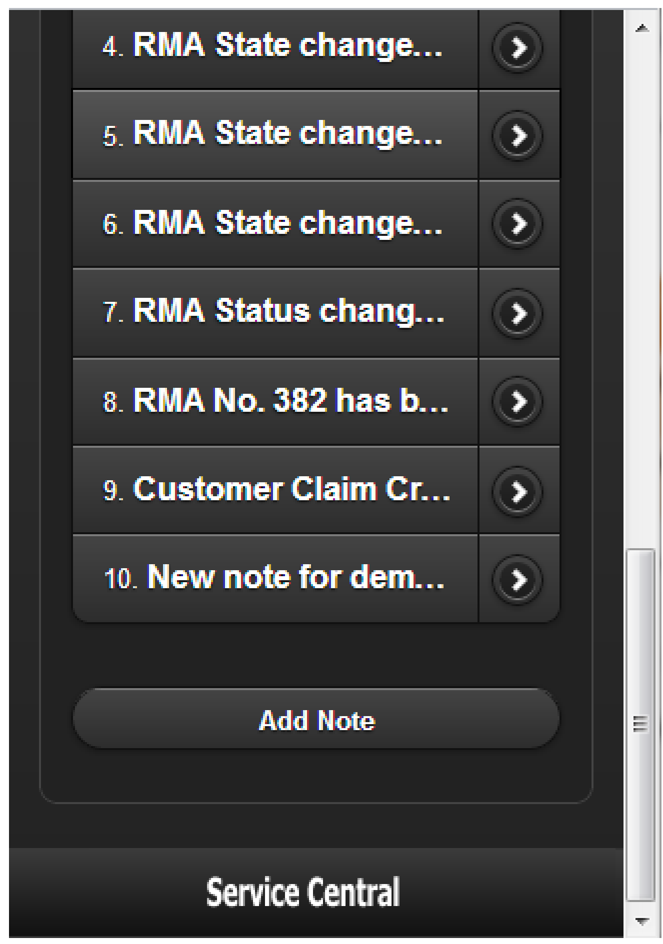

Figure 42: *RMA Note Screen, Notes Section – New Note Added*

APPENDIX G: SERVICE CENTRAL'S FRIENDLY WEB SERVICE INTERFACE

Friendly Web Service Interface (FWSI) is an architectural style that can be applied to Service Oriented Architectures (SOAs) [14]. It expands upon a subset of Dr. Roy Fielding's REST constraints and introduces the concept of Friendly Uniform Resource Identifiers (FURIs) [19].

The primary constraints of FWSI are:

1. Must use Hypertext Transfer Protocol (HTTP) to interact with Web service via four HTTP methods:
   a. GET (read resource)
   b. POST (create resource)
   c. PUT (update resource)
   d. DELETE (delete resource)
2. The HTTP response status from a Web service must always be status 200 (i.e., ok).
3. All HTTP request messages are JSON.
4. All HTTP response messages are JSON.
5. Resources are either collections made up of elements or a specific element within a collection.
6. Errors on the service-side must be caught, handled, and returned in the HTTP response body as a JSON formatted error message. Error messages should be as verbose and descriptive as possible.
7. FURI conform to the following syntax:
   a. http://{DomainName}/{ServiceDomain}/{CollectionResource}/{ElementResource}/{CollectionResource}/{ElementResource}?{QueryStringKey}={QueryStringValue}&{QueryStringKey}={QueryStringValue}
   b. Example FURI:
      http://5.221.208.53/sc/rmas/1/notes/2?userid=mantle&Password=1232ds3&Sysid=S1
      Mapping:
      {DomainName} = 5.221.208.53 (IP address in)
      {ServiceDomain} = sc
      {CollectionResource} = rmas
      {ElementResource} = 1
      {CollectionResource} = notes
      {ElementResource} = 2
      {QueryStringKey} = userid
      {QueryStringValue} = mantle
      {QueryStringKey} = Password
      {QueryStringValue} = 1232ds3
      {QueryStringKey} = Sysid

{QueryStringValue} = S1

c. Example of FURI with JSON template request:
http://5.221.208.53/sc/rmas/1/notes.template?userid=mantle&Password=1232ds3&Sysid=S1

The character string ".template" can be appended to the last collection resource (e.g., {CollectionResource}.template) of an HTTP GET type request. This will provide a JSON template of an element within the collection.

As illustrated in figure 43, FWSIs are realized across the *client* and *service façade* layers. The *service façade* layer is the realization of FURIs, which enables the *client* (service consumer) to easily send JSON formatted HTTP messages of the GET, POST, PUT, and DELETE types and expect to receive HTTP responses that allows contain JSON formatted data in the response body.

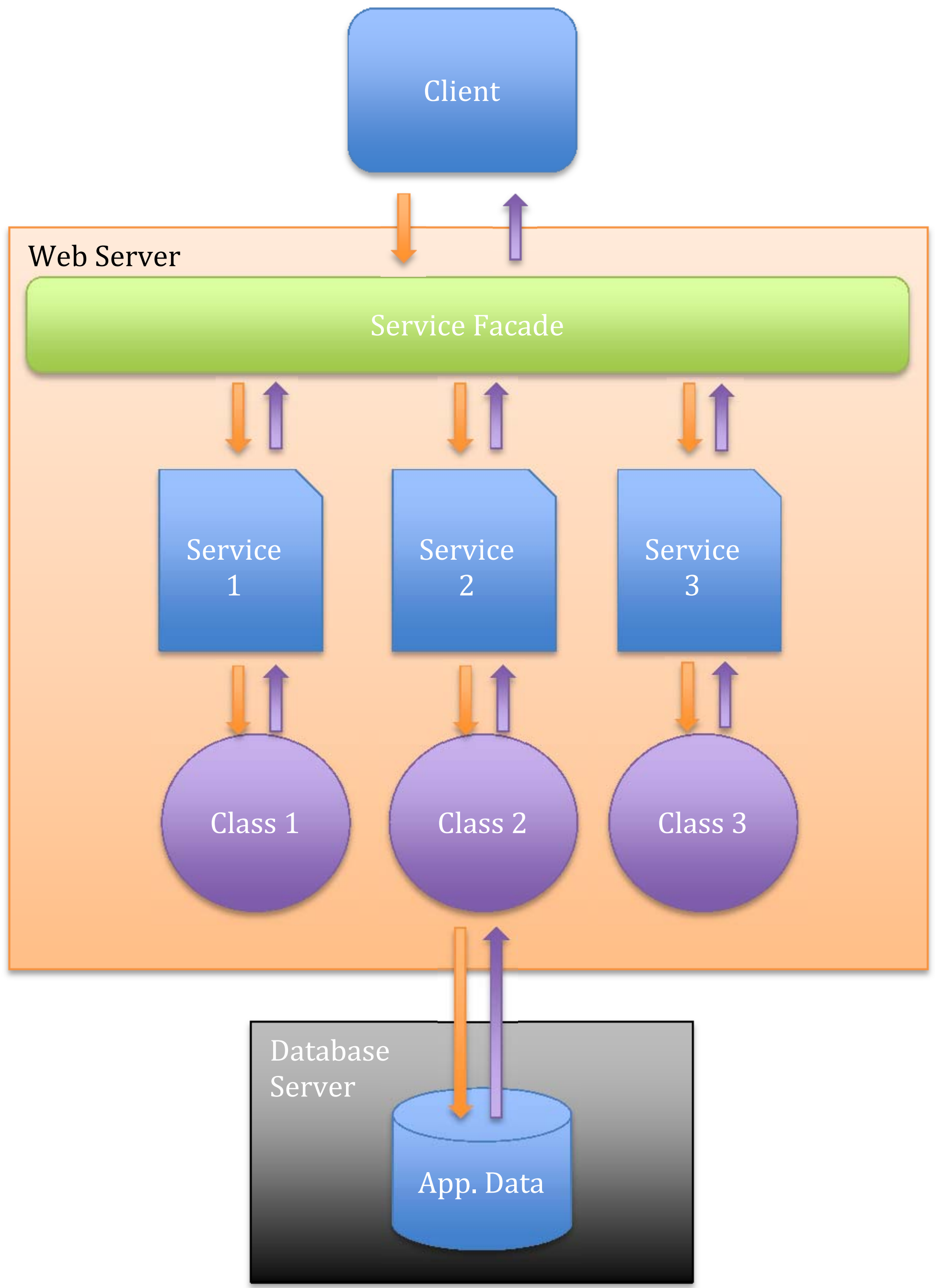

Figure 43: *FWSI Architecture*

APPENDIX H: SERVICE CENTRAL'S URI COLLECTIONS HIERARCHY

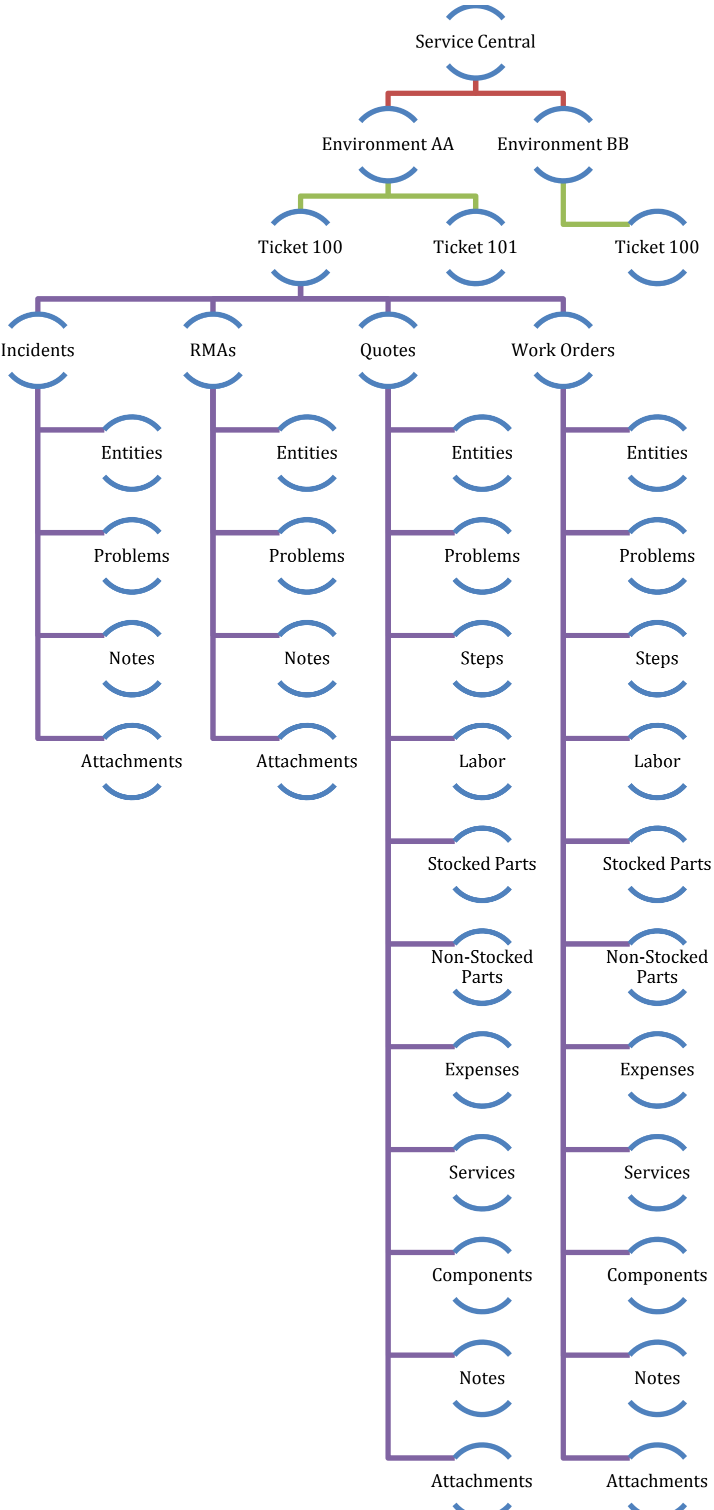

Figure 44: *URI Hierarchy*