



A COMPREHENSIVE EVALUATION FRAMEWORK FOR SYSTEM MODERNIZATION: A Case Study Using Data Services

Meredith Barnes

Supervisor: Prof. Charmain Cilliers



December 2010

Submitted in partial fulfilment of the requirements for the degree of Magister Scientiae in the
Faculty of Science at the Nelson Mandela Metropolitan University

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Charmain Cilliers for her continued support, encouragement and valuable advice. I would also like to thank my family for their support through this endeavour as without them I surely would not have coped. To Jason, dad, Aunty Meg, Aunty Diane and Kade especially: thank you from the bottom of my heart. My life has been richly blessed with your love and support. Much appreciation for my wonderful friends, Natalie, Philippa, Christiaan and Clayton must be expressed. Each of you has counselled me wisely and I am very grateful to call each of you a friend. Also, many thanks are due to Danie Venter for advising and assisting me with my statistical experiments. Lastly, the strength and focus that I maintained to complete this research was truly a gift from God and to Him I am eternally grateful. This scripture guided my way:

“Trust in the Lord with all your heart, and do not lean on your own understanding. In all your ways acknowledge him, and he will make straight your paths.” Proverbs 3:5-6.

Abstract

Modernization is a solution to migrate cumbersome existing systems to a new architecture for improved longevity of business processes. Three modernization approaches exist. White-box and black-box modernization are distinct from one another. Grey-box modernization is a hybrid of the white-box and black-box approaches. Modernization can be utilised to create data services for a Service Oriented Architecture. Since it is unclear which modernization approach is more suitable for the development of data services, a comprehensive evaluation framework is proposed to evaluate which of the white- or black-box approaches is more suitable.

The comprehensive framework consists of three evaluation components. Firstly, developer effort to modernize existing code is measured by acknowledged software metrics. Secondly, the quality of the data services is measured against identified Quality of Service criteria for data services in particular. Thirdly, the effectiveness of the modernized data services is measured through usability evaluations. By inspection of the combination of application of each of the evaluation components, a recommended approach is identified for the modernization of data services.

The comprehensive framework was successfully employed to compare the white-box and black-box modernization approaches applied to a case study. Results indicated that had only a single evaluation component been used, inconclusive results of the more suitable approach may have been obtained. The findings of this research contribute a comprehensive evaluation framework which can be applied to compare modernization approaches and measure modernization success.

Table of Contents

CHAPTER 1: RESEARCH CONTEXT

1.1. BACKGROUND	1
1.2. PROBLEM STATEMENT	4
1.3. THESIS STATEMENT	5
1.4. RESEARCH OBJECTIVES	6
1.5. RESEARCH METHODOLOGY	6
1.6. SCOPE AND CONSTRAINTS.....	11
1.7. DISSERTATION STRUCTURE.....	12

CHAPTER 2: MODERNIZATION APPROACHES AND TECHNIQUES

2.1. INTRODUCTION	15
2.2. EVOLVING LEGACY SYSTEMS.....	15
2.3. WHITE-BOX MODERNIZATION	18
2.4. BLACK-BOX MODERNIZATION.....	22
2.5. MODERNIZATION TECHNIQUES	24
2.5.1. <i>User Interface Modernization</i>	24
2.5.2. <i>Data Modernization</i>	25
2.5.3. <i>Functional Modernization</i>	26
2.6. CONCLUSION.....	28

CHAPTER 3: SERVICE ORIENTED ARCHITECTURES

3.1. INTRODUCTION	30
3.2. SERVICE ORIENTED ARCHITECTURE AND THE WEB	31
3.3. SOAP SERVICES.....	32
3.4. REST SERVICES	36
3.5. DATA SERVICES.....	38
3.6. CONCLUSION.....	39

CHAPTER 4: CASE STUDY MODERNIZATION USING DATA SERVICES

4.1. INTRODUCTION	42
4.2. THE STUDENT ASSISTANT SYSTEM	43
4.3. WHITE BOX MODERNIZATION APPROACH.....	46
4.3.1. <i>Program Understanding</i>	47

4.3.2. Code Migration	51
4.3.3. Web Service Discovery.....	54
4.3.3.1. Data Service Architecture.....	55
4.3.3.2. Implementation of Data Services.....	57
4.4. BLACK BOX MODERNIZATION APPROACH	62
4.4.1. User Interface Modernization	62
4.4.2. Component Based Service Discovery.....	64
4.5. CONCLUSION.....	67

CHAPTER 5: COMPREHENSIVE EVALUATION FRAMEWORK

5.1. INTRODUCTION.....	69
5.2. COMPREHENSIVE EVALUATION FRAMEWORK.....	70
5.2.1. Software Metrics and Developer Effort.....	71
5.2.2. Performance Metrics and QoS	75
5.2.3. Effectiveness of Data Services	76
5.3. VALIDATION OF FRAMEWORK.....	78
5.4. CONCLUSION.....	81

CHAPTER 6: ANALYSIS OF RESULTS

6.1. INTRODUCTION.....	83
6.2. EXPERIMENTATION DESIGN.....	84
6.2.1. Developer Effort Evaluation	84
6.2.2. Quality of Service Evaluation.....	85
6.2.3. Effectiveness of Data Services Evaluation	86
6.3. RESULTS.....	89
6.3.1. Developer Effort Results.....	90
6.3.2. Quality of Service Results	93
6.3.3. Effectiveness of Data Services Results.....	96
6.4. CONCLUSION.....	103

CHAPTER 7: CONCLUSION

7.1. INTRODUCTION.....	106
7.2. ACHIEVEMENTS	107
7.3. CONTRIBUTIONS	110
7.3.1. Theoretical Contribution	110
7.3.2. Practical Contribution.....	111
7.3.2.1. Application of Modernization Approaches	112

7.3.2.2. Application of the Evaluation Framework	113
7.4. LIMITATIONS	116
7.5. FUTURE WORK	118
7.6. SUMMARY	119

REFERENCES

120

APPENDICES

124

List of Figures

Figure 1.1. Adapted from Emerging Technologies Hype Cycle (Fenn <i>et al</i> , 2009)	1
Figure 1.2. Decomposition of Hypotheses	9
Figure 1.3. Chapter Flow of Dissertation	12
Figure 2.1. Adapted from “Taxonomy of Legacy Modernization Approaches”	17
Figure 2.2. Adapted from “Risk-Managed Modernization Approach”	19
Figure 2.3. Comparison of White-Box Modernization Approaches	21
Figure 2.4. Adapted from “The architecture of the wrapper”	23
Figure 3.1. Adapted from “A Sample SOA Environment”	33
Figure 3.2. Adapted from “SMART Inputs and Outputs”	35
Figure 4.1. Demi System Interface	43
Figure 4.2. Demi System Electronic Application Form	44
Figure 4.3. White-Box Modernization Phases	46
Figure 4.4. Class Dependency Diagram	47
Figure 4.5. Extract from Method Dependency Diagram	48
Figure 4.6. Subsystem Representation	50
Figure 4.7. Java Print Dialog	52
Figure 4.8. Adapted from “J2EE Architecture” (Kachru and Gehringer 2004)	56
Figure 4.9. Java Web Service Annotations	56
Figure 4.10. Excerpt from Check Student Method	57
Figure 4.11. Successful Application Dialog	58
Figure 4.12. Session Selection Interface	59
Figure 4.13. Print Session Interface	61
Figure 4.14. Adapted from “.NET Architecture” (Kachru and Gehringer 2004)	65
Figure 4.15. Excerpt from Session Class	66
Figure 5.1. Directed Graph of Cyclomatic Complexity	73
Figure 6.1. WMC Comparison for White-Box and Black-Box Modernization	90
Figure 6.2. Comparative Number of Methods of Client Classes	91
Figure 6.3. DIT Comparison for White-Box and Black-Box Modernization	93
Figure 6.4. Success Rate Comparison over All Five Tasks	98
Figure 6.5. Comparison of SUS Scores per Participant	101
Figure 7.1. Effects of Experiments on Hypotheses	109

List of Tables

Table 1.1. Research Methods Related to Research Questions	7
Table 4.1. Legacy System Classes Suitable for Modernization	49
Table 4.2. Web Services Discovered during White-Box Modernization	55
Table 4.3. User Interactions with the Legacy System	63
Table 4.4. Web Services Discovered during Black-Box Modernization	64
Table 5.1. The Candidate Metrics	71
Table 5.2. Quality of Service Attributes.....	75
Table 6.1. Comparison of WMC and Number of Methods for White-Box Services	92
Table 6.2. CBO Comparison for White-Box and Black-Box Data Services.....	94
Table 6.3. Wilcoxin Matched Pairs Test on Performance.....	95
Table 6.4. Biographical Statistics of Participants.....	97
Table 6.5. Participants Familiarity with the Demi System.....	97
Table 6.6. Frequency Distribution of System Functionality	97
Table 6.7. Frequency Distribution of Errors across Both Systems	98
Table 6.8. Comparative Statistics for Task Completion Times.....	100
Table 6.9. Results Obtained from Comprehensive Evaluation Framework.....	104
Table 7.1. Comprehensive Evaluation Framework Metrics.....	111
Table 7.2. Detailed Results of Comparison using the Comprehensive Evaluation Framework.....	114

Chapter 1: Research Context

1.1. Background

Service Oriented Architecture (SOA) has been identified as an emerging technology (Fenn *et al.* 2009). Research into this technology is currently on the rise and will reach maturity in a few years (Figure 1.1). A need for current research in this field has risen to gain clarification on the technology.

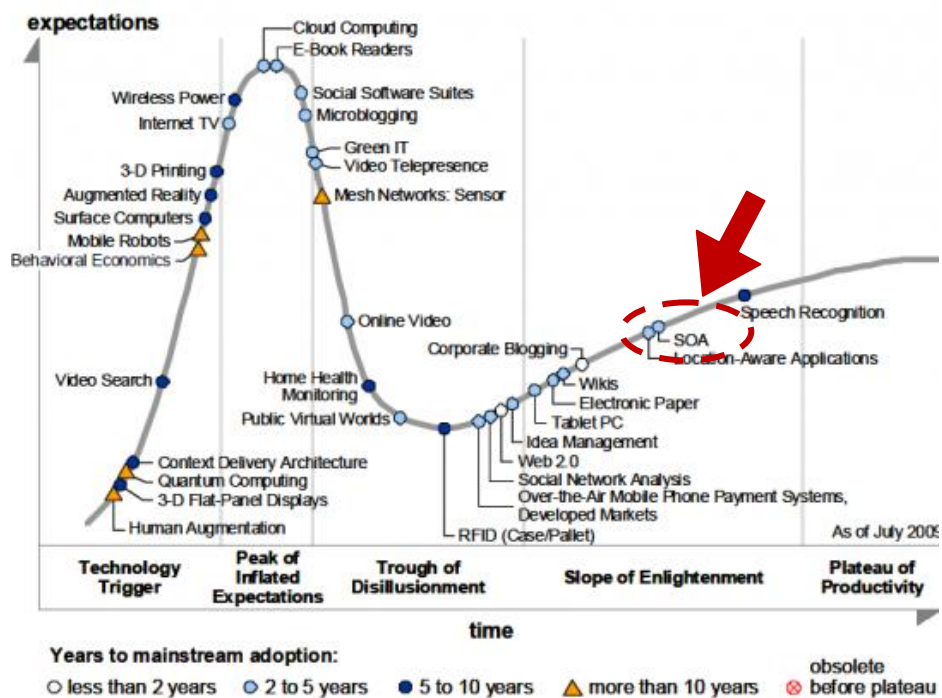


Figure 1.1. Adapted from Emerging Technologies Hype Cycle (Fenn *et al.* 2009)

The rising SOA awareness (Figure 1.1) implies that a need to modernize legacy systems into web services conforming to SOA requirements will benefit organizations. Legacy systems are often still in use in large organizations and are thus required to be modernized or migrated to adhere to the requirements of an SOA. Legacy systems are defined as software systems that start to oppose changes and updates during maintenance (Seacord *et al.* 2003).

This situation leads to a need to modernize or replace these systems as they contain knowledge that is of utmost importance to the organization that uses the software. Modernization is defined as changing the system in terms of structure, functionality or system attributes, yet preserving a large fraction of the system (Seacord *et al.* 2003). Modernization approaches can be used to migrate legacy systems towards service-oriented computing in an SOA (Canfora *et al.* 2006).

SOA has become an accepted paradigm for an architecture that supports the development of distributed systems that satisfy business objectives (Bianco *et al.* 2007). This approach allows the core functionality of the programs to be accessed via an interface (Liegl 2007). Liegl (2007) also states that successful interaction with this interface is achieved through XML (eXtended Markup Language) messages.

The provision of services that provide this core functionality is the responsibility of a Service Provider (Liegl 2007). Likewise, these programs, or services, are also required by other services or subcontractors, who are Service Consumers. Thus the SOA is a combination of service providers and services consumers providing and requesting services, all of whose information is stored, and made accessible to actors, in a Service Registry.

Services can be defined as a collection of software elements, each of which can execute a business process (Sanders *et al.* 2008). The definition of a service is not uniform, as more than one organization has defined the term.

A service should be platform independent and it should be able to interface with other services to share its self-reliant business process, which in turn should be loosely coupled with other services (Sanders *et al.* 2008). A software service can be described as a computational process with a well defined functional interface that is easily found and easily accessible (Zhang and Yang 2004). This service must process well defined XML document

requests that it has received in adherence to certain application protocols. Services are described as being small, loosely-coupled components of functionality that are provided, consumed and merged with one another over a network (Canfora 2004). Services also have the characteristic of being coarse-grained and self-contained software entities that communicate with other services and applications via asynchronous communication in the message-based format (Lewis *et al.* 2006).

The term “self-reliant” infers the service’s ability to maintain the same core functionality when not coupled with other services (Sanders *et al.* 2008). Loosely coupled defines the ability of services to interface, and access functionality, with one another without having knowledge of the core technical functionalities of the other services. Services are thus required to be reusable in terms of functionality in more than one application.

A data service, in particular, is described as having the core functionality of one specific business object, such as an order or an employee (Carey 2006). This data service is composed of service calls that can be used by an application to access and change any particular instance of a business object (Borkar *et al.* 2006). It thus follows that the structure of each data service describes the information stored for the particular business object type (Carey 2006).

The importance of quality metrics of services is stressed (Jeong *et al.* 2009). The functional attributes of services, which are deciding factors for service consumers when requesting services from service providers, need to be considered along with the non-functional Quality of Service (QoS) attributes (Jeong *et al.* 2009). Considering both the functional and non-functional attributes leads to the selection of high-quality services from the services discovered. Bianco *et al.* (2007) describe several quality attributes that need to be adhered to when generating services for a SOA. These attributes include performance and availability.

QoS metrics are essential in selecting the best possible execution plan with regard to budget and time constraints (Jeong *et al.* 2009). The most important quality attributes, identified as major quality attributes, include operation cost and accessibility.

Software metrics for Object-Oriented (OO) applications are useful for the analysis of effort required by the developer to create an application (Chidamber and Kemerer 1994). A suite of

software metrics is presented by Chidamber and Kemerer (1994) for measuring components contributing to the size and complexity of OO design. Services fall under the OO application category and thus, can be evaluated in terms of software metrics.

An empirical evaluation strategy with the purpose of comparing two legacy system modernization tools using a between subjects design is presented (Colosimo *et al.* 2008). This evaluation supported the determination of whether a new modernization tool was a feasible replacement for a modernization tool already in use by an organization used as a case study. The effectiveness of the data services developed from modernization is defined in terms of how well the services perform the tasks that they are supposed to perform (Sharp *et al.* 2007).

A need to merge these various evaluation approaches is evident, as each evaluation approach has been independently used in studies. The combination of evaluating developer effort, QoS and effectiveness of services generated from modernization is presented to complement one another in the determination of success of a modernization approach. The development of a comprehensive evaluation framework for the modernization of legacy systems to web services could provide an envisaged contribution to the body of knowledge surrounding system modernization.

1.2. Problem Statement

There are three techniques that exist for legacy system modernization, namely white-box, grey-box and black-box. These techniques could be applied to an existing legacy system to generate data services for an SOA. A comprehensive evaluation framework that combines metrics could be applied to determine the success of any single modernization approach. The application of this evaluation framework in a comparative way could determine which approach is more suitable for the generation of data services from system modernization.

A wrapping approach can migrate a legacy system to a web service in a SOA, based on the black-box modernization approach (Canfora *et al.* 2008). This black-box approach is presented as a successful method for migrating legacy systems to services in an SOA. Alternatively, a white-box based methodology can be used to migrate legacy systems to services using feature engineering and component-based software engineering for migrating legacy code into components (Mehta and Heineman 2002). An alternative approach, namely

grey-box modernization, can be implemented based on component-based development to generate services from legacy code more economically than black-, or white-box approaches (Zhang and Yang 2004).

All of these approaches are presented as successful solutions to a migration problem. There is, however, no comparison of these approaches conducted on the same domain and evaluated using a comprehensive evaluation strategy to determine which approach may be more suitable for data service generation. For the purposes of this research, a comparative study and extensive evaluation of white-box and black-box modernization will be conducted to provide a holistic comparison of the two approaches. The comparison of these two modernization approaches will form a proof of concept for the proposed holistic evaluation framework.

This lack of comparison leads to a need for an answer to the question: Will the application of a comprehensive evaluation framework help to determine which modernization approach is comparatively more appropriate for generating data services of high quality when both are applied to the same legacy system domain?

1.3. Thesis Statement

The application of a comprehensive evaluation framework to the comparative study of two modernization approaches, namely white-box and black-box, on a specific legacy system domain can clarify which approach is more suitable for the provision of data services.

The black-box and white-box legacy system modernization approaches will be implemented and applied to the same legacy system domain based on the case study used. The comparison of the two approaches comprises of three measurement tools. The three evaluation measures are:

- Quality of the services resulting from the legacy system modernization process will be measured against acknowledged Quality of Service (QoS) standards;
- A measure of the effort required by the developer to apply the modernization approach; and
- A measure of the effectiveness of the data services generated will be determined by the use of empirical evaluations with system users.

The different metrics evaluated during the comparison are distinct and carry the same weight. The results obtained from each of the approaches may be compared to determine the overall success of the modernization approach. This will provide a high-level overview of which modernization approach is more suitable for the development of data services for an SOA through system modernization.

1.4. Research Objectives

The primary research objective for this investigation is:

- To develop a comprehensive evaluation framework for the evaluation of a modernization approach

The secondary objectives to support the primary research objective are:

- To determine the QoS measures for data services in a SOA;
- To determine the effectiveness metrics for user evaluation of data services;
- To determine the metrics for developer's effort in implementing the modernization approaches;
- To apply white-box and black-box modernization approaches to a case study;
- To develop data services for an SOA through modernization of the case study; and
- To apply the comprehensive evaluation framework to the comparative analysis of the two modernization approaches to determine which approach is most suitable.

The research objectives will be achieved by devising specific research questions to be answered.

1.5. Research Methodology

The research questions that will guide the research during the study are listed along with the research methodologies related to each research question. The research questions 5, 6 and 7 relate directly to the main objective of this investigation (Table 1.1). The primary research objective (Section 1.4) is met by answering the questions regarding how to measure developer effort of the modernization approaches, quality of the data services generated and effectiveness of the data services. Answering research question 1 will achieve one of the secondary research objectives, namely to understand what modernization is and what approaches exist. Answering research question 2 will achieve the objective of understanding data services and the SOA. After understanding these concepts, the third research question (Table 1.1) will be answered in connection with the two research objectives that aim to apply

the modernization approaches to a cases study to develop data services. Research question 4 (Table 1.1) will be answered to achieve the secondary research objective of this investigation, namely the application of the proposed comprehensive evaluation framework to the comparison of modernization approaches.

	Research Question	Research Method
1	What are white-box and black-box modernization approaches and how do they compare with one another?	Literature Review
2	What are web services in general and data services in particular?	Literature Review
3	How do you develop data services by modernizing legacy systems using white-box and black-box modernization?	Literature Review (related work) Prototyping (development of services)
4	What measurements are used to compare the modernization approaches?	Literature Review Validation of comprehensive evaluation framework
5	How do you measure the effectiveness of the approaches in terms of the services developed?	Literature Review (related work) Empirical evaluations (services created) Comparative analysis of services generated regarding effectiveness metrics
6	How do you measure the developer's effort to generate the modernization approaches?	Literature Review (related work) Software engineering metrics analysis of modernization approach Comparative analysis of services generated regarding developer's effort metrics
7	How do you measure the QoS of the data services generated by the modernization approaches?	Literature Review (related work) Software engineering metrics analysis of modernization approach Comparative analysis of services generated regarding QoS metrics

Table 1.1. Research Methods Related to Research Questions

The research methods shown (Table 1.1) are relevant to this research as it is a comparative analysis of two legacy system modernization approaches on a single domain of a specific case study. It is advised that a comparative analysis be conducted on two items in a focused,

detailed approach (Hofstee 2006). A case study is a useful tool to measure and observe the effects of a study conducted in a current, real-life domain (Olivier 2004).

The use of a case study is beneficial to this research as it serves as a platform to gain a lot of information on a specific domain. Common information collection techniques that are used when performing research on a specific case study include participant observation and direct measurements (Olivier 2004). These data collection techniques will serve as a means to validate a specific hypothesis about the case study (Hofstee 2006).

A literature review will be conducted to cover the theory base for this research (Hofstee 2006). The literature review will cover legacy systems, the different legacy system modernization approaches as well as the SOA environment and data services. These topics cover the background theory that this research requires to build on. The literature study will also serve to reveal the QoS, developer effort and effectiveness metrics required to compare the services generated by the modernization tools developed.

A system development methodology involving incremental system development will need to be chosen when applying the two modernization approaches to the legacy system. This development will be applied directly to the case study, as the case study is the domain for the modernization approaches. The incremental iterative development chosen for the system development methodology will execute the development stage of the system life cycle (Whitten *et al.* 2004). The development of the modernized systems will be supplemented by a literature study to draw knowledge from extant systems related to the tools.

Finally, experiments for each of the three components of the evaluation framework will be conducted to provide a holistic evaluation of the modernization approaches and the data services generated by modernization. The purpose of these experiments is to test a specific hypothesis as to which modernization approach is more suitable in the case of the data services developed from the case study modernization (Hofstee 2006). The evaluations performed during the experiments will compare the two modernization approaches in terms of QoS requirement satisfaction, developer effort measures and effectiveness of the services that the modernization approaches generate. Not all criteria are quantitatively compared during experimentation.

The developer effort metrics are calculated and compared between the two approaches by inspection and not using Analysis of Variance (ANOVA). Only the performance of the modernized data services in terms of method latencies is compared for the QoS component of the evaluation framework using a comparative statistical test. Similarly, for the data service effectiveness component, the error rates, task completion times and SUS scores are compared for statistically significant difference. The main hypothesis for these two components of the framework is thus:

H_0 : Black-box and white-box modernized data services are not equivalent

H_1 : Black-box and white-box modernized data services are equivalent

Specific null hypotheses exist for the performance and effectiveness of the data services experiments designed for the comprehensive framework (Figure 1.2).

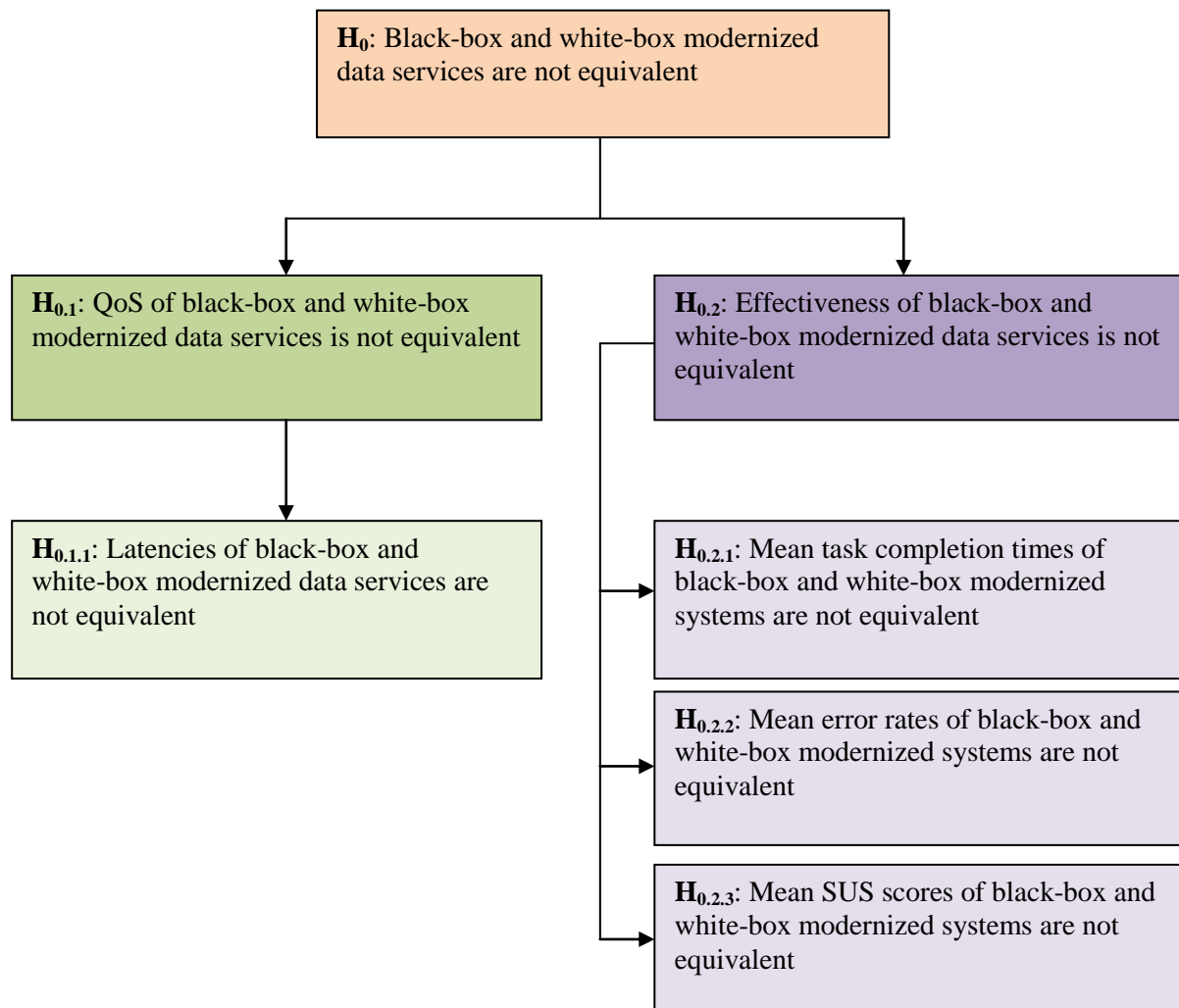


Figure 1.2. Decomposition of Hypotheses

The hypothesis for the data service performance is:

H_{0.1.1}: Mean service method latencies for black-box and white-box modernized data services are not equivalent

H_{1.1.1}: Mean service method latencies for black-box and white-box modernized data services are equivalent

A comparative analysis of the latencies will identify whether any significant difference exists in the mean latencies for a service method, thus rejecting the null hypothesis.

The experiments for the user evaluation consist of three hypotheses (Figure 1.2). The hypothesis for the comparison of task completion times is:

H_{0.2.1}: Mean task times for black-box and white-box modernized data services are not equivalent

H_{1.2.1}: Mean task times for black-box and white-box modernized data services are equivalent

A comparative analysis of the mean task times recorded for each task can identify if a significant difference exists between the mean task times, therefore rejecting the null hypothesis.

The hypothesis for the comparison of the error rates of the modernized systems is:

H_{0.2.2}: Mean error rates for black-box and white-box modernized systems are not equivalent

H_{1.2.2}: Mean error rates for black-box and white-box modernized systems are equivalent

A comparative analysis of this hypothesis could indicate if a significant difference exists in the number of errors encountered in the two modernized systems, thus rejecting the null hypothesis.

The third hypothesis to be evaluated for the comparison of self-reported results obtained from the System Usability Scale (SUS) is:

H_{0.2.3}: Mean SUS scores for black-box and white-box modernized systems are not equivalent

H_{1.2.3}: Mean SUS scores for black-box and white-box modernized systems are equivalent

A comparative analysis of the SUS scores will reveal if any significant difference is reported in the user's satisfaction with the two systems.

1.6. Scope and Constraints

The research conducted and the application of the identified modernization approaches will be related to the case study selected. The case study in question is the Nelson Mandela Metropolitan University (NMMU) Department of Computing Sciences Student Assistant system. This case study has been chosen as it provides the necessary functionality for the discovery of data services. The Student Assistant System has been in use for a few years in the department. Although the system has not been used for a very long time, it was developed in a language that is not suitable for future maintenance in the Department of Computing Sciences.

For the purposes of this study, two of the three modernization approaches will be compared to one another. The modernization approaches considered are white-box and black-box. This study excludes the grey-box modernization approach from the comparative study as it is a hybrid of the white-box and black-box approaches (Zhang and Yang 2004). A clearer distinction between the two modernization approaches can be made.

Thus, a comparison between two distinct and alternative approaches, namely white-box and black-box has been chosen. The black- and white-box approaches each have various techniques that exist to modernize existing system code and develop services for an SOA (Comella-Dorda *et al.* 2000). These different techniques will be presented and a decision will be made on which technique is most appropriate to use for each modernization approach.

A literature review on target architectures for the data services will be included in this study, as more than one type of service style exists. Two recognised ways of building a service are the SOAP-style service and the Representational State Transfer (REST) style service (Liu and Connelly 2008). Only one target service architecture will be chosen, supported by findings in literature based on which style of service is more acceptable for the development of data services in particular.

For the evaluation of the modernization approaches, a comprehensive evaluation framework is proposed. The three metrics considered for the comparison of the two modernization approaches will include:

- Adherence of the data services generated to QoS metrics;

- A measure of the developer’s effort required to modernize the legacy code; and
- A measure of the effectiveness of the data services generated.

The selected modernization approaches, as well as the services that they have generated, will be compared to one another based on these three evaluation criteria. This evaluation strategy provides a holistic evaluation, where the results from the measurements taken can be combined to achieve an overall evaluation outcome for each approach.

1.7. Dissertation Structure

The dissertation’s chapter flow and structure will now be explained (Figure 1.3). The chapter structures will include explanations of what is covered by the chapter, as well as what research questions the chapter answers.

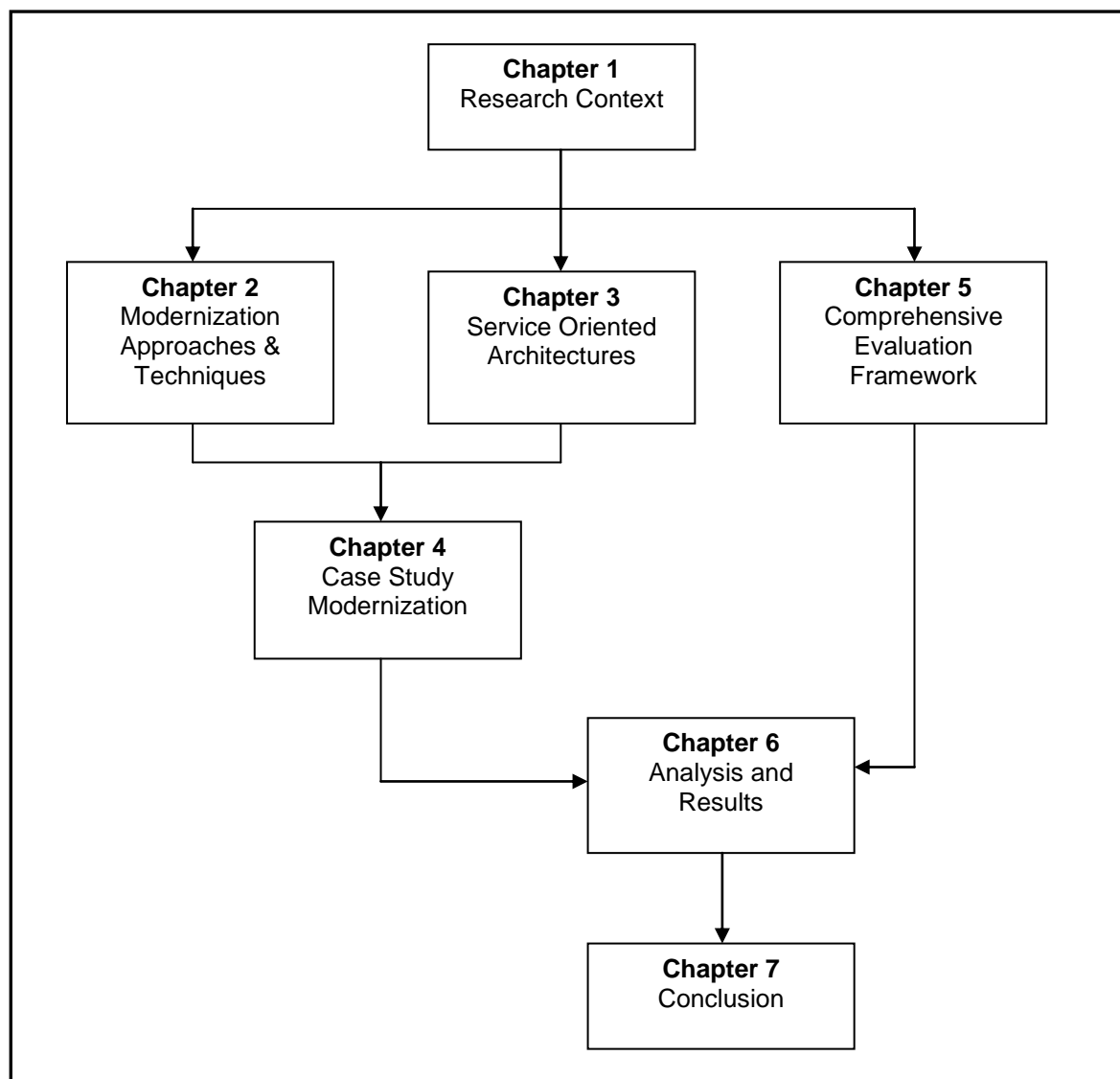


Figure 1.3. Chapter Flow of Dissertation

Chapter one introduces the research context for this study. A brief overview of the domain background is presented followed by the problem and thesis statements which outline the direction of the research. The relevant research questions and methodologies are described as well as the scope and constraints of the research.

Chapter two discusses related work in the areas of legacy systems and modernization. Here, the two different modernization approaches are presented as well as the need for modernization of legacy systems. Various black-box modernization techniques and white-box modernization techniques are discussed in this chapter to help address research question 1 (Table 1.1). The conclusion of this chapter discusses which techniques will be chosen for this research.

Chapter three presents the discussion on web services and SOA architecture in terms of various service architectures and SOA environments. This chapter aims to address research question 2 (Table 1.1) by examining the comparison of existing service architectures. In conclusion, a specific target architecture is chosen for the case study.

Chapter four elaborates on the implementation and execution of the chosen modernization approaches, thus proposing an answer to research question 3 (Table 1.1). The development of data services through white-box and black-box modernization efforts are presented respectively.

Chapter five presents the comprehensive evaluation framework consisting of three evaluation strategies, thus presenting a solution to research questions 5, 6 and 7 (Table 1.1). QoS metrics are presented, that services are required to comply with in an SOA environment as an answer to research question 7. The elaboration on software metrics and their relation to developer effort will also appear in this chapter, thereby discussing research question 6. The empirical evaluation procedure used to test the effectiveness of the data services is also explained to resolve research question 5. Results of the three-legged evaluation strategy are presented to conclude this chapter.

Chapter six presents the experimental design and results of the application of the proposed evaluation framework to a comparison of the two modernization approaches applied to the

case study. This chapter focuses on addressing research question 4 (Table 1.1), namely how to compare two modernization approaches.

Finally, chapter seven will present the context of the research and conclusions drawn from the research as well as any suggestions for future research. Achievements and contributions are presented in this final chapter as well.

Chapter 2: Modernization Approaches and Techniques

2.1. Introduction

To appreciate the need to modernize legacy systems to web services that conform to SOA requirements, it is first necessary to understand what legacy systems are (Section 2.2). The requirement for modernization leads to the presentation of the modernization approaches that will be considered by this research to carry out this modernization. The modernization of these legacy systems can be categorised into two distinct approaches, namely white-box (Section 2.3) and black-box (Section 2.4). Various techniques for the implementation of white-box and black-box approaches exist and are discussed (Section 2.5).

Legacy system modernization forms the foundation for this research, as the development of data services for an SOA is based on the modernization of the legacy system in the case study. Conclusions (Section 2.6) include which specific techniques of the white-box and black-box approaches will be implemented to perform the modernization of the legacy system to data services fit for an SOA. This conclusion aims to review the white-box and black-box modernization approaches in comparison to one another (Section 1.5).

2.2. Evolving Legacy Systems

Since legacy systems contain functionality that is of utmost importance for business longevity, it is necessary that they are evolved to integrate with more modern platforms and architectures, such as SOA (Erradi *et al.* 2006).

There are three categories for system evolution activities (Seacord *et al.* 2003). These three categories are:

- Maintenance;
- Replacement; and
- Modernization.

Continuously maintaining a system is sufficient for business needs for a period of time, but as the system ages, the business needs are not met by maintenance any longer (Seacord *et al.* 2003). At this point, it may be necessary to consider replacement of the system.

Replacement is a resource intensive procedure that involves the complete rebuild of the legacy system from first principles (Seacord *et al.* 2003). There is an appropriate time for this measure, namely when the legacy system cannot perform its required business processes any longer despite ongoing maintenance to the system. Another reason for replacement is if a modernization approach cannot be cost-justified. Different types of replacements exist, including an all at once, “big-bang” approach or an incremental replacement procedure.

Certain risks exist when attempting a system replacement (Seacord *et al.* 2003). These risks include the unfamiliarity of maintenance procedures of the new system for the IT personnel, as well as an extensive evaluation of the new system to determine whether the new system can perform as well as the legacy system. There is no guarantee that a new system can retain all of the functionality as the legacy system currently in use by the organization. This leads to a third option that conserves the system functionality, namely modernization.

Modernization covers a broader range of changes to the existing system than maintenance (Seacord *et al.* 2003). These changes could include restructuring of the system, improvement of system functionality or modification of system attributes. Modernization must, however, conserve a sizable portion of the existing legacy system (Comella-Dorda *et al.* 2000).

Legacy system modernization approaches can be split into two distinct categories (Erradi *et al.* 2006), namely:

- Legacy Integration and Service Enablement; and
- Legacy Transformation.

The two categories of legacy system modernization are further decomposed into non-invasive and invasive approaches (Figure 2.1). The non-invasive approaches tend to promote legacy integration and service enablement and the invasive approaches are shown to result in legacy transformation (Erradi *et al.* 2006).

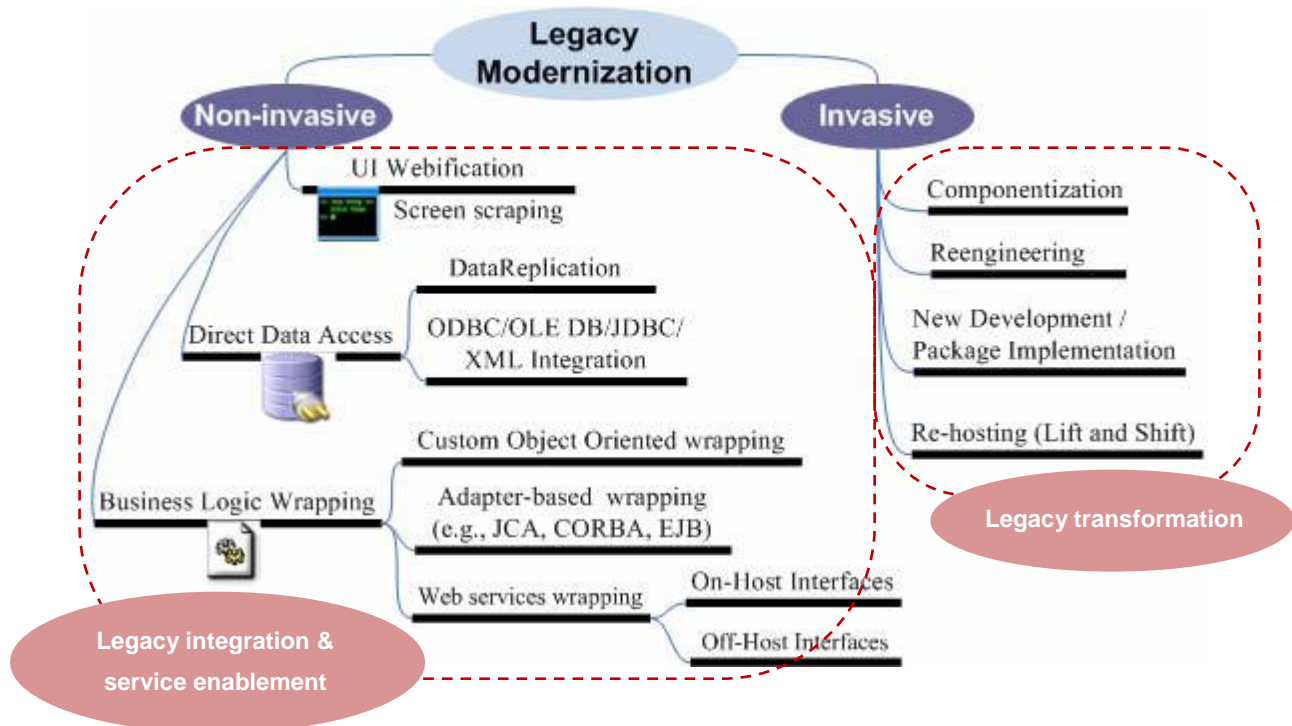


Figure 2.1. Adapted from “Taxonomy of Legacy Modernization Approaches” (Erradi *et al.* 2006)

Legacy integration uses non-invasive (Figure 2.1) wrapping of legacy systems to hide system complexity and emphasises modern interfaces to improve interoperability (Erradi *et al.* 2006). This process is used to help elongate the lifetime of legacy systems by exposing the integral functionality of these systems. Legacy wrapping reduces the cost of integration whilst requiring less immediate planning and design. This is, however, only a temporary short-term solution since it could actually complicate maintenance and management of the legacy system over a longer term period (Zhang and Yang 2004).

Legacy Transformation follows an invasive (Figure 2.1) reengineering approach to transform legacy systems and thus ease maintenance and extension (Erradi *et al.* 2006). This approach involves a detailed analysis of the existing legacy code and an understanding of the system functionality and data structure which leads to the extraction of data definitions and business

rules. Thus, the process of componentizing the functionality can be executed to produce more modular code.

Legacy system modernization can be classified by the degree of knowledge of system internals required to sustain the modernization approach (Comella-Dorda *et al.* 2000). Modernization that requires understanding of the functionality and structure of a legacy system is inherently white-box, and modernization as a result of analysis of interactions with the interface of the legacy system is black-box.

2.3. White-Box Modernization

White-box modernization requires knowledge of the legacy system code and functionality (Seacord *et al.* 2003). An initial reverse-engineering process is required to gain understanding of the legacy system internal structure and operation (Comella-Dorda *et al.* 2000). This process is called program understanding and involves modelling the domain, extracting information from the legacy code and creating representations of the system hierarchy (Seacord *et al.* 2003). Program understanding is a procedure used if the internals of the legacy system are unavailable. Program understanding may in some cases be a risky, labour intensive process.

After program understanding is completed, the restructuring of the code, or system, can begin to define a transformation of the system from one representation to another at the same level of abstraction (Comella-Dorda *et al.* 2000). The code restructuring process should maintain the legacy system's external procedures, or functionality and semantics (Seacord *et al.* 2003). Both Seacord *et al.* (2003) and Comella-Dorda *et al.* (2000) agree that this process typically alters quality attributes of the system including maintenance and performance.

Seacord *et al.* (2003) describe a case study which involves replacing a legacy system incrementally using various white-box modernization techniques known as Risk-Managed Modernization (RMM) (Figure 2.2). Knowledge of the legacy system code and functionality is required due to the choice of a white-box modernization approach. Once the code has been analysed and understood by using program understanding, the code restructuring step follows. The benefit of this code restructuring step is to improve quality of the system, whilst still maintaining the original business processes of the system.

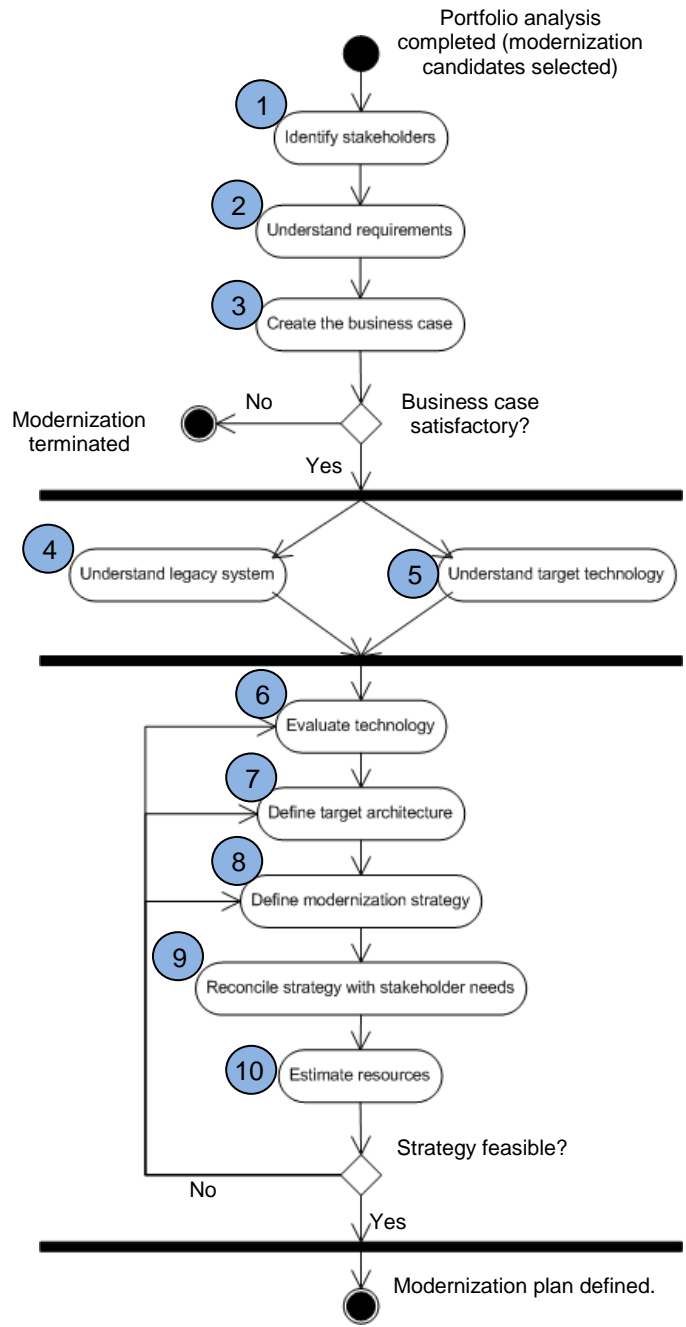


Figure 2.2. Adapted from “Risk-Managed Modernization Approach” (Seacord *et al.* 2003)

The RMM approach incorporates software engineering concepts to assist with the white-box modernization process as well as knowledge of the technologies defining the solution space of the information system (Seacord *et al.* 2003). The primary goal of this approach is to manage risk and migration successfully, thus leading to the development of a modernization plan that minimizes risk.

The case study revolves around the modernization of a 30 year old Retail Supply System (RSS) written mostly in COBOL with an infrastructure consisting of mainframe, mini-computer and desktop applications, both centralized and distributed (Seacord *et al.* 2003). Due to the size and complexity of the legacy system and the requirements of the company's Standard Retail Framework (SRF), an incremental componentization process was used to modernize the system.

The ovals in the diagram (Figure 2.2) are activities and arrows are transitions between activities (Seacord *et al.* 2003). The horizontal bars in the diagram (Figure 2.2) are "synchronization bars" that force the completion of previous activities before progressing onto the following activities. A process that begins with a legacy system modernization project first has an undefined plan and is then complete when the integrated plan is defined.

Alternatively, a simpler two step approach to white-box modernization is defined (Comella-Dorda *et al.* 2000). The first step is the process of initially reverse-engineering to achieve understanding of the legacy system's internal functionality (Figure 2.3). This reverse engineering step used is equivalent to program understanding. Components of the system and the relationships between these components are identified and a depiction of the system at a higher level of abstraction is formed. Legacy code modelling is a critical step in the program understanding phase (Zhang and Yang 2004). The modelling process is supplemented with all available resources, including source code, user interfaces and documentation.

The transformation process (Step 4 in Figure 2.2) is commenced by the identification of candidate classes and objects. The output is thus an object oriented model (Zhang and Yang 2004). In other words, the outputs from this program understanding process include the creation of representations of the system at a higher level of abstraction that aid in understanding of the system layout (Comella-Dorda *et al.* 2000).

After this program understanding process is complete, the next phase of white-box modernization is to include some form of system or code restructuring (Comella-Dorda *et al.* 2000). Program slicing, or code slicing, is a commonly used technique of software restructuring used to improve the quality attributes of the system, defined at the beginning of this section. This restructuring process is decided upon at step 8, "Define Modernization Strategy" in the RMM approach (Figure 2.2).

It is reported that the RMM approach is not a suitable approach for the modernization of a system towards web service discovery for a SOA (Chung *et al.* 2007). The RMM approach lacks detail on what specific software reengineering techniques should be applied to the system in order to modernize it. Furthermore, the approach does not incorporate the use of web services in modernization and the effects thereof.

Thus, another approach is presented which highlights four steps to achieve legacy code transformation (Errickson-Connor 2003). The four steps are as follows:

- Clean legacy code by removal of program anomalies;
- Restructure software by identifying business rules to extract reusable services;
- Transform code from extracted business rules into components; and
- Manage these reusable components in a software environment.

This approach can be related to the approach presented by Comella-Dorda *et al* (2000), as depicted (Figure 2.3).

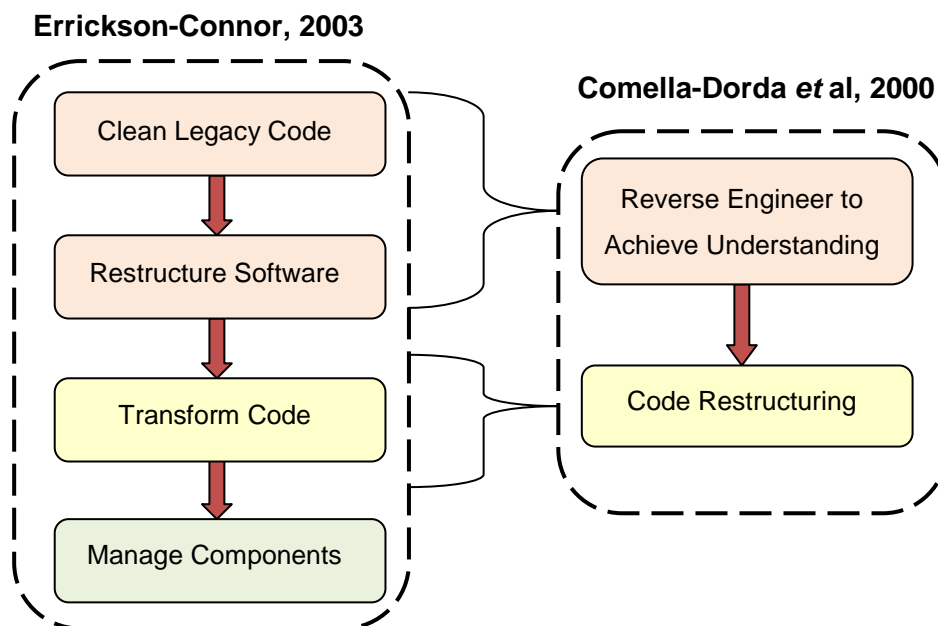


Figure 2.3. Comparison of White-Box Modernization Approaches

Program understanding is identified as a reverse-engineering process to analyse the legacy system in order to identify the system's modules and relationships (Chiang and Bayrak 2006). The high-level representation of the system can assist in the identification of anomalies in the legacy code as well as the identification of business rules (Figure 2.3).

Legacy code related to a specific business rule may be extracted by a technique known as program slicing (Chiang and Bayrak 2006). Software code is fragmented into slices, where each slice consists of a subset of the original program. These slices of business logic can then be transformed into components. Lastly, the components can be managed in a software environment by the identification of the target architecture. The process of legacy software modernization may be partially or fully automated depending on requirements of the system.

Alternatively to white-box modernization which considers the legacy system internal functionality and code, another modernization approach exists which ignores the internal complexities of the legacy system. This other modernization approach is thus black-box in nature.

2.4. Black-Box Modernization

Black-box modernization is concerned with the evaluation of the inputs and outputs of the legacy system during operation to gain knowledge of the interfaces (Seacord *et al.* 2003). It is reported that this approach is a less complex task than white-box modernization as it incorporates wrapping.

Wrapping involves encasing the legacy system in a software layer that masks the unnecessary intricacies of the old legacy system whilst generating a modern interface (Seacord *et al.* 2003). This approach is black-box since after the legacy interface is analysed the internals are disregarded (Zhang and Yang 2004). This solution is not always realistic and can sometimes require knowledge of some system internals (Seacord *et al.* 2003).

A black-box modernization approach is presented, based on wrapping (Canfora *et al.* 2006). The wrapper interfaces with the legacy system during execution of every possible interaction instance. Thus, the interaction between the user and the legacy system is mediated by the wrapper (Figure 2.4).

Data wrappers also provide a solution to integrity problems that are prevalent in legacy databases (Thiran *et al.* 2006). The addition of a data wrapper, described as a dedicated software component inserted between the legacy database and the modern application,

extends the lifetime of the legacy application. This is due to the advantage of the wrapper allowing integration of the legacy database into modern distributed systems.

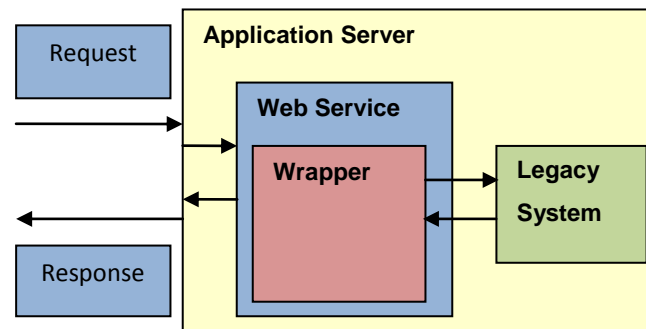


Figure 2.4. Adapted from “The architecture of the wrapper” (Canfora *et al.* 2006)

The wrapper created is required to decipher the request sent to the legacy system dynamically for each interaction, thus allowing the required operation to be performed (Figure 2.4). The creation of a wrapper can be achieved via observation of all possible interactions with the interface of the legacy system (Seacord *et al.* 2003). To obtain a dynamic wrapper however, a model of interaction between users and the legacy system can be created using Finite State Automata (FSA).

An FSA created to model user interaction with the legacy system forms the primary basis of the design of a legacy system wrapper (Canfora *et al.* 2006). To successfully have the wrapper communicate with the legacy system during user interaction for all use cases, users have to execute every possible interaction scenario so that they can be recorded for the FSA. Once this process is complete, screen templates are recorded to provide the correct output regarding each user interaction scenario.

There are many appealing advantages for incorporating wrappers into legacy applications (Thiran *et al.* 2006). Firstly, the legacy database is not altered in any form or functionality. Secondly, a wrapper promotes database heterogeneity by its provision of a common interface between applications and the database. Thirdly, the wrapper allows for further functionality to be added to existing legacy applications, such as statistics collection and performance visualization. Lastly, wrappers allow for an incremental modernization process of complex legacy systems.

It is suggested that another advantage of a black-box approach incorporating a wrapper is its reusability for exporting other use cases implemented by other systems (Canfora *et al.* 2006). On the other hand, the disadvantages to the black-box modernization approach are the manual recording of interaction scenarios and screen templates. Additionally, the task of analysing only the inputs and outputs of the legacy system during black-box modernization are not always sufficient (Comella-Dorda *et al.* 2000).

2.5. Modernization Techniques

Modernization can occur at different levels, namely the interface, data or functional (logic) level (Comella-Dorda *et al.* 2000). In relation to white-box and black-box modernization, some specific implementation techniques for each approach exist. The techniques are split into levels as follows:

- User Interface Modernization;
- Data Modernization; and
- Functional Modernization.

A white-box technique can be applied at some levels depending on the level and the amount of code analysis required. Alternatively, a black-box technique can be employed if less knowledge of the internal business logic and functionality is required.

2.5.1. User Interface Modernization

User interface modernization is the process of exposing aspects of the legacy system interface to a new interface which communicates with the legacy system to execute business processes (Stroulia *et al.* 2002). Common legacy system user interfaces are text-based requiring text input from the user (Seacord *et al.* 2003). Due to the user interface being the most visible part of a system, the modernization of the user interface improves usability and user satisfaction (Comella-Dorda *et al.* 2000).

A widely used black-box technique for modernizing interfaces is screen scraping. Screen scraping involves wrapping old, text-based interfaces with graphical user interfaces (GUIs) (Comella-Dorda *et al.* 2000). This new GUI can be PC-based or a lightweight HTML page running in a web browser. This is an easily extensible technique, since one new interface can be used to wrap many legacy systems. The new GUI communicates with the old interface using a specialised tool.

Commercial tools can often generate new screens automatically by mapping the old interfaces. The end user will experience a new, usable modern interface and they will interpret modernization of the legacy system as successful, whereas from a business perspective, the new system becomes inflexible and hard to maintain.

2.5.2. Data Modernization

Data modernization allows access to a host program's database through various data wrapping techniques (Sneed and Sneed 2003). Data wrapping allows access to the legacy system data via a different interface or protocol than what the data was designed for (Comella-Dorda *et al.* 2000). There are two types of data wrapping techniques:

- Database Gateway; and
- XML Integration.

The benefit of both of these data modernization techniques is to improve connectivity and allow for integration of the legacy data into modern architectures.

A *Database Gateway* enables access to legacy data via a specific software gateway that translates between two or more data access protocols (Comella-Dorda *et al.* 2000). Many vendor-specific access protocols are in use, but there are also industry standards such as the Open Database Connectivity (ODBC). This is Microsoft's interface for access to data in a heterogeneous environment of relational and non-relational database systems. The Java Database Connectivity (JDBC) is another standard data access interface released by Sun for database-independent connectivity between Java applications and a large range of databases.

A database gateway usually translates a vendor-specific access protocol into one of the industry standard protocols (Comella-Dorda *et al.* 2000). This gateway proves to be useful since modern applications normally support one or more of the industry standard protocols. This is where the improvement in connectivity occurs when applying database gateways to access legacy data. This also leads to more seamless integration of legacy systems with modern systems.

The *eXtensible Markup Language (XML)* is a widely accepted format of structuring documents and data on the internet (Comella-Dorda *et al.* 2000). XML exhibits characteristics such as simplicity and flexibility in terms of its text format. This flexibility

allows XML to be a powerful tool for business to business application integration. This integration is the automated exchange of information between systems from different organizations.

The main component in XML-based architectures is the XML server (Comella-Dorda *et al.* 2000). This server acts as the communication point between the organization's infrastructure and the rest of the world. The server communicates with the internal infrastructure; either the ERP system or the databases etc. The server then exchanges information with external organizations by exchanging XML messages. Most commercial XML servers support numerous communication protocols that enable cost-effective integration with most common legacy applications.

2.5.3. Functional Modernization

Functional modernization consists of various techniques designed to access business logic in the host system (Sneed and Sneed 2003). Three techniques for logic wrapping are presented (Comella-Dorda *et al.* 2000). The three techniques are:

- CGI Integration;
- Object-Oriented Wrapping; and
- Component Wrapping.

The *Common Gateway Interface (CGI)* is a standard for communicating with external applications with information servers, for example web servers (Comella-Dorda *et al.* 2000). If fast web access to existing assets is required, then legacy integration using CGI is preferable. A new GUI is created, but as opposed to the screen scraping technique where the old interface is wrapped, this technique involves the new GUI interacting directly with the business logic or data of the existing legacy system. This technique is far more flexible than screen scraping since the new GUI doesn't need to match the old interface; however, it has been shown that this procedure still does not satisfy maintenance issues.

Object-oriented wrapping involves the depiction of individual applications as objects, which seems to be a simple concept, but this is misleading (Comella-Dorda *et al.* 2000). Realistically, object-oriented wrapping involves many tasks including code analysis, decomposition and construction of the Object-Oriented (OO) model.

Clustering analysis is used for transferring procedural code into an object-oriented model for system understanding (Zhang and Yang 2004). Clustering analysis involves grouping numerous entities in a dataset into clusters with regard to their relationships and similarities. This analysis can only take place after the entities to be clustered have been defined as well as the similarities in their relationships. The clustering algorithm to be used must then be defined as well, dependent on the domain. This form of clustering analysis is applied during the program understanding stage of the white-box modernization approach.

OO systems can be designed and created in such a way that they closely portray the actual business processes that they model (Comella-Dorda *et al.* 2000). In addition to this, these systems are easier to comprehend due to the OO techniques used to create them, such as encapsulation and inheritance. The identification of objects in procedural legacy code is a result of these clustering techniques (Zhang and Yang 2004). Translating the semantics of procedural legacy system code to a hierarchical structure of an OO system can be a complex task (Comella-Dorda *et al.* 2000). Domain-specific knowledge of the legacy system structure is necessary to create meaningful objects.

Component wrapping bears a resemblance to OO wrapping; however, components must conform to a component model which gives the framework of components the ability to provide quality services (Comella-Dorda *et al.* 2000). Components offer a higher-level representation of a domain and are considered more coarse-grained than traditional objects (Lee *et al.* 2003).

Components encapsulate single pieces of business logic (Comella-Dorda *et al.* 2000). They are installed on an application server providing a runtime environment for the component as well as management facilities for common services such as security, transactions, state management, amongst others. This creates an environment where the developer can focus on the business problem requiring a solution.

The first process in component wrapping is to separate the legacy system interface into modules comprising of logical units (Comella-Dorda *et al.* 2000). The next process in wrapping business logic into components is to create a single point of contact to the legacy system. The last process in component wrapping is to implement a wrapper for each

component of the legacy system, thus allowing the wrappers to communicate with the legacy system by sending requests through the single connection point.

The component wrapping technique has several advantages over the other techniques (Comella-Dorda *et al.* 2000). Flexibility is achieved first and foremost by wrapping pieces of business logic. Secondly, there is a full integration and support capability on the application server for all services generated by component-based modernization. Thirdly, this technique paves the way for the substitution of the legacy system incrementally. Lastly, the more sophisticated structured nature of component-based software is a solution to the complex accumulation of classes and objects that may be difficult to manage (Lee *et al.* 2003).

2.6. Conclusion

Two distinct modernization approaches, white-box and black-box, are relevant to this investigation. White-box modernization involves understanding the internals of a legacy application and its functionalities before migrating the code to a more modern architecture (Section 2.2). Comparatively, the black-box approach involves the analysis of user interactions with the interface of the legacy system to gain knowledge of all inputs and outputs.

The advantages of a white-box modernization approach include the depth of knowledge of the functionality of the legacy system, gained from the program understanding phase (Section 2.3). Also, the understanding of the business logic of the original system leads to the development of more modular, componentised code in the modernized system. On the other hand, employing the white-box modernization approach can lead to certain disadvantages. Disadvantages include complexity of analysis and re-engineering of the legacy system, due to possible lacking system documentation. Also, operation cost in terms of time and effort are a serious consideration when applying white-box modernization techniques.

Black-box modernization attempts to address the disadvantages discovered from the white-box modernization approach (Section 2.4). By ignoring system internals and focusing on the interaction with the interface of the system, black-box modernization decreases the complexity of system analysis encountered in the white-box modernization approach. The introduction of a wrapper to encase the legacy system has the advantage of leaving the legacy

business logic and code unchanged, but allowing a modern interface to be added onto the front end of the new system whilst allowing communication with the original legacy code via the wrapper. The disadvantage of this black-box approach is that it becomes a potential short-term solution to modernization, since complications may occur after prolonged maintenance of the modernized system. Black-box modernization is also not always a realistic solution as it may require the incorporation of white-box techniques in some cases.

Various specific implementation techniques exist for each of the white-box and black-box modernization approaches (Section 2.5). Data Modernization provides access to the existing system database through various techniques (Section 2.5.2). Functional modernization provides access to business logic through various forms of wrapping or componentising business logic (Section 2.5.3). User interface modernization, or screen-scraping, is a technique often used during black-box modernization (Section 2.5.1). This black-box technique is used to develop a modern interface for the legacy system based on analysis of the inputs and outputs of the original legacy system interface.

The white-box modernization approach adopted for this investigation incorporates steps from the various approaches presented (Section 2.3). The white-box approach therefore includes a program understanding phase that will require re-engineering techniques to obtain structure diagrams of the legacy system. These system model diagrams will assist with understanding the internal structure and functionality of the system before code migration is possible. The second phase will be the code migration phase, which will transform the existing code into a new language and platform. In the third and final phase, the data services will be discovered through the application of component-based analysis of the legacy system (Section 2.5.4).

The black-box modernization approach will make use of a technique presented during interface modernization (Section 2.5.1). This technique is chosen due to its black-box nature, where the interface is modernized through the analysis of inputs and outputs of the legacy system during user interaction. Component-based analysis will be applied to the existing system in order to identify specific data service components (Section 2.5.3). The black-box modernization approach will then incorporate web-service wrapping (Figure 2.1) of the components identified during component-based analysis of the host program.

Chapter 3: Service Oriented Architectures

3.1. Introduction

Service Oriented Architecture (SOA) is considered an influential development in software evolution (Garcia-Rodriguez de Guzman *et al.* 2007). SOA provides system processes as services and makes these services accessible over the web (Section 3.2). In this way, the potential exists for legacy systems to be wrapped and offered as services.

Two types of service styles exist, namely the Simple Object Access Protocol-Remote Procedure Call (SOAP-RPC) style and the Representational State Transfer (REST) style (Liu and Connelly 2008). SOAP style services have tightly coupled processes and their interactions with other parties are based on business process modelling (Section 3.3). REST style services are handled with a common web invocation in the same architecture. REST services are designed by identifying resource types and are implemented by remote invocation via standard HTTP (Hypertext Transfer Protocol) protocols (Section 3.4).

The two service styles have both been used during the modernization of legacy systems. Various approaches concerning both REST and SOAP services are elaborated upon. These two service styles are distinct and it is necessary to understand both styles to conclude which service architecture better suits this research. A service style that suits the provision of data as a service is necessary for this investigation. Data services are a specific subset of web services that focus on the provision of data as a service in an SOA (Section 3.5). Conclusions

will justify the choice of service architecture for this investigation by analysing the service styles presented in literature and the appropriateness of these styles in terms of the data services required by this study (Section 1.5).

3.2. Service Oriented Architecture and the Web

Provision of shared information to various clients is becoming a necessity and despite differences in programming languages or platforms, the client applications should have access to these shared resources (Lim *et al.* 2010). Web 2.0 is a term that was initiated to describe the direction in which the World Wide Web is headed (O'Reilly 2005). Web 2.0 technologies provide a platform for the development of web applications consisting of a combination of multiple applications (Lim *et al.* 2010). Some simple characteristics of Web 2.0 include:

- Exploit collective and interactive processes;
- Provide richer user experience through data sources; and
- Lightweight interfaces and an architecture that allows interaction of applications.

Service Oriented Architecture represents a model in which small, loosely-coupled software components are deployed, requested and combined with other applications over a network (Canfora 2004).

A web service is a practical implementation of this architectural model (Canfora 2004). These services leave the technical internal details undefined, but communicate via a uniform set of transfer protocols and standards. SOA is a design framework that encapsulates a piece of business logic within a specific service (Wang *et al.* 2007).

The goal when integrating legacy systems into the SOA is to leverage the fast, flexible infrastructure that the SOA provides (Wang *et al.* 2007). As a result, the legacy system can be utilised by more business processes with less reuse of system functionality. Interoperability is improved with various systems that the legacy system interfaces with as a result of integration with an SOA.

There are various benefits of integrating legacy systems with SOA (Wang *et al.* 2007). These benefits include exposing integral business logic and data through service enablement. SOA provides a standard interface for collaboration of services with other systems. Another benefit

of exposing legacy systems as services in an SOA is functionality reuse. Functionality reuse results in reusability of legacy functionality without rewriting of code. Lastly, the adaptability to provide for future business processes or altered business processes is beneficial to the longevity of legacy systems.

Various challenges also exist when integrating legacy systems into an SOA environment (Wang *et al.* 2007). The first challenge is designing the services in such a way that any proprietary data definitions are catered for if the services are required to be consumed by various applications. Secondly, legacy systems originally contain tightly coupled business logic and functionality. Services must therefore be designed to enable the legacy system processes to interact with other services in a more loosely-coupled, flexible manner. Lastly, the performance of services generated from legacy code could negatively impact on critical business processes if not carefully considered. It is advised that the impact on performance be researched and minimised (Wang *et al.* 2007).

SOA plays a major role in exposing these pieces of business logic in a reusable, loosely-coupled manner where code complexities are hidden (Lim *et al.* 2010). Web 2.0 offers two common web-based methods to provide access to data and functionality, namely REST and Simple Object Access Protocol (SOAP). Both REST and SOAP services cater for the exchange of structured information. The REST approach employs the use of simple HTTP protocols whilst the SOAP approach utilises more complex message formats.

3.3. SOAP Services

In a SOAP-based architecture, messages are XML-encoded and transmitted over HTTP (Mulligan and Gračanin 2009). SOAP Services share information amongst one another via SOAP messages and information stored in the service registry is based on the Universal Description, Discovery and Integration (UDDI) standard (Liegl 2007). The interface between service providers, or consumers, and the service registry can be built based on the Web Service Description Language (WSDL) standards.

Sets of SOAP service methods are defined in WSDL files (Mulligan and Gračanin 2009). The WSDL files are XML documents that adhere to a W3c-specified grammar. Web service methods that are written are defined in terms of the structure of parameters that are defined in

service requests or responses. Since the WSDL file defines the SOAP interface for a set of web methods linked to a web service, the WSDL file maps the SOAP messages onto the correct operations via the correct port. The WSDL file needs to be made publicly available to all clients that require access to the web services defined within the file. Thus, the WSDL file is made available on a SOAP-enabled application server.

The architecture of a SOA environment (Figure 3.1) depicts the relationship between the service provider, service consumer and service registry. Enterprise A consists of two services; Service A (consumer) and Service B (provider).

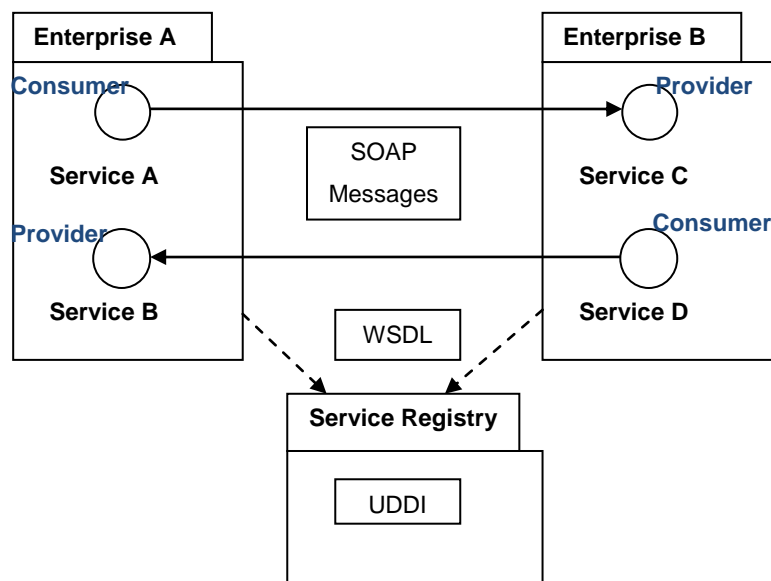


Figure 3.1. Adapted from “A Sample SOA Environment” (Liegl 2007)

Enterprise B similarly has Service C (provider) and Service D (consumer). Standards and protocols such as SOAP, WSDL and UDDI are imposed during this service provisioning process (Figure 3.1). The composition of services is sometimes necessary to achieve a single business objective and this composition is possible through Business Process Execution Language (BPEL) (Liegl 2007). BPEL is a standard which defines and controls the order in which services can be invoked.

The WSDL and UDDI information model allows for dynamic binding of web services to form a consolidated business process (Canfora 2004). An approach for discovering services based on the classification of these services by pre-processing the textual information in a WSDL document is presented. An algorithm, taking as input the goals to be achieved, such as

the service to be found, evaluates the possibility of achieving the goal with a single service. This algorithm identifies the possible services in the service registry and the similarity to the goal is then calculated using an ad-hoc entity matching algorithm. This approach relies on the standards and protocols defined in a SOAP-based architecture to classify and discover services required.

A service-oriented framework consisting of service providers and service requesters forms the architecture in which the black-box modernization strategy is implemented to migrate a GUI-based legacy system to a web service (Zhang *et al.* 2008). The web services are required to be wrapped and deployed in advance. A service wrapper is involved to allow heterogeneous applications to access the web services.

This black-box modernization approach migrates legacy systems towards SOAP services (Zhang *et al.* 2008). To make the legacy system accessible as a web service, the interface is required to be adapted to conform to the distributed paradigm of service-oriented computing. The approach is tailored towards GUI-based systems, where the interfaces are reused and integrated into an SOA environment. Services are combined in this architecture using the BPEL standards. The strength of this approach is the encapsulation of heterogeneous applications, thus promoting reusability.

The most common form of SOA is web services (Lewis *et al.* 2006). These services must adhere to the following:

- Service interfaces described by WSDL;
- Requests and responses sent via SOAP over HTTP; and
- UDDI used as a directory for services.

Enabling legacy systems to operate within an SOA is sometimes as simple as parsing the SOAP requests to invoke the legacy code and then wrap the results and send them out again via a SOAP message.

An approach for the migration of legacy systems to an SOA environment is the Service-Oriented Migration and Reuse Technique (SMART) (Lewis *et al.* 2006). This approach collects a broad range of information regarding the legacy components, the target SOA environment and the possible services to be created (Figure 3.2).

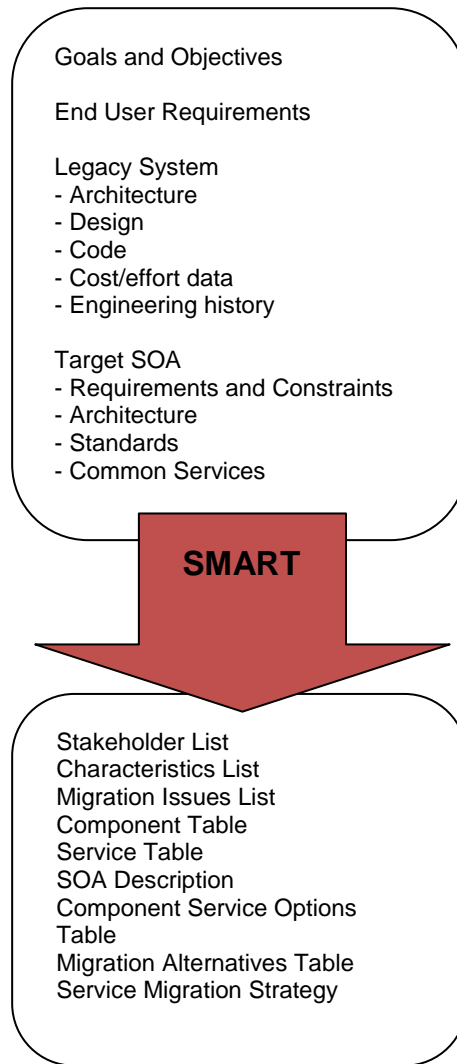


Figure 3.2. Adapted from “SMART inputs and outputs” (Lewis *et al.* 2006)

SMART also produces outputs that are valuable to the organization requiring a possible legacy system migration (Figure 3.2). The conclusions drawn from this process include the benefits to the customers requiring legacy migration due to the inclusion of SMART (Lewis *et al.* 2006). The analysis of the inputs made it clear whether migration was a feasible option for the organization.

SOA allows a new way to continue to use, or reuse, the business processes offered by legacy systems (Canfora *et al.* 2008). This re-use potential is achieved by overcoming interoperability restrictions by making use of web services. Form-based systems are a type of interactive system where the interaction between user and computer is session-based, consisting of the alternating exchange of messages between the user and the computer. Many

legacy systems fall under this classification and a migration approach is presented to modernize these legacy systems to operate within an SOA.

A wrapping approach is presented to expose the legacy system as a web service that conforms to SOAP service standards (Canfora *et al.* 2008). This black-box modernization approach is useful in situations where access to the legacy code is not possible. The wrapper is thus developed to translate users' service requests into a format that the legacy system recognises to process the results required. The result of this modernization technique allows the legacy functionality to be migrated towards the SOA.

Software systems modernization towards SOA provides a beneficial option for the elongation of the lifetime of legacy systems (Canfora *et al.* 2008). Deploying these legacy systems as services promotes the interoperability of heterogeneous systems and this can be achieved by wrapping techniques. The successful implementation of the wrapping approach in related studies showed the cost-effectiveness of this type of modernization. The scalability of this approach was still reported as being uncertain and needed to be addressed by the addition of tools and techniques for the most expensive tasks of the migration process.

SOAP-based web services have a large variety of tool support from vendors (Lim *et al.* 2010). Furthermore, SOAP services are supported by other web service standards such as WS-Security and WS-ReliableMessaging, amongst others. The support of these standards makes SOAP web services more well-defined, providing them with security and error-handling capabilities. Conceptually, with regard to all of the standards supported by SOAP services, these services may be more difficult to understand and develop. With the aid of development tools, however, these services are more simply designed. SOAP-based web services are reported as being flexible because they address quality of service attributes such as security (Pautasso *et al.* 2008).

3.4. REST Services

Services that are based on the REST philosophy are often labelled "RESTful" services (Peng *et al.* 2009). It is reported that RESTful web services are more simple for the developer to understand and create (Lim *et al.* 2010). REST style services are abstractions of the architectural components within a distributed hypermedia system (Fielding and Taylor 2002).

REST focuses on the roles of components and the constraints on their interactions with other components. REST specifically overlooks the details of the implementation of each component and protocol syntax.

The principles of REST can be split into four characteristics (Peng *et al.* 2009), namely:

- Resources are used to define application state and functionality;
- Resources have a unique address defined by a universal syntax;
- A uniform interface is adopted for state transfers between a client and a resource; and
- All interactions with resources must remain stateless.

RESTful web services focus on the simple transmission of messages over HTTP, where resources may be used or controlled using HTTP methods (Lim *et al.* 2010). Furthermore, it is reported that RESTful web services are lightweight since the SOAP messaging layer is not required. The absence of the SOAP messaging layer means that no XML wrapper is required for REST messages, thus reducing their size.

RESTful services have a client-server architecture where communication is stateless (Fielding and Taylor 2002). Thus, every request from the client to the server must contain all information required to process the request. The implication of this stateless style of communication is that the session state is solely on the client side. Scalability is therefore positively affected, since the server is not required to store the state between requests and can free up resources quickly. On the other hand, the disadvantage of the REST architecture is that network performance may be decreased by the increase in repetitive data sent in a succession of requests. Also, storing the application state on the client side implies that server's control over consistency of the application's behaviour is reduced.

The core feature that sets the REST architecture apart from any other network-based architecture is the uniform interface between components (Fielding and Taylor 2002). This uniform interface is achieved by the software engineering principle of generality in terms of the component interface. Implementations are decoupled from the services provided by the components. Thus, independent evolution of components is supported. On the other hand, this uniform interface reduces efficiency due to the fact that data is transferred in a standard form, as opposed to a format which is specific to an application's requirements.

A white-box reengineering process by source code analysis for migrating legacy systems to RESTful web services is presented (Liu and Connelly 2008). The reengineering approach includes the extraction of candidate resources by identifying entities in the legacy code. Blend services are then created by analysing static relationships in the legacy system. Uniform Resource Identifiers (URI's) are designed to represent the services. This is due to the fact that the service will be established on the resources, their constraints and their operations. Existing operations in the legacy system are analysed to assign standard operations to the URI's. The service representations are then designed to create the interface for the service on the web. The existing functionality is then wrapped by the REST service to create the fully functional final product.

The key issue for generating RESTful web services is the identification of resources to define URI's (Liu and Connelly 2008). Related studies have addressed this problem by incorporating the semi-automatic identification of informative entities in the legacy system. This approach from the related study did not, however, consider all issues, such as dependency relationships and their representation in URI's. It was thus advised that these dependencies be studied further to improve the semantics of web service dependence on other web services.

RESTful services do not have standards that support security, reliable messaging or error handling (Lim *et al.* 2010). Developers are therefore required to implement any of these features that they require themselves. Another limitation of RESTful web services is that all operations executed must follow the standard HTTP methods, namely GET, POST, PUT and DELETE. This requirement of RESTful web services may not be a practical solution for all types of applications.

3.5. Data Services

Universal interoperability is achieved by the adherence to the protocols that define web services (Li *et al.* 2009). This interoperability is synonymous with performance loss due to message serialization into the correct format for transmission. It is thus necessary to carefully design and select services in such a way that they are interoperable in nature, yet can still provide responses to requests quickly. These services do not necessarily have to replace an entire application, but they may be designed to act as data stores (Selçuk Candan *et al.* 2009).

A data service has the core functionality of one specific business object, such as an order or an employee (Carey 2006). Data services have a collection of read methods (Borkar *et al.* 2006). These methods are service calls that allow access to instances of business object types in various ways. Similarly, data services have write methods which allow for the insertion, updating, or deletion of different business object instances. Lastly, navigation methods are service calls used to navigate the relationships between business objects returned via different services, for example the relationship between a customer and an order.

These methods represent the CRUD model (Kilov 1990). The CRUD model allows Creating, Reading, Updating or Deleting of data from a data store. Data services can conform to this model as each data service can essentially perform one of the CRUD actions. An SOA will thus provide the infrastructure to support these data services during data transmission (Mulligan and Gračanin 2009).

Data security and service reusability are emphasised in the development of data services, as shown in the AquaLogic Data Services Platform (ALDSP) (Borkar *et al.* 2006). It is advised that access to the data services be controlled, as well as query processing security. Reusability is highlighted as an important feature of data services and is achieved by encapsulation of method call details. For example, the method `getAll()` can be reused with different business object types since encapsulation hides the technical code detail of the method used to return all attributes of an object. Latency, defined as the time taken to achieve results after submitting a query, is another attribute of data services reported as being an important consideration during design of these services.

3.6. Conclusion

The future of the web lies in the provision of interactive, interoperable services to fit consumers' needs (Section 3.2). This new direction for the web has been called Web 2.0. A Service Oriented Architecture provides the framework for the provision and consumption of these services. Data services in particular can be provided in this SOA framework. Two distinct architectural styles for web services exist, namely SOAP-RPC services and RESTful services. It is necessary to contrast and compare these two service styles to make an informed decision on which style is more appropriate for the creation of data services in particular.

SOAP-based services follow standards and protocols such as WSDL for the interface between consumers and the services, and SOAP for the structure and format of request and response messages (Section 3.3). SOAP style services are far more commonly used as a target architecture for the modernization of existing systems and exhibit better tool support for development. SOAP services offer other benefits such as security and reliability of messages during transmission.

Alternatively, RESTful services are more lightweight, due to the elimination of encasing messages in a SOAP request or response (Section 3.4). This lightweight design is achieved by using simple HTTP protocols where messages have to conform to HTTP message formats, namely GET, PUT, POST and DELETE. RESTful services have become more popular in recent years as they were developed to address some of the structural complexities that SOAP services carry. Furthermore, it has been shown that both of these types of services have been used as a target architecture for the migration of legacy systems to the SOA.

Data services encapsulate single pieces of business logic, whilst their design follows a CRUD model and hides implementation complexities (Section 3.5). Data services must also satisfy design requirements such as security, reusability, reliability and interoperability. Taking into consideration the requirements of data services and the arguments presented for SOAP and RESTful web services, the preferred service style for the purposes of the case study in this investigation is SOAP. This is due to the fact that SOAP style services are widely accepted and well documented in literature. SOAP services offer built in security and message reliability (Section 3.3). Although REST services may show better performance than SOAP services due to reduced message size, it has been reported that the true performance benefits of REST require further investigation (Section 3.4).

It is necessary to apply the modernization approaches identified for this investigation (Chapter 2) to a case study in order to create data services that operate within an SOA. The services must adhere to criteria specified for data services in particular (Section 3.5), namely:

- Security;
- Performance;
- Reusability; and
- Interoperability.

Furthermore, these data services will need to conform to the SOAP standards and protocols presented (Section 3.3). An appropriate case study has been selected for the application of these modernization approaches to leverage the data from the existing system as a service.

Chapter 4: Case Study Modernization using Data Services

4.1. Introduction

The two modernization approaches, namely white-box and black-box, will be applied to a legacy system case study for this investigation. A legacy system that is currently in use in the Department of Computing Sciences at Nelson Mandela Metropolitan University will be modernized (Section 4.2). The current system requires analysis at a conceptual level to understand the domain.

These two modernization approaches will make use of appropriate specific implementation techniques (Chapter 2). The white-box modernization approach will follow the process of first analysing the internal functionalities of the system to form a higher level representation of the system (Section 4.3.1). A code migration phase will follow this program understanding phase to translate the legacy code into a more modern format (Section 4.3.2). In the last phase of the white-box approach the specific data services are discovered and deployed to finalise the modernized system (Section 4.3.3). The black-box modernization approach will entail the creation of a modern interface for the existing system (Section 4.4.1). The componentization of the original system code is necessary to divide the functionality into data services for web service discovery. This component-based modernization technique will follow the user interface modernization (Section 4.4.2).

The research question regarding the application of both modernization approaches to the case study in order to generate data services will thus be answered (Section 1.5). Conclusions will revisit the approaches conducted on the existing system and motivate the need for evaluation of these modernized systems to realize their merits.

4.2. The Student Assistant System

The Student Assistant System, or Demi System as it is informally known, is currently in use by student assistants for computer-based courses in the Department of Computing Sciences at NMMU. The Demi System was developed to reduce the paper load required for the hiring and management of student assistants in the department (Figure 4.1).

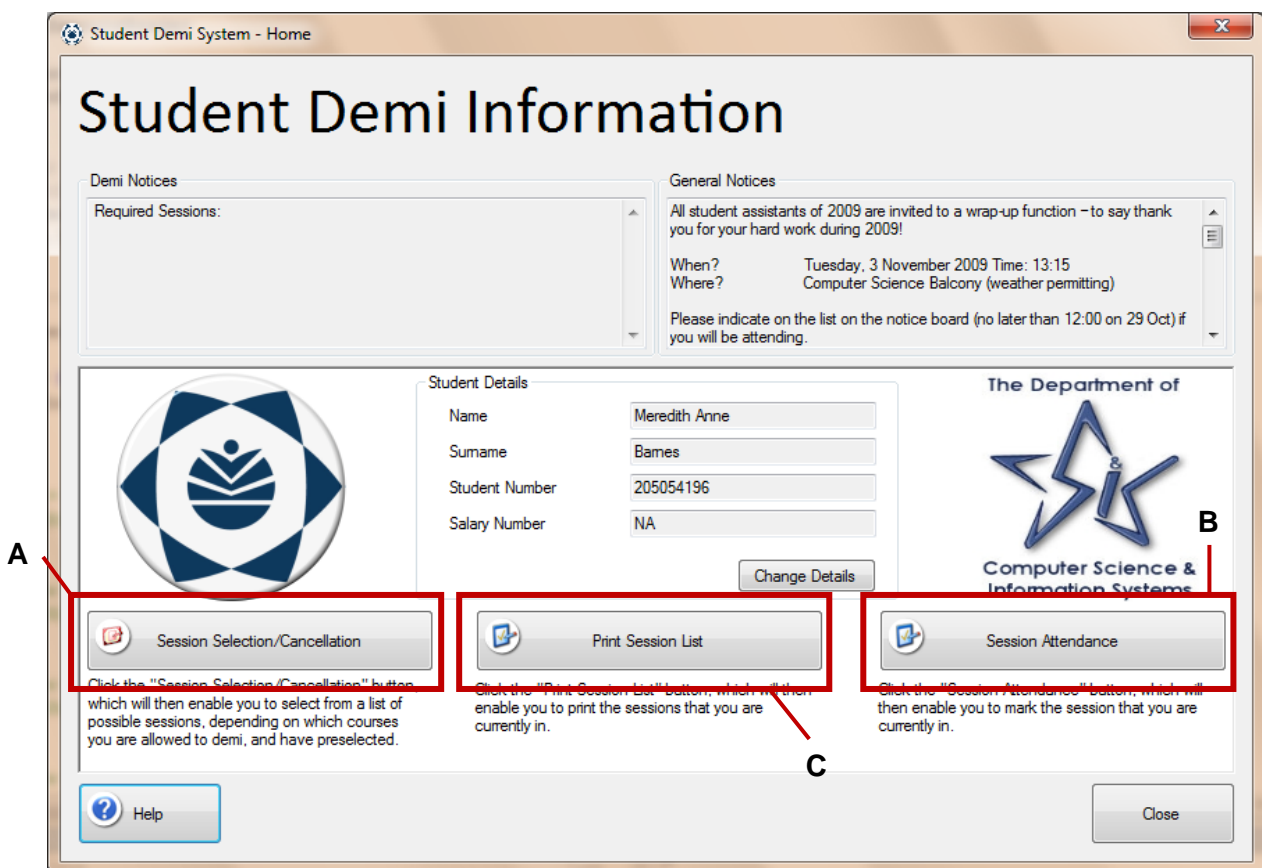


Figure 4.1. Demi System Interface

Originally, prospective student assistants were required to fill in large paper-based application forms and submit these forms to the administrative staff to be processed. If their application was successful, the student assistants were then required to manually select the courses that they wished to assist with. As the student assistants are paid for their services,

they were required to sign off their attendance at each session on an attendance sheet which was subsequently submitted to an administrative staff member for capturing purposes.

To reduce the amount of forms required, the automated Demi System was developed. The Demi System allows prospective student assistants to apply for positions by completing an electronic application form (Figure 4.2).

The screenshot shows a web-based application form titled "Application Form" for the Department of Computer Science and Information Systems, specifically for a "Student Assistant Application: 2010". The form is divided into three tabs: "Personal Details", "Previous Experience", and "Lecturer". The "Personal Details" tab is currently selected. The form contains several input fields and radio button options. The "Student Number" field is filled with "205054196". The "Title" field has radio buttons for "Mr", "Miss", "Mrs", and "Ms", with "Mr" selected. The "Correspondence Preference" field has radio buttons for "English" and "Afrikaans", with "English" selected. The "I can speak fluently" field has checkboxes for "English", "Afrikaans", and "Xhosa", with "English" checked. The "Year Level in 2010" field has radio buttons for "1", "2", "3", and "4 and higher", with "1" selected. The "Period Available in 2010" field has radio buttons for "Entire Year", "1st Semester", and "2nd Semester", with "Entire Year" selected. At the bottom of the form, there are "Save" and "Cancel" buttons.

Figure 4.2. Demi System Electronic Application Form

After the form is completed, confirmation emails are sent to the relevant lecturer to make the recommendation needed for the application to be successful. Once the student has been accepted, the student may then log into the system to select the courses that they would like to assist with (Figure 4.1 A). Users are able to choose specific session times from lists of courses that they are qualified to assist. A confirmation email of the student's choice of session will be sent to the relevant lecturer of the course for approval. After session selections are made, it is the student assistant's responsibility to attend the sessions that they have been

approved to assist. During the session they may log into the Demi System in the computer lab in which the session is held to mark off their attendance electronically (Figure 4.1 B). Their attendance is thus logged so that their total hours can be calculated for the correct payment of their services. At any point in time if the student wishes to stop assisting with a specific session, they may log into the Demi System to remove the session. This action will again send a confirmation request to the lecturer who coordinates the course.

Student assistants who have selected sessions with which to assist may also use the Demi System to view a tabulated list of their chosen sessions (Figure 4.1 C). The list of sessions for a given student is available for the student to print. A student may also request to view their total hours attended for certain periods of time, as specified by the student using a specified date range. The total hours worked may also be printed out.

The Demi System has now been in use for two full years in the Department of Computing Sciences. Despite the recent development of this system, it portrays characteristics of a legacy system that allow for the modernization of the system. This system was developed in house to suit the needs of the staff and students of the Computing Sciences department. No system documentation exists, thus making maintenance of the system a challenge for anyone other than the system developer. Furthermore, the system was developed in a language that is no longer supported in the department, thus complicating future maintenance. The white-box analysis of the Demi System has to include the generation of system representational diagrams to understand the domain. Verbal communication with the developer of the Demi System also proved useful in understanding the roles of the legacy system.

A more technical analysis of the legacy system included the investigation into programming language used and the environment on which the system runs. The Demi System was developed in an Object-Oriented programming style using Visual Basic (VB) code. The database used by the Demi System is a Microsoft SQL Server database. The Demi System runs within a client-server architecture with the database residing on a server in the department to which all client computers in the computer labs have access. All functionality is programmed on the client side and only the database remains on the server. The significant role that data operations play in the functionality of the Demi System makes the system a prime candidate for the modernization of the legacy system into data services.

After a higher-level requirements and design analysis of the Demi System was completed, the modernization approaches were ready to be applied to the system. The white-box approach required a further, lower-level analysis of the legacy system at the code level. This more detailed analysis prepared the way for the migration of the legacy code and functionality to a modern architecture. The black-box approach required analysis of the inputs and outputs of the system to design modern interfaces for the new system architecture. Component-based analysis of the legacy system was required for both approaches to identify specific data services.

4.3. White Box Modernization Approach

The white-box modernization approach applied to the Demi System follows the principles identified for the approach (Section 2.3). The approach consists of three phases (Figure 4.3).

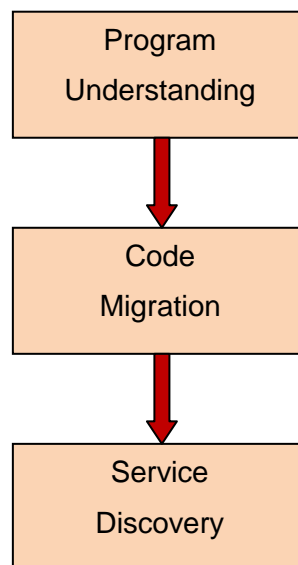


Figure 4.3. White-Box Modernization Phases

An invasive approach is used to first gain understanding of the code structure of the legacy system as well as the system schematics. This in depth analysis of the legacy code contributed toward the program understanding phase (Section 4.3.1). Thereafter, code migration is necessary to modernize the internal functionality of the legacy system, whilst the data source remains intact (Section 4.3.2). Thirdly, the discovery of the data services as a result of component-based analysis of the legacy system concludes the modernization of the system (Section 4.3.3).

4.3.1. Program Understanding

Before commencing code migration from the legacy code to the modernized code, it was necessary to understand the internal code structure and functionality of the legacy system in lower level detail. Relationships and dependencies between classes needed to be understood, as well as what subsequent method invocations occurred on specific method calls. Diagrams and higher level abstractions of these relationships and dependencies were created during the program understanding phase to aid in comprehension of the system, as identified (Section 2.3).

A software program that automates the code analysis and creation of these schematics was utilised for this phase (Salste 2008). A class-level dependency diagram was created to depict how classes instantiate new instances of other classes during runtime of the system (Figure 4.4).

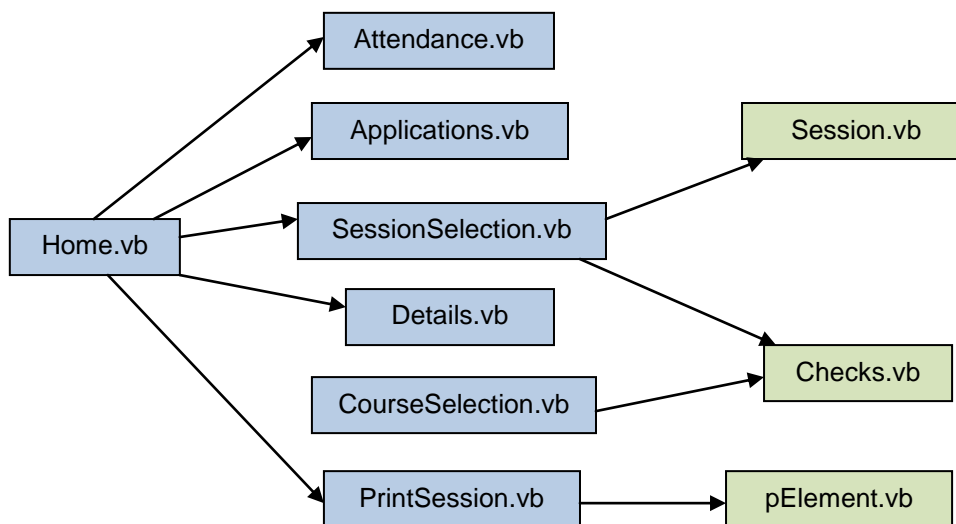


Figure 4.4. Class Dependency Diagram

The classes shaded in blue were all identified as interface classes depicting Windows forms, each with its own functionality relating to the specific requirements of the interface (Figure 4.4). For instance, the Home class allows users to move to the other interface classes such as Attendance or Applications. The classes shaded in green were identified as business logic classes, each containing functionality to support the needs of the interfaces that rely on them. Specifically, the Session class is used to store Session information retrieved from the database for use by the SessionSelection class. The Checks class contains data operations required by other interface classes and the pElement class contains code

specifically for the formatting of data when printing reports. This functionality is required by the `PrintHours` class. The `CourseSelection` class appears to be independent of the `Home` class. No explanation is provided in this diagram as to how the class is used however. A more detailed method analysis diagram was generated to view method invocations (Figure 4.5).

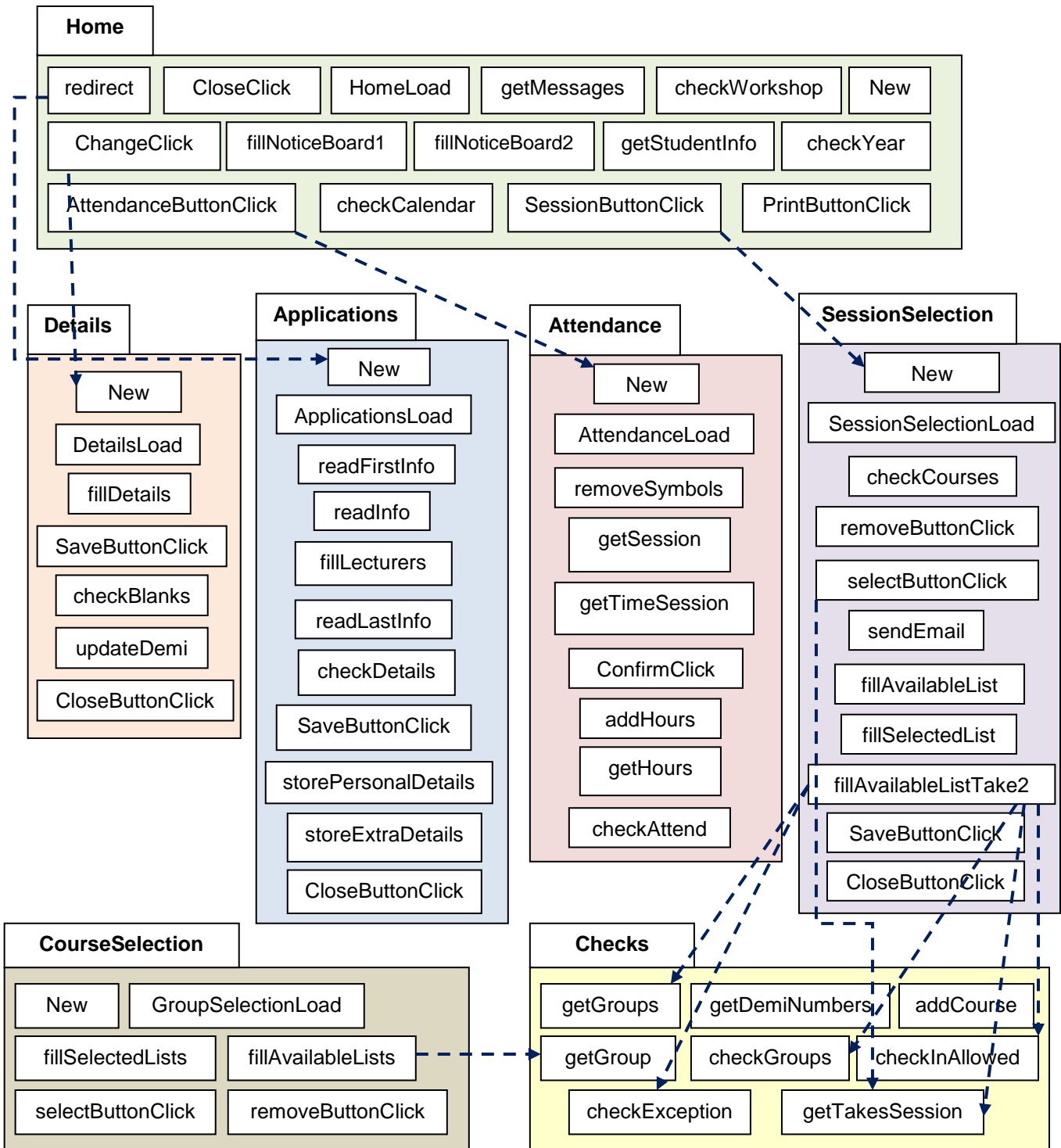


Figure 4.5. Extract from Method Dependency Diagram

This diagram illustrates how the calling of certain methods instantiate new classes. From this system representation it is clear to see that the `CourseSelection` class is never instantiated from any other class in the system. Only information pertinent to the investigation is shown in the diagram depicting method dependencies across classes (Figure 4.5). The class dependencies show the relationships between classes which is useful for understanding how the classes relate to one another during use of the system. The diagram does not, however, draw any attention to which classes may be insignificant due to the existence of dead code (Figure 4.4).

Since the *New* method of the `CourseSelection` class has no arrow pointing to it, the implication is that this method is never called (Figure 4.5). Thus, `CourseSelection` is never instantiated. The other direct implication from closer examination of the diagram is that the `getGroup()` method in class `checks` is never called, despite the arrow. This is due to the fact that the method that invokes `getGroup()` is never called as it exists in the `CourseSelection` class. The `CourseSelection` class had become obsolete after modifications were made to the system, but it had not been removed from the project, thus leaving behind dead code. As a result of thorough analysis of these system representations (Figure 4.4, Figure 4.5), certain classes were found to be suitable for migration (Table 4.1).

Class Name	Description
Home	Interface providing welcome information and links to other actions such as session selection, session attendance and printing lists of sessions
Details	Interface providing functionality to change user information
Applications	Interface providing electronic application form for student assistants
Attendance	Interface allowing student assistants to mark off their session attendance
Session Selection	Interface allowing addition and removal of sessions from a list of available sessions
Checks	Performs queries to validate data retrieved from data source
Print Session	Interface providing printing of a user's selected session list as well as the action to calculate and print total hours worked in a given date range
Session	Stores Session information retrieved from database regarding sessions
pElement	Formats table information for printing purposes

Table 4.1. Legacy System Classes Suitable for Modernization

Another system representation generated during the program understanding phase was the subsystem report (Figure 4.6). This report identified file groups that formed independent subsystems of the legacy system. These subsystems were split and listed for potential reuse purposes as components (Section 2.5.3).

Independent subsystems are depicted in the blocks with boldface text, whilst the classes that these subsystems depend on are included in the block (Figure 4.6). The possible subsystems identified by the program understanding software indicate that attendance, printing sessions, session selection, applications and details are independent of one another. These independent subsystems and the classes that they are associated with may be suitable components for the discovery of data services.

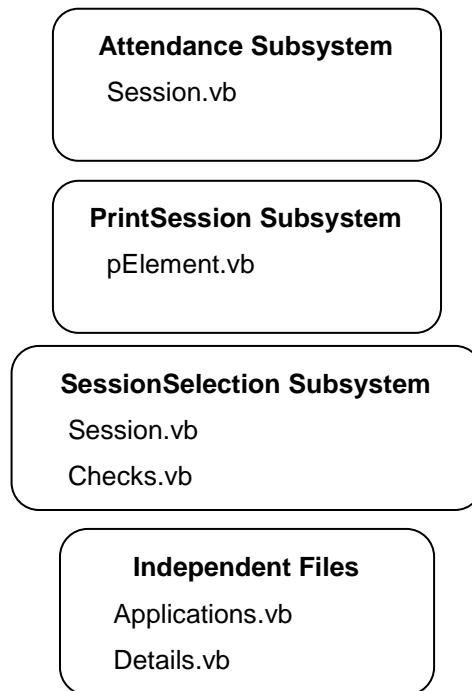


Figure 4.6. Subsystem Representation

Now that a much deeper investigation into the Demi System has been conducted, the process of code migration is necessary to modernize the existing system functionality. Throughout the entire white-box modernization approach the original Demi System data source remains unaltered. Only the business logic and code functionality is migrated towards a modern architecture.

4.3.2. Code Migration

To remain true to the white-box legacy system modernization approach, it is necessary to migrate the code away from its original language (Section 2.3). Java was chosen as the new language that the original system code would be migrated towards, therefore seeing as the Demi System was developed in VB code, the VB code was treated as though it were legacy code. In addition to this, the core functionality of the modernized system is required to remain the same as the original system.

As a result of the preservation of system functionality and the nature of the system used for this investigation, the decision was made to maintain the same look and feel in the interfaces of the modernized system as the original system. The adherence to these modernization rules posed certain challenges during development due to the differences in programming languages and native capabilities of these languages. The original Demi System, having being developed in the .NET environment, imported native collections to provide printing and emailing capabilities to users. The migration of the existing code to a Java environment thus meant that the new Demi System was required to provide users with the same printing and emailing capabilities as the original system.

To overcome this challenge, the inclusion of the JavaMail API was necessary. The JavaMail Application Programming Interface (API) provides a platform-independent framework to compose and send emails in an application (Rajshekhar 2005). The inclusion of this API enabled the creation and transmission of email messages required to be sent at various stages throughout the system. To resolve the printing challenge in the new language, the JTable component proved a useful tool as it provides a simple API that allows the user to print tables from the interface of a running application (Figure 4.7).

The JTable has a `print()` method which triggers a standard printing dialog (Figure 4.7). The user may then specify the printer to which they wish to print as well as how many pages and so on. The developer may use one of several overloaded print methods to define the format of the printed table. Inclusion of a page header with a title or a footer with a page number is simply achieved by the creation of a `MessageFormat` object. This `MessageFormat` object may then be passed as a parameter in the JTable print method.

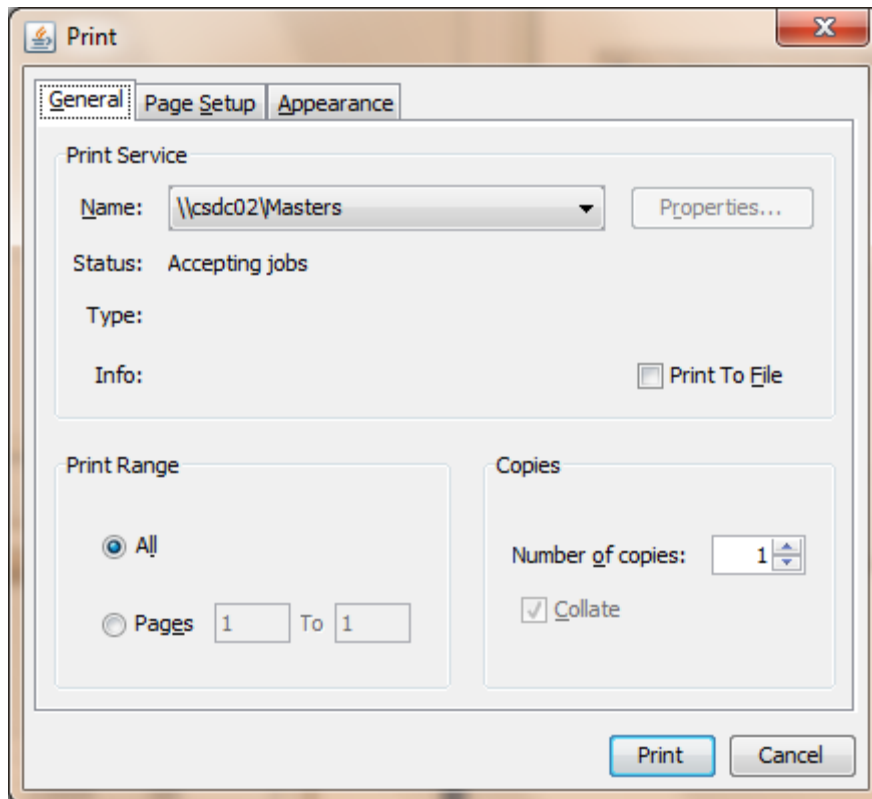


Figure 4.7. Java Print Dialog

Once these challenges were overcome, the interfaces for the modernized system needed to be designed. One major architectural change to be considered when modernizing the legacy code would be the separation of functionality into server-side logic and client-side logic. Any data operations would need to be processed on the server-side of the modernized application, whereas any other system functionality could remain on the client-side. This division of system functionality did not occur in the existing Demi System, as all functionality was programmed into the client-side of the application.

The implication of the separation of code into data operations and client-side functionality prepared the system for service discovery. This conceptual division of code allowed the discovery of specific data service methods which could be grouped into distinct data service entities. The programming of the client-side of the application would thus only include interface and system state maintenance operations. More specifically, any functionality not pertaining to data retrieval, modification or deletion was programmed into the client-side application. The separation of system functionality into client-side logic and data services reduced the amount of service requests and responses messages transmitted during runtime

(Li *et al.* 2009). The data services were designed in this manner to maintain good performance in the modernized Demi System.

The interfaces for the Home, Application, Attendance, Details, SessionSelection, PrintSession and PrintHours classes were designed to mimic the interfaces used in the original Demi System. The PrintHours interface in the modernized legacy system was created as a separate class as opposed to a dialog, as it existed in the legacy system. A dialog interface was created for the “Help” button on the Home interface (Figure 4.1) in the modernized system for ease of development. The Help interface provides users with an image of a button appearing on the Home interface supplemented with a detailed description of the action that the Demi System performs on selection of the button.

Due to the Object-Oriented nature of the original Demi System code, the class structures of the modernized system remained the same. Any methods that performed data validation of values entered into text fields on the interfaces remained on the client-side of the application. Methods that converted object types into the correct format for display purposes or for service requests were developed on the client-side. Methods that populated interface components with data retrieved from the database also remained on the client-side implementation.

Due to the differing programming environments, a developmental challenge was encountered. Not all components used in the original legacy system were supported in Java. These components included:

- The data grid component; and
- The date picker component.

To allow the modernized system to maintain the same interface look and feel as the legacy system, it was necessary to find suitable Java components that could perform the same functions as the .NET components. The JTable was used as an alternative to the .NET data grid component. The inclusion of the JTable meant that a Table Model was required to populate the component during runtime of the system. The implementation of the Table Models for the various JTables was not a direct code translation exercise, as the legacy system data grid component did not make use of any such model.

The Java Table Model is an interface which may be extended in the Java application to leverage inherited methods (Flanagan 2001). The implementation of this table model organises your data and displays it in a tabular format despite what the underlying data actually looks like. Certain methods are required to be implemented when using the Java table model. These methods include:

- `getColumnCount();`
- `getRowCount();`
- `getColumnName(int columnIndex);`
- `getValueAt(int rowIndex, int columnIndex);` and
- `isCellEditable(int rowIndex, int columnIndex).`

These methods assist in the correct formatting and display of the data in the JTable component managed by the table model. In addition to these methods, methods to add rows, delete rows and refresh the table after any changes were included in the implementation of the table model.

After implementation of the interface and the client-side logic was complete, the data services had to be developed with the specific data operations pertaining to them. The components identified during program understanding (Figure 4.6) were a guideline for the discovery of specific data services.

4.3.3. Web Service Discovery

Service-enabling an entire legacy system is not always a feasible option (Wang *et al.* 2007). If only certain portions of functionality are required in the SOA, then it is necessary only to deploy the relevant functionality as services. The existing system components identified during program understanding contain the functionality necessary to be modernized into data services. These data services are required to be developed in adherence to SOAP standards and protocols (Section 3.6). The migration of the existing system to an SOA is thus necessary (Section 4.3.3.1).

The services identified during component-based analysis of the Demi System each contain the data operations related to the business process encapsulated by the service (Section 4.3.3.2). The data services identified from the analysis of the Demi system resulted in the discovery of eight data services (Table 4.2).

Service	Description
Application	Handles applications to become a student assistant
Attendance	Handles marking attendance for a session
Change Details	Handles any updates to a student's personal information stored on the system
Checks	Performs data validation queries to populate tables on interfaces
Login	Handles student verification and notice board message collection on login
Print Hours	Handles queries to view hours worked over a specified date range
Print Session	Handles queries to view the student's list of selected sessions
Session Selection	Allows student assistants to add and remove sessions

Table 4.2. Web Services Discovered during White-Box Modernization

The Application, Attendance, ChangeDetails, Checks and SessionSelection services were directly implicated during the program understanding phase. After the code migration phase, the necessity for the Login, Print Hours and Print Session services as distinct services was made clear.

4.3.3.1. Data Service Architecture

The data services were developed in Java using Netbeans Integrated Development Environment (IDE). Netbeans was chosen as the implementation tool for the data services, since after their development they could be deployed on the GlassFish application server. GlassFish is an open source application server which implements features from the Java Enterprise Edition (EE) 5 platform. Netbeans allows the developer to define and create stateless SOAP-style services.

Several libraries exist within the Java Enterprise Edition (J2EE) to support web services (Kachru and Gehringer 2004). J2EE provides a platform-independent framework for the development of web services. Applications developed in platform-independent environments can be run on different operating systems without the need for any changes. Java achieves this platform independence through the generation of byte-code instructions by the Java compiler for the Java Virtual Machine (JVM).

Containers are provided for the developer to simplify the development of multi-tier applications. These containers provide specific complex functionality, thus allowing

developers to focus on creating the business logic portion of the application. The development of the web services for this investigation is based on this framework (Figure 4.8).

Presentation and Access	Interactive GUI (Swing)	Web Service Clients
Business Logic	Client-side Classes	Data Services
Connectivity	JDBC	SOAP
Runtime	Java Runtime Engine	

Figure 4.8. Adapted from “J2EE Architecture” (Kachru and Gehringer 2004)

The Java API for XML Web Services (JAX-WS) allows the development of stateless web services. These web service classes are created at the business logic layer (Figure 4.8). The web services are created by providing specific annotations when creating the service classes (Figure 4.9).

```

package ws;

import javax.jws.WebService;

@WebService()
@Stateless()

public class SessionSelection {
    /* web service methods implemented here */
}

```

Figure 4.9. Java Web Service Annotations

Annotations are modifiers that, when inserted into a class using the specific “@” notation, can provide the Java interpreter with additional information about the class (Sangeetha and Chinnici 2007). Also, the correct Java `WebService` package must be imported into the project (Figure 4.9). The addition of the `@WebService()` annotation informs the Java interpreter that all methods in the class are intended to be published as web services (Eckstein and Mordani 2006). The `@Stateless()` annotation provides metadata that declares the web service as a stateless session. The implementation of stateless sessions means that any

state maintenance of the application is required to be handled by the developer on the client-side. This reduces the size of service requests and responses as the state of the application is not transmitted back and forth between the client and the server.

Each web service identified during the web service discovery phase of modernization had to be implemented in such a manner as to provide data as a service (Table 4.1). All methods pertaining to data creation, retrieval, updating or deletion (CRUD) were thus implemented as web methods in their respective web service classes. The web methods were also created to follow the GET and SET format identified in similar studies (Borkar *et al.* 2006). The data services were created by the division of data operations based on the component-based logical analysis of the legacy system code. These data services used the Java Database Connectivity (JDBC) API for the connection to the Demi System database (Figure 4.8). These data connections had to be migrated from the original system code due to the difference in programming environments.

4.3.3.2. Implementation of Data Services

The Login service was created to perform all processes required on logging in to the Demi System. The retrieval of messages for the notice boards on the Home interface is handled by the Login service. Any queries to the database to retrieve student information to populate the Home interface are handled by this service. The first and most important query performed by the Login service is the check to see if a student exists in the database (Figure 4.10). If the student does not exist in the database then it is necessary to route them to the electronic application form.

```

public boolean checkStudent(String studentNo){
    boolean found = false;
    /* Connection instantiations made here */
    try {
        /* Statement instantiation made here */
        sql = "SELECT * FROM Demi WHERE StudentNo = '"+studentNo+"'";
        ResultSet r = statement.executeQuery(sql);
        if (r.first()) {
            found = true;
        }
    } catch (Exception ex) {
        /* Exception handled here */
    }
    return found;
}

```

Figure 4.10. Excerpt from Check Student Method

The `CheckStudent` web method in the `Login` web service class takes the student number as a parameter to perform a query on the database to search for the required student. If the student is found in the database, the `CheckStudent` method returns a true value; otherwise the result of the method is false. On the client-side of the application the response from the web service is checked to determine whether the student may log in to the system if their information exists in the database. The client application will alternatively reroute the student directly to the application form if the `CheckStudent` method returns a false value.

The `Application` web service performs `GET` methods to populate some of the selection components in the interface with data from the database such as lecturer's email addresses. The student is required to select from the lists of lecturers whom they wish to provide a character reference for their application. The `Demi` system will send the email request to the selected lecturer for consideration. The student's application will be pending after submission of this electronic form until a response is received from the lecturer.

This web service also performs the essential `SET` methods for the addition of a student to the `Demi System` database for future login purposes. The `SetPersonalDetails` and `SetExtraDetails` methods perform insertion queries to add the student information from the application form to the database. These methods return a boolean value which the client application uses to determine the success of the insertion query. The student assistant applicant is thus informed of their successful application by means of a confirmation message dialog (Figure 4.11).

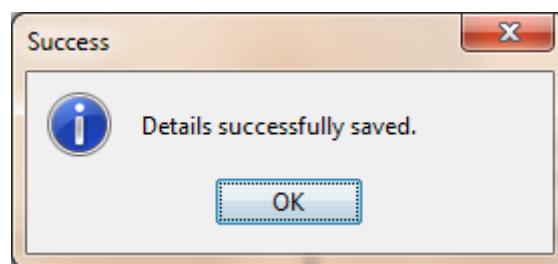


Figure 4.11. Successful Application Dialog

The student is also informed that after their information has been saved that their application is pending until their selected lecturer responds to the email notification sent to them. As soon as the student has been approved by the lecturer their status is changed in the database to give

them access to the system. After access has been gained to the system, the student assistant may log in to the system to select their sessions which they wish to assist. The Session Selection web service handles the session addition and removal functionality (Table 4.1). The `SessionSelection` interface allows users to select their preferred sessions from a list of all available sessions (Figure 4.12).

Session Selection

Total Amount of Hours:
(Maximum of 6 hours)

Note: Simply click the close button to cancel any changes made.
(This is only applicable before you have clicked the save button.)
If you close this form before saving, any changes will be lost!

Selected Sessions					
CourseCode	Day	StartTime	EndTime	NoHours	Location
WRA202	Tuesday	15:15	16:35	1.3333334	Lab2
WRA202	Monday	15:15	16:35	1.3333334	Lab2
WRFC102	Monday	10:45	12:05	1.3333334	Lab2

Available Sessions					
CourseCode	Day	StartTime	EndTime	NoHours	Location
WRC202	Tuesday	13:45	15:05	1.3333334	Lab2
WRFC102	Tuesday	07:45	09:05	1.3333334	Lab1
WRFC102	Thursday	09:15	10:35	1.3333334	Lab4
WRFE101	Wednesday	17:30	18:50	1.3333334	Lab4
WRFE101	Wednesday	19:00	20:20	1.3333334	Lab4
WRFE101	Wednesday	17:30	18:50	1.3333334	Lab5

<- Add Remove ->

Save Close

Figure 4.12. Session Selection Interface

The data for all the available sessions and the student’s selected sessions needs to be queried from the database. Thus, the Session Selection data service provides GET methods to populate the “Available Sessions” table in the interface (Figure 4.12). The “Selected Sessions” table is populated by the GET method from the Print Session data service. After selections or removals have been made, the user must save their changes before closing the interface. If any sessions have been selected, the save action will invoke the data service’s SET-style method to add the chosen session to the list of selected sessions for the current user. The parameters required for this action are the student number of the user and the unique session identifier for their selection.

The save action also makes use of the Checks data service to perform some validation on the session data from the database. Each session has a limited number of student assistants that

may attend the sessions. A query needs to be run when a session is selected to verify that there is still space available for another student assistant. The appropriate web method from the Checks data service is thus invoked to verify the session's availability before the session is booked for the user.

Once a user's sessions have been allocated to them, they need to attend these sessions to use the Demi System to mark off their attendance. The Attendance data service allows users to perform this functionality. The user is required to mark off their attendance from a computer in the lab in which the session is held. Furthermore, the student must mark their attendance between the start and end times of the session. The Attendance data service performs GET methods to determine the student assistant's session for which they may mark their attendance. For the student assistant to confirm their session attendance, the Attendance service provides a SET method to insert their session attendance into the database.

Once the student assistant has confirmed their attendance of the session they may not attempt to mark off the attendance for the same session again. The Attendance service provides a method which checks if the user has already confirmed their attendance for the current session. The Attendance service provides another method which verifies that the student assistant does not attempt to mark off their session attendance on a public holiday or outside of allocate term times. An appropriate error message informs the user that they have attempted to mark off their session attendance at the incorrect time.

Student assistants may use the Demi System to view and print their list of selected sessions. The Print Session service allows users to perform this functionality. The GET methods provided by the Print Session service take the student number of the current user as a parameter. These methods then query the database as to what sessions the user has selected in order to populate a table on the interface (Figure 4.13).

The GET methods in the Print Session data service are reusable throughout the system, as mentioned when populating the "Selected Sessions" table in the Session Selection Interface (Figure 4.12). The reusability of these methods is a benefit of the modernization of the legacy system which resulted in the removal of repeated code in two different classes.

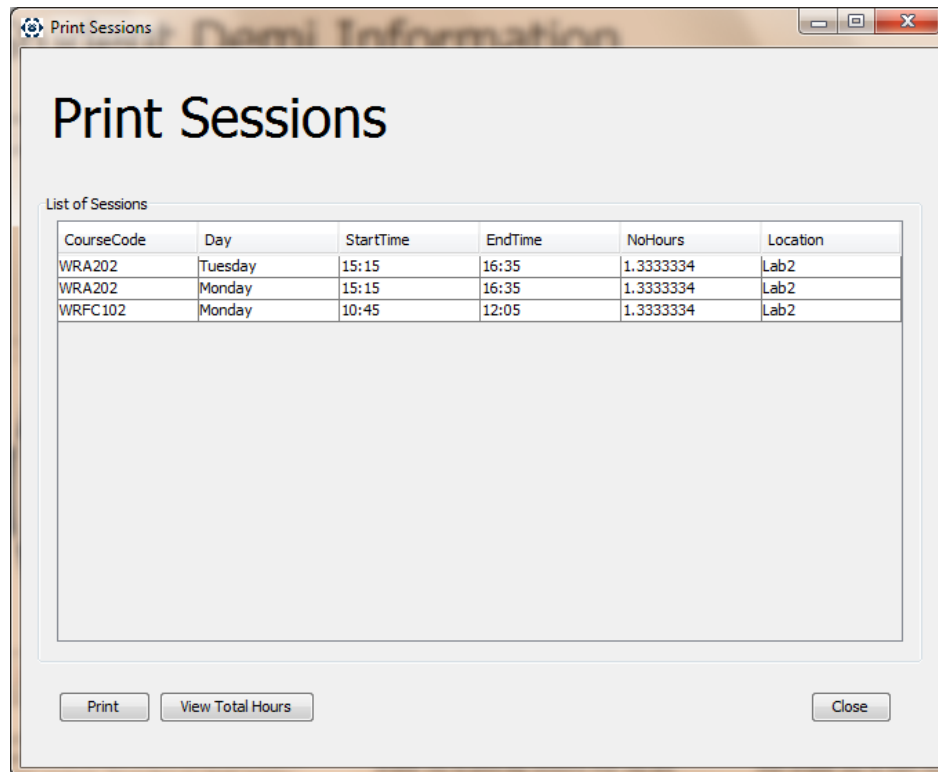


Figure 4.13. Print Session Interface

The Print Hours data service allows student assistants to request the total number of hours that they have worked within a given date range. The GET method for this data service takes in two Date parameters that signify the start and end of the range over which the hours are summed. The method then queries the database for the sum of the hours within the date range. The result of the query is then displayed in a table in the GUI for the user to view and print if they wish.

Lastly, the Change Details data service provides the student assistant with the functionality to update their personal information stored by the Demi System database. The GET method for this service populates the interface with the user's personal details, whilst the SET method updates the changes which they have made. The SET method is initiated when the save action is performed.

These services were clearly identified as separate logical components during analysis of the Demi system. The program understanding phase proved useful in the generation of high-level system representations. These system representations were of great use for the service

discovery phase. The next modernization approach has no detailed program analysis phase, but instead has an analysis of the inputs and outputs of the system during user interaction.

4.4. Black Box Modernization Approach

The black-box modernization approach applied to the Demi System followed the black-box principles discussed in related studies (Chapter 2). The user interactions with the interface of the legacy system were analysed to identify the inputs and outputs of the existing system (Section 2.4). This knowledge of the inputs and outputs would be used to create modern interfaces for the modernized Demi system (Section 4.4.1). The inputs and outputs of the existing system also highlighted the request and response formats for the data services.

Thereafter, a more non-invasive approach was used to wrap the original system code as data services. The identification of these data services, however, was performed by doing a component-based analysis of the Demi system, which was a more invasive procedure. This component-based analysis was necessary to identify the processes required to be transformed into data services since not all existing system functionality could be published as services (Section 4.4.2).

4.4.1. User Interface Modernization

Similarly to the white-box modernization approach, the interfaces for the black-box modernization were designed to have the same look and feel as the original Demi System. Therefore, all interfaces remained the same with the exception of the `PrintHours` interface which was created as a separate `JFrame` class instead of a dialog, as it was in the original Demi System. For the purposes of this investigation, to remain true to the black-box modernization guidelines the modernized interfaces were developed in Java using the Netbeans IDE to design the Graphical User Interface (GUI). The choice of Java as the programming language was made to migrate the interface code away from the existing code that the Demi System was developed in (Section 2.5.1).

The analysis of all of the inputs and outputs during interaction with the user interfaces of the original Demi System revealed which data operations were required to perform certain actions. These data operations would need to be split into logical components to form the data

services. The interactions with the interfaces of the system were observed during use to record all inputs and outputs (Table 4.3).

Interface	Inputs	Outputs
Home	Login information	Read messages on notice boards; View name & student no. after log in; View message if no workshop attended; Divert to application if not found
Application	Entering of all requested information	List of lecturers email addresses; Send email notification to lecturer; Important notices regarding application
Attendance	Confirm attendance of session	Current session being attended; Error notices if attendance not possible
Change Details	Edit information to be updated	View information for editing purposes
Session Selection	Add a session to list of selected sessions; Remove sessions from selected sessions	View list of selected sessions; View list of available sessions; View message if session fully booked; Send email notification to lecturer
Print Sessions	-	View list of selected sessions; Print list of sessions
Print Hours	Specify date range to calculate total hours worked	View total hours worked for given date range; Print total hours worked

Table 4.3. User Interactions with the Legacy System

The inputs and outputs recorded from this analysis could be split into data operations required to be performed by the data services and functionality that could be performed by the client application. Specifically, the printing functionality provided by the `PrintSessions` and `PrintHours` interfaces would not be required to be implemented as a web service for this investigation (Table 4.3). Similarly, the email notifications were not required to be implemented as data services. Thus, the same challenges were met in the modernization of the interfaces as with the white-box approach regarding email and printing (Section 4.3.2).

In the case of the black-box modernization approach, a Java client was developed to control the interfaces. This Java client would be required to interface with a .NET back-end which would house the data services. The creation of the modernized system in this structure allows for web service wrapping of the existing system code (Section 2.5.3). On the client-side of the application, the implication of this design is the formatting of data types to transmit and receive the correct requests and responses respectively.

4.4.2. Component Based Service Discovery

Similarly to the white-box modernization approach of the Demi System, the black-box approach required some component-based analysis of the existing system to separate the data operations from any other system functionality (Section 2.5.3). The classes existing in the Demi System were organised in an Object-Oriented manner, as previously discussed. Thus the data services were discovered by identification of logical components (Table 4.4).

Service	Description
Application	Handles applications to become a student assistant
Attendance	Handles marking attendance for a session
Change Details	Handles any updates to a student's personal information stored on the system
Checks	Performs data validation queries to populate tables on interfaces
Login	Handles student verification and notice board message collection on login
Print Hours	Handles queries to view hours worked over a specified date range
Session Selection	Allows student assistants to add and remove sessions; Performs queries to retrieve selected sessions for the student logged in

Table 4.4. Web Services Discovered during Black-Box Modernization

In comparison to the J2EE environment for web service application development, the black-box modernization approach made use of a combination of the J2EE and the .NET frameworks (Figure 4.14). The grey-shaded blocks represent the portions of the modernized Demi system which operate within the J2EE framework. The white blocks represent the portions of the system operating in the .NET framework. It is thus clear to see that the data services are controlled by the .NET environment, whilst the GUI operations are handled by the J2EE environment.

Presentation and Access	Interactive GUI (Swing)	Web Service Clients
Business Logic	Client-side Classes	ASP.NET Services
Connectivity	ADO.NET	SOAP
Runtime	Common Language Runtime	
	Java Runtime Engine	

Figure 4.14. Adapted from “.NET Architecture” (Kachru and Gehringer 2004)

The .Net framework provides a language independent architecture in comparison to J2EE, which is a platform-independent architecture (Kachru and Gehringer 2004). This language independent nature of the .NET environment allowed the wrapping of original VB legacy code in ASP.NET data services. The .NET framework is not dependent on one specific programming language. This language integration is achieved through the common API that is provided by the .NET framework. Compilers in the .NET framework compile source code from all supported languages into an intermediate format. This black-box approach of developing the services leaves all data connections to the database in their original format, thus retaining the existing system functionality. This differs to the white-box modernization approach, where all database connectivity had to be migrated from the existing system code to Java code in the data services.

Similarly to the white-box approach, the data services developed from wrapping existing system functionality were as a result of component-based analysis of the legacy system (Table 4.4). The Login data service performed the data operations necessary for the inputs and outputs of the Home interface (Table 4.3). The interface components were populated with messages retrieved from the database by the GET methods of the Login service. Furthermore, the `CheckStudent` method required for successful login to the system performed similarly to that of the white-box modernized system (Figure 4.10). However, the original Demi system code was called by the data service to perform this functionality.

For each of the rest of the data services, the Java client transmits the requests to the services to retrieve responses required for display purposes on the GUI. The `Application` interface

sends SOAP requests to the Application data service to retrieve the outputs required during interaction. The ChangeDetails interface sends SOAP requests to the Change Details data service to get student information to populate the interface. Once a user has made the necessary changes to their information, the Change Details interface sends the SOAP request to the service to set the user's updated information. Similar requests and responses were developed for the Attendance and Checks services in order for them to convey the correct inputs and outputs required by the system user.

A difference from the white-box modernization approach occurred in the development of the SessionSelection and PrintSession interfaces and the corresponding Session Selection data service. The data service method which retrieves Session information from the database was developed to return a Session object (Figure 4.15).

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Web.Services
Imports System.Runtime.Serialization

<Serializable()> _
Public Class Session
    Private SessionID As Integer
    Private CourseCode As String
    'Other attributes declared here

    Public Sub New()
    End Sub

    Public Property SessID() As Integer
        Get
            Return SessionID
        End Get
        Set(ByVal value As Integer)
            SessionID = value
        End Set
    End Property

    Public Property CCode() As String
        Get
            Return CourseCode
        End Get
        Set(ByVal value As String)
            CourseCode = value
        End Set
    End Property
    'Other properties implemented here
End Class
```

Figure 4.15. Excerpt from Session Class

A `Session` object is an instantiation of the `Session` class created in the legacy system code. The `Session` class is written in the original VB code and contains properties for the necessary attributes of a `Session` object (Figure 4.15). The `Session` class imports the `System.Runtime.Serialization` namespace to allow the serialization and deserialization of objects. Serialization of an object is the process of converting the object into a format that is compatible for transfer over a network (Strawmyer 2003). There is more than one format that the .NET framework provides for serialized objects, but the namespace imported in this case serializes objects into a SOAP format. The transmission of this serialized object as a SOAP response allows the client application to access all the object's properties (Figure 4.15). Thus, the Java client application that receives the `Session` object as a SOAP response from a service request can access the specific `Session` attributes it requires.

The benefit of implementing the data services and the client-side application using the serialization of objects achieves interoperability between the J2EE and .NET frameworks. The interoperability achieved is a result of using SOAP as the message transfer protocol. The development of web services in this manner thus proves that interoperability is achievable through the successful communication of the client application developed in one environment and the services developed in a different environment.

4.5. Conclusion

The Demi System provides students at the Department of Computing Sciences at NMMU with an automated system to apply to become a student assistant and if their application is successful, the student assistant can use the system for a variety of functions (Section 4.2). These functions include session selection and removal, session attendance, printing lists of selected sessions and calculating total hours worked over a period of time.

The Demi System was selected as an appropriate candidate for modernization of data services due to the fact that the language in which it was developed can no longer be supported in the department (Section 4.2). The Demi System therefore became the subject of two modernization approaches, namely white-box and black-box. The modernization approaches were applied to the Demi System in order to create data services which conformed to the

requirements of the SOA (Section 3.5). The provision of these data services required component-based analysis of the existing system in both cases of modernization approaches.

The white-box modernization approach saw a further, more detailed investigation into the original Demi System known as program understanding (Section 4.3.1). This analysis provided the developer with the knowledge of system functionality and complexities for the code migration phase. The code migration phase enabled the modernization of the existing Demi System code into a new architecture with a modernized interface, yet still providing users with all the same functionality and look and feel as the original system (Section 4.3.2).

The black-box modernization approach included the process of examining inputs and outputs of the original system during user interaction (Section 4.4.1). This analysis of the user's interaction with the system resulted in the discovery of distinct data components. These data components were then individually wrapped in web services as distinct data services (Section 4.4.2). As this investigation only required the modernization of data services, any original system functionality not related to the data services was migrated to the modernized system language (Section 4.4.1). The component-based discovery of data services and the web-service wrapping of these identified components was effectively implemented, as identified in similar studies (Section 2.6).

The services developed by the modernization approaches were not only developed to satisfy structural requirements (Section 3.6), but will have to be evaluated in terms of quality metrics for data services, effort required by the developer to modernize the services as well as the effectiveness of the data services. Thus usability evaluation will be necessary to determine the effectiveness of the modernized services. These measures for evaluation contribute towards a comprehensive evaluation framework for modernization.

Chapter 5: Comprehensive Evaluation Framework

5.1. Introduction

The application of the white-box and black-box modernization approaches to the case study has produced two modernized systems using web services. These modernized systems operate within an SOA where they provide data as a service. The application of a comprehensive evaluation framework to the comparison of these two modernization approaches can determine their equivalence. A holistic evaluation strategy based on multiple facets is proposed in this investigation as the necessary instrument for the determination of modernization success (Section 5.2). The comprehensive evaluation consists of three components, each based on separate related studies.

The first branch will cover the evaluation of QoS metrics of the data services generated from the modernization (Section 5.2.1). The second branch will evaluate the effort required by the developer to modernize the system using a specific modernization approach (Section 5.2.2). The third and last branch evaluates the effectiveness of the generated services based on empirical studies (Section 5.2.3). The results of a pilot study of this evaluation framework are presented to motivate its relevance (Section 5.2.4).

A multi-faceted evaluation framework for the modernization of systems is required to be able to evaluate the design, process and outputs of a modernization approach in terms of its

success. The combination of three evaluation components that complement one another is necessary to create this framework and address the research question (Section 1.5). The application of the combination of the three evaluation strategies presented contributes to a deeper understanding of the success of the modernization approach being studied.

5.2. Comprehensive Evaluation Framework

This investigation requires evaluation of the white-box and black-box modernization approaches applied to the existing system to determine their suitability for the development of data services. Data services that conform to SOA requirements were a result of legacy system modernization efforts. It is necessary to evaluate the modernization approaches, as well as the services generated by each of these approaches.

It is advisable to use a variety of metrics when comparing a software product to another (Tullis and Albert 2008). A framework for the evaluation of the foundation, process and outputs of legacy system reverse engineering has been proposed (Chiang *et al.* 1997). This framework was developed to ensure the good design of a reverse engineering method as well as validate the process of and outputs of the reverse engineering approach. This framework consists of eight criteria related to reverse engineering methodologies. The relevance of this framework, however, is the ability to ensure good design of the reverse engineering process as well as ensuring quality outputs from this process. The outputs of the database reverse engineering are measured through empirical studies.

The development of a comprehensive evaluation strategy based on three distinct evaluation legs aims to holistically compare the modernization approaches and their outputs. The three legs of evaluation are:

- Effort required by the developer to modernize the system (Section 5.2.1);
- Quality of Service (QoS) of the data services generated (Section 5.2.2); and
- Effectiveness of the data services measured through empirical evaluation (Section 5.2.3).

Related studies record the benefits of each of these branches of evaluation. The combination of these studies forms the comprehensive evaluation framework which will be used to compare the two modernization approaches. These measures have been selected to be combined to provide a framework where the three evaluation strategies complement one

another. Each of the evaluation branches is distinct from the other; therefore no overlap of analysis exists. The results obtained from a comprehensive evaluation utilising this framework could provide a more detailed understanding of the success of the modernization approach being evaluated. A pilot study investigation of the application of the comprehensive evaluation framework is necessary to determine its validity (Section 5.3).

5.2.1. Software Metrics and Developer Effort

A suite of metrics for the measurement of complexity in object-oriented (OO) code is proposed (Chidamber and Kemerer 1994). The metrics are based on measurement theory and are presented as six distinct metrics created specifically for measuring components contributing to the size and complexity of OO design (Table 5.1).

Metric	Explanation
Weighted Methods per Class (WMC)	The weighted summation of the number of methods per class
Depth of Inheritance Tree (DIT)	The height of the class in the inheritance tree of class hierarchy
Number of Children (NOC)	Number of immediate subclasses of a class in the class hierarchy
Response For a Class (RFC)	The set of all methods that can be invoked as a result of a message to the object
Lack of Cohesion in Methods (LCOM)	The number of disjoint sets formed by the intersection of sets of instance variables for methods pertaining to a class
Coupling Between Objects (CBO)	The number of non-inheritance related couples with other classes

Table 5.1. The Candidate Metrics

The candidate metrics were developed to be independent of any programming language (Chidamber and Kemerer 1991). The metrics were developed based on three principles concerning the design of classes, namely:

- Definition of objects;
- Attributes of objects; and
- Communication between objects.

A useful way to understand the development of object-oriented systems is to measure the complexities of the different objects. The metrics proposed were collected using automated tools specifically for their research (Chidamber and Kemerer 1994). It is reported that this suite of OO software metrics were well used after their development (Concas *et al.* 2010).

The number of methods per class in an OO system, as well as the complexity of these methods (WMC), is an indication of the amount of time and effort required by the developer to create and maintain the class (Table 5.1). If a parent class contains a larger number of methods, a larger impact is made on the subclasses (Chidamber and Kemerer 1994). This is due to the fact that they inherit all methods in the parent class. It is also believed that the larger the number of methods, the more likely the class is application specific. This could negatively impact the reusability of the class. The equation for the summation of m methods for this WMC metric is given by equation (5.1).

$$f(m) = \sum_{i=1}^m c_i \quad (5.1)$$

The weights given to the methods (c_i) for this investigation are determined by cyclomatic complexity analysis of the method. The cyclomatic complexity for each method is calculated by finding the number of independent paths through the method. This measure is based on graph theory and was developed by Thomas McCabe in the 1970's (McCabe 1976). The equation used for cyclomatic complexity is given by equation (5.2).

$$V(G) = e - n + 2 \quad (5.2)$$

In the calculation of $V(G)$, e is the number of edges of the graph G and n is the number of nodes of G . If G is a directed graph representing the structure of the method, then each node in G is a block of statements where the flow is sequential, and each directed edge, or arc, represents a branch taken by the method (Nath 2009). The arc represents the transition from one block of statements to another, thus forming the directed graph (Gupta 2004). From the example depicted (Figure 5.1), the calculation of the cyclomatic complexity can be computed as 2 using equation (5.2).

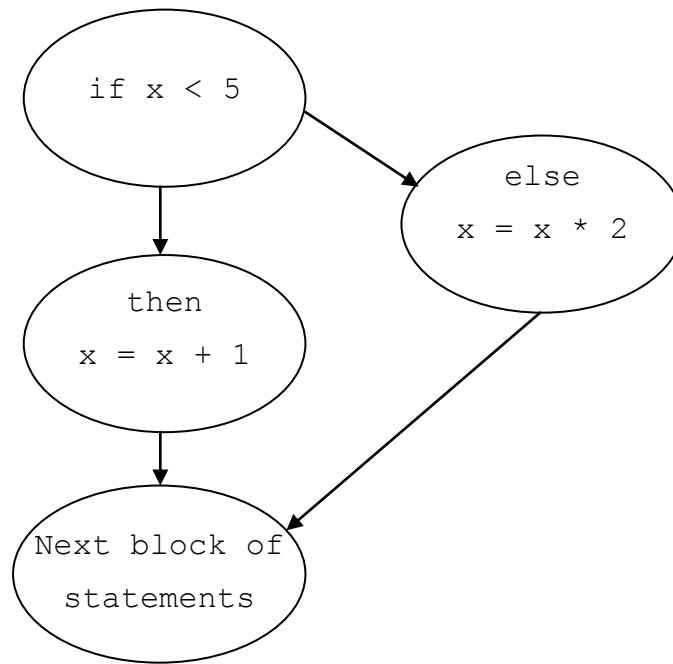


Figure 5.1. Directed Graph of Cyclomatic Complexity

The deeper a class is in a hierarchy of classes (DIT), the greater the number of methods it inherits from its hierarchy (Table 5.1). This makes behaviour of the class more complicated to predict (Chidamber and Kemerer 1994). This greater tree depth implies larger design complexity as more methods and classes exist. However, the deeper the class is in the hierarchy, the greater the ability to reuse the inherited methods.

The larger the number of subclasses (NOC), the greater the reuse of methods from the parent class, due to inheritance (Table 5.1). Incorrect parent class abstraction is another possible implication of a large number of subclasses (Chidamber and Kemerer 1994). This could increase the chances of misusing a subclass. The number of subclasses that a class has could show the possible influence of that class on the design of the system. This means that if a class has many subclasses, there may be more need for testing each of the methods of that class.

The set of methods of a class that could possibly be invoked in response to a request received by the object of the class is known as RFC (Table 5.1). There is also a measure of the communication that is likely to occur between the class and other classes (Chidamber and Kemerer 1994). If the number of methods that are invoked due to a response is large, then the testing of the class becomes more complex due to the need for a deeper understanding of the

code. The class itself is more complex if it contains a greater number of methods that could be invoked.

The count of pairs of methods where the similarity of these methods is zero minus the count of method pairs whose similarity is not zero describes the LCOM (Table 5.1). The greater the number of similar methods, the more cohesive the class is (Chidamber and Kemerer 1994). This confirms the measure of inter-relatedness between elements of a program. The cohesiveness of the methods in a class is a positive aspect and it implies good OO design, such as encapsulation. This measure could thus help to identify flaws in the design of the classes. The equation for calculating LCOM is given as equation (5.3) (Gupta 2004).

$$\text{LCOM} = \begin{cases} P - Q, & \text{if } P > Q \\ 0 & \text{otherwise} \end{cases} \quad (5.3)$$

To calculate the LCOM for a class, consider each method pair in the class and if they access distinct sets of attributes then increase P by 1. Otherwise, if they share at least one attribute then increase Q by 1. A zero value indicates that the class is cohesive and well designed (Gupta 2004). An LCOM value of greater than zero, however, implies that the class may need to be redesigned as two separate classes.

The idea of an object being coupled to another object because it acts upon the other object describes the CBO (Table 5.1). More specifically, an object's method that uses the methods or variables of the other object describes this coupling. The extensive use of coupling objects negatively impacts the reuse and modular design of a class (Chidamber and Kemerer 1994). The more independent a class is, the higher the possibility for reuse of that class in another application. Coupling should be kept to a minimum to improve modularity as well as encapsulation. This could also improve the maintainability of the code, since less change to other parts of the design is necessary due to the reduced coupling of object classes. This measure of CBO can imply the complexity of testing required for the various parts of the OO design.

For the purposes of this study, the software metrics describing the weighted methods per class (WMC) will provide a useful measure to determine effort required by the developer to create the data services from the legacy code. The DIT and NOC metrics will prove useful in

the verification of complexity of code created by the developer. CBO metrics collected will also prove useful in determining the reusability of classes of code in the data services generated (Section 5.2.2), but not as a measure for developer effort. The other metrics presented, namely LCOM and RFC will not be necessary for this study and will not be used in the evaluation of the data service generated by the two modernization approaches.

5.2.2. Performance Metrics and QoS

Seven quality attributes need to be adhered to when generating services for a SOA (Bianco *et al.* 2007). QoS metrics are essential in selecting the best possible service development execution plan with regard to budget and time constraints (Jeong *et al.* 2009). These QoS attributes are similar and can be related to one another (Table 5.2).

Seven Quality Attributes (Bianco <i>et al.</i> 2007)	Major Quality Attributes (Jeong <i>et al.</i> 2009)
Performance	Performance Operation Cost
Availability	Availability Accessibility
Security	Security
Testability	-
Interoperability	Interoperability
Modifiability	-
Reliability	Reliability

Table 5.2. Quality of Service Attributes

Performance is measured in terms of time taken for responses to be returned after a request to the service (Bianco *et al.* 2007). This measure of latency is indicative of the operation cost of invoking the service (Jeong *et al.* 2009). Availability is defined as a service’s ability to be available despite server maintenance or system overload (Bianco *et al.* 2007). The availability of a service thus determines its accessibility at any point in time (Jeong *et al.* 2009).

Security is defined in terms of confidentiality and protection of the data being accessed by a service. Services are also required to be testable for new or updated versions (Bianco *et al.* 2007). The service is required to be interoperable with various platforms and other services that it interfaces with. The service’s modifiability is measured with respect to its reusability.

The reliability of a service is defined in terms of the amount of system errors and the recovery of the system as a result of these errors (Jeong *et al.* 2009).

Data security and service reusability are emphasised for the development of data services (Borkar *et al.* 2006). Access to the data services must be controlled, as well as query processing security. The reuse of data services is an important characteristic to adhere to and is achieved by encapsulation of method call details. Latency, the time taken to achieve results after submitting a query, is also emphasised as a quality attribute that is required for data services. Improved latency leads to improved performance.

For the purposes of this investigation, a consolidated list of QoS attributes needs to be identified to measure data services specifically. The comprehensive evaluation framework will thus include specific quality of data service metrics on which to measure the outputs of a modernization approach. The QoS metrics that are emphasised and commonly used for data services include:

1. Performance;
2. Interoperability;
3. Security;
4. Reliability; and
5. Modifiability/Reusability.

Testability has been excluded from the consolidated list of metrics as it is reported as being a minor quality attribute (Jeong *et al.* 2009). Availability and accessibility are reported as a result of ensuring the security of data services (Borkar *et al.* 2006), thus only security is considered for this investigation. Operation cost is implicated by the measurement of performance of a service. The five QoS metrics listed will form the basis of the evaluation on quality of the data services generated by the white-box and black-box modernization approaches.

5.2.3. Effectiveness of Data Services

Numerous design and quality guidelines for the development of services have been suggested to assist the developer in the modernization of legacy systems. After these services have been deployed, the user requires an effective and efficient means of carrying out the functionality provided by these services. Effectiveness is defined as a general goal as to how well a product performs the task that it is supposed to (Sharp *et al.* 2007). Empirical research will be

conducted in the form of a usability evaluation to determine the effectiveness of the data services generated by the modernization approaches. Usability testing is defined as the evaluation approach that involves measuring users' performance and analyzing their satisfaction with the system under evaluation in a formal controlled environment (Tullis and Albert 2008). Usability testing is conducted in real-world laboratory conditions under the supervision of the evaluator who collects data from the evaluation (Miller and Jeffries 1992).

It is stated that empirical research aims to explore, describe and forecast natural, social or cognitive occurrences using data collected during observation or experimentation (Sjøberg *et al.* 2007). Approaches to empirical studies include both qualitative and quantitative methods. Quantitative methods are explained as collecting numerical data and analysing it by statistical procedures. On the other hand, qualitative data is collected as text or images from observations, interviews or questionnaires (Sjøberg *et al.* 2007). Lastly, it is explained that there are also design components that may be used to reinforce the empirical research with regard to reducing the occurrence of internal threats to validity.

Empirical evaluations are used to support the comparative analysis between an existing legacy system modernization tool and MELIS, the legacy system migration tool (Colosimo *et al.* 2008). The hypothesis of the evaluation stated that MELIS significantly reduced the effort required to migrate legacy code in comparison to existing tools. The experiment was designed to allow two groups of users to test the new tool, MELIS, and the original modernization tool. The experience of the users was under investigation in the hopes that the new modernization tool could bridge the gap between software engineers with differing experience levels. Counterbalancing was used to remove bias of group ability.

Analysis of the data collected included a two-way Analysis of Variance (ANOVA) test to validate the effects of other variables on a dependent variable (Colosimo *et al.*, 2008). The dependent variable included both effort required to perform the migration tasks and effort required to understand the original program. The ANOVA tests revealed that the effort required to perform the migration tasks was significantly reduced. The Multifactorial ANOVA test has also been used to analyse the relationship between time taken to complete a task and other experimental variables (Turpin and Scholer 2006).

5.3. Validation of Framework

To validate the comprehensive evaluation framework designed for this investigation, a pilot study (Appendix A¹) was conducted (Barnes and Cilliers 2010). The white-box modernization approach was applied to a subsystem of the Demi System (Section 4.2) as a proof of concept. The validation of the evaluation framework was conducted using the identified three legs of evaluation. The experiment was designed to allow seven participants to perform three tasks on the modernized components of the Demi System. The participants gathered for this evaluation were all required to be existing student assistants so that they would have access to the system as well as have selected sessions. There were one female and six male participants for this study. The developer effort and QoS metrics, with the exception of performance metrics, were gathered separately from the user study metrics.

The QoS evaluation of the data services generated was derived from the QoS metrics identified (Section 5.2.2). The QoS evaluation was conducted using four metrics, namely:

- Performance;
- Reliability;
- Interoperability; and
- Reusability.

Performance metrics were gathered for each web service method called during use with the system. Latency of the service calls were measured as the amount of time the service took to send a response to a service request. The latency was obtained by using the current system time in milliseconds at the time of the service request to determine the start time. The elapsed time was then calculated as the start time subtracted from the current system time in milliseconds after the response from the service. The measurement of performance in this way has been applied in similar experimental studies (Mulligan and Gračanin 2009).

The reliability of the data for the Demi system could not be measured quantitatively, but was catered for in the design of the system. Validation of data entered into the interfaces occurred on the client-side of the application ensuring that only valid data was entered into the database by the data services. The retrieval of data from the database made use of methods from the Checks data service (Section 4.3.3) to ensure that only reliable data would be displayed.

¹ Published in the proceedings of the South African Telecommunications Network and Applications Conference (SATNAC), 2010

Interoperability was another design consideration during the development of the data services and the client application. Interoperability through platform independence was ensured through the use of the J2EE framework for the development of the modernized Demi System (Section 4.3.3). The use of SOAP requests and responses for communication with the data services also ensured the interoperability of the modernized system. Reusability of these web services on any platform is a result of the interoperability of the system. Software metrics were also used to measure the reusability of the code in the web service client (Barnes and Cilliers 2010). Low reusability was measured in the client-side classes due to the migration of data operations to data services. The lack of reusability was implicated by the values calculated for the DIT and NOC metrics (Section 5.2.1). The other consequence of low DIT and NOC measures, however, is the reduced complexity and increase in prediction of behaviour of the class.

With regard to the measure of developer effort in modernizing the existing system code, complexity of the code is a serious consideration. The low complexity of the classes implied by the DIT and NOC metrics are not the only means to calculate complexity of code. The WMC measure provides a more detailed analysis of methods written by the developer. Each method is calculated in terms of its cyclomatic complexity. The sum of these complexities of all the methods for a class reveals the WMC (Section 5.2.1). The developer effort evaluation revealed that the `Home`, `Details` and `PrintHours` classes (Section 4.3.3) showed the highest values for WMC. The indication is therefore that these classes required more effort to develop than the other classes. Due to the white-box approach used for this pilot study, both the client-side application as well as the data services required code migration (Section 4.3.2). The implication of this approach is that more effort was required by the developer to create the data services as well as migration of the client-side functionality.

For the effectiveness of the data services branch of the evaluation framework, both quantitative and qualitative data was gathered during user tests (Section 5.2.3). The seven participants performed three tasks on the system in a lab environment under observation by the evaluator (Barnes and Cilliers 2010). Several metrics were collected during user testing, namely:

- Time taken per task;
- Task success;

- Task completion;
- Error rates per task; and
- Self-reported metrics.

The time on task, task success, task completion and error rates metrics measure performance of the system (Tullis and Albert 2008). The satisfaction of the user is gauged by the collection of self-reported metrics from instruments such as a questionnaire.

The performance metrics collected during the pilot study revealed that all participants ($n = 7$) were able to complete all three tasks (Appendix B). This positive result for task completion indicates the ability of the services to effectively perform the tasks required by the users. Furthermore, the task success results yielded that all seven participants completed all three tasks successfully. High error rates were observed for the third task despite the completion and success rates. The cause of the high error rates was identified as a navigation issue and a recommendation for improved ease of navigation was made.

The times taken to complete each task were recorded for all seven participants. The nature of the second task resulted in a lower mean time. The second task involved the users printing their list of selected sessions. No input was required from the users for this task, thus the results could be retrieved from the system quickly. The first task, which requested users to update their personal information, obtained a higher mean time than the second task. The third task, which requested users to print their total hours worked over a specified date range, also resulted in a higher mean time than the second task. Both the first and third tasks required participants to enter information into the interfaces, thus resulting in higher mean task times than the second task. The data services were able to effectively provide the results that the users required in consistently low times for all tasks.

Lastly, the self-reported metrics to determine user satisfaction of the system were collected by means of a recognised instrument, namely the System Usability Scale (Tullis and Albert 2008). The System Usability Scale (SUS) is a post-session questionnaire consisting of ten statements, each with a 5-point Likert scale of agreement. The odd numbered statements are worded positively whilst the even numbered statements are worded negatively. This instrument has shown more consistent ratings at relatively small sample sizes (Tullis and Stetson 2004). Due to the alternating positive and negative statements, normalization of the

score for each statement of this questionnaire is necessary to sum the scores and achieve an overall result. Consistently high SUS scores were observed for all seven participants with a mean score of approximately 87% (Barnes and Cilliers 2010).

For this investigation, the QoS standards were catered for during design of the data services, as well as during use of the modernized system. Software metrics used to determine developer effort required to modernize the system showed that significant effort was required by the developer to migrate existing system code to a new architecture (Barnes and Cilliers 2010). The results of the modernization approach were then evaluated by means of user studies. The data services generated through modernization performed their tasks effectively. Usability issues were identified during this evaluation and recommendations were made. Furthermore, user satisfaction with the modernized system was observed.

If only the evaluation of developer's effort were applied to the modernization of this system, the results would indicate that the modernization approach was effort intensive. A high developer effort result could indicate that the modernization approach may not be favourable. The developer's effort evaluation was combined with the user evaluation, however, where positive feedback in terms of user satisfaction and performance was achieved. Furthermore, the adherence to QoS guidelines for the development of data services resulted in successful development of the modernized system. After consideration of all results obtained from evaluation, the benefits of the applied modernization approach outweigh the negatives, thus indicating successful modernization.

5.4. Conclusion

The development of a comprehensive evaluation framework has been presented to evaluate the success of a modernization approach (Section 5.2). The framework consists of three distinct evaluation strategies. The evaluation strategies are not weighted in the framework, but complement one another in the determination of modernization success.

The first evaluation strategy contributing to the evaluation framework is developer effort to modernize a system (Section 5.2.1). Software metrics such as WMC, NOC and DIT can be used to describe complexity of code, and thus indicate the time and effort required by the developer to create a class of code. The second evaluation strategy that the evaluation

framework consists of is Quality of Services generated by modernization (Section 5.2.2). Various quality attributes of web services exist and those pertaining to data services in particular should be adhered to in the generation of data services through modernization. The third and final evaluation strategy branch of the comprehensive framework involves user evaluation to the modernized system (Section 5.2.3). Performance and user satisfaction results can be obtained from user evaluation to measure effectiveness of the data services generated from modernization.

The outcome of the validation of the comprehensive evaluation framework shows how the combination of differing evaluation strategies is beneficial to judging modernization success (Section 5.3). The evaluation framework was applied to the white-box modernization of a small subsystem of the existing Demi System. This pilot study was conducted to validate whether the comprehensive evaluation framework would be beneficial in the evaluation of success of a modernization approach. Results from the validation showed that when only one evaluation strategy is applied a single, perhaps misleading, result is obtained. When results from the three evaluation strategies of the comprehensive framework were combined, a clearer understanding of whether the modernization was a success was achieved. This framework will be applied to the comparison of the two different modernization approaches used for the Demi System case study to determine which approach, if any, is more suitable for the generation of data services.

Chapter 6: Analysis of Results

6.1. Introduction

A comprehensive evaluation framework consisting of three distinct evaluation strategies has been presented to determine the success of a modernization approach. The application of this evaluation framework to the white-box and black-box modernization approaches will form the comparative analysis of the two approaches.

Experiments must be designed for each of the three evaluation strategies in the comprehensive evaluation framework (Section 6.2). The experiment to measure developer effort is designed using software metrics (Section 6.2.1). The experiment to measure QoS of the data services generated from modernization is designed using the QoS guidelines for data services (Section 6.2.2). The experimental design for effectiveness of the modernized system is presented in terms of empirical studies (Section 6.2.3). The results of each of these experiments combined intend to determine which of the white-box or black-box modernization approaches is more suitable for the provision of data services from modernization (Section 6.3). The application of each of these evaluation components to evaluate the case study modernization in terms of the developer's effort, the QoS and the effectiveness of the data services addresses its own research question on how this evaluation is conducted (Section 1.5).

6.2. Experimentation Design

The metrics chosen for a comparative analysis are product dependent. The development of the comprehensive evaluation framework combines three distinct metrics to evaluate the design, process and outputs of a modernization approach. The experiments for each of the three branches of the evaluation framework must be designed to suit the gathering and analysis of data regarding the metrics.

The application of software metrics to the modernized code will provide an indication as to how much effort was required by the developer to modernize the existing system (Section 6.3.1). These software metrics as well as other design guidelines must be adhered to for the production of high quality data services (Section 6.3.2). Empirical studies can provide insight into user satisfaction and effectiveness of the data services generated by the modernization approaches (Section 6.3.3). The empirical study will follow the guidelines dictated for user testing.

6.2.1. Developer Effort Evaluation

Developer effort can be measured quantitatively by the calculation of certain code complexity metrics with regard to the code written by the developer (Chidamber and Kemerer 1991). The specific metrics useful in the investigation to compare the white-box modernization approach to the black-box approach are as follows:

- Weighted Methods per Class (WMC);
- Depth in Inheritance Tree (DIT); and
- Number of Children (NOC).

Each of these metrics is collected by analysis of the classes of code developed. For the purposes of the comparative investigation these metrics will be calculated on all the modernized code that the developer was required to migrate.

Thus, in the case of the white-box modernization approach all client-side functionality classes (Section 4.3.2) will be analysed as well as all the data service classes (Section 4.3.3). In terms of the black-box modernization approach all the classes modernized during user interface modernization (Section 4.4.1) will be analysed. The data service classes will not be analysed for the black-box modernization as the methods used in this case contain the original system code, according to the black-box modernization approach (Section 4.4.2).

The WMC metric will be calculated for each class by summing the cyclomatic complexities of all the methods in the class (Equation 5.1). The cyclomatic complexity can be calculated for each method by identifying the number of independent paths in the method (Equation 5.2). The DIT and NOC metrics are both calculated by the analysis of class hierarchy. The DIT metric is calculated as the depth of the class in the inheritance tree, whereas the NOC metric is calculated by summation of the number of children that a class has.

The results for the three metrics presented here will be compared to one another for the white-box and black-box approaches to determine which approach required less effort by the developer. After the complexity and effort measurements of the modernized code, it is necessary to determine the quality of the services generated during both modernization approaches.

6.2.2. Quality of Service Evaluation

The data services generated by both modernization approaches need to be evaluated in terms of the quality requirements identified for data services (Section 5.2.2). The resulting comparison in these quality metrics will be used to determine if one of the modernization approaches produces higher quality data services than the other. The QoS metrics that need to be adhered to are:

- Performance;
- Interoperability;
- Security;
- Reliability; and
- Modifiability/Reusability.

Not all of these characteristics can be measured quantitatively. In the case of the Security, Reliability and interoperability metrics, guidelines are met for the development of services that adhere to these characteristics. Security and Reliability are characteristics of SOAP services which is a reason why they were chosen for implementation of the data services (Chapter 3).

Interoperability was considered in the design of the modernized applications. The white-box modernization approach made use of the J2EE framework which is platform-independent, thus promoting interoperability of the services (Section 4.3.3). The black-box modernization

approach used a hybrid of the J2EE and .NET frameworks for the provision of data services (Section 4.4.2). The ability of the Java client to communicate with the .NET web services is a result of the protocols that the SOAP-style data services adhere to (Chapter 3). The black-box services therefore exhibit interoperability through this protocol-enabled communication between the differing client and web service environments.

The measurement of the performance of the data services can be conducted by using an approach identified in similar experimental studies (Section 5.3). The difference in the current system time before the service request is sent and after the response is retrieved can determine latency (Mulligan and Gračanin 2009). The values for the elapsed time can be calculated in milliseconds and recorded in a log file during runtime of the modernized system. The captured performance data for each service method can then be compared for the services generated from the different modernization approaches. The comparison of mean latencies can determine whether the following null hypothesis (Section 1.5) can be rejected:

H_{0.1.1}: Mean service method latencies for black-box and white-box modernized data services are not equivalent

H_{1.1.1}: Mean service method latencies for black-box and white-box modernized data services are equivalent

Reusability can be measured by analysis using software metrics. The CBO metric provides a good indication of code modularity and reusability (Chidamber and Kemerer 1994). The CBO measurement will be determined for each of the data services created from the white-box and black-box modernization approaches. The CBO metric is a measure of the relationship between classes. This metric is calculated for a class A by counting the number of classes that A references as well as the number of classes that reference A. If, however another class, B, is referenced by A and A references B then B is only counted once. If differences exist in these results, it could indicate that the data services of one approach are more reusable than the other.

6.2.3. Effectiveness of Data Services Evaluation

The effectiveness of the data services generated by the two modernization approaches will be determined through user testing. The user evaluations will be conducted in a formal, controlled lab environment under the supervision of an evaluator (Section 5.2.3). This experiment has been designed to gather performance and self-reported metrics to determine

the effectiveness of the data services as well as user satisfaction with the modernized systems. Each modernized system needs to be evaluated, thus allowing the design of the system to be the independent variable during analysis of the results.

The experiment will be conducted using a within-subjects design (Tullis and Albert 2008). The within-subjects design implies that each participant will evaluate both modernized systems. To reduce bias in the data gathered, counterbalancing will ensure that the order in which the systems are used will be alternated between the users. Specifically, half of the participants will first test the system generated by the white-box modernization approach followed by the black-box system and in reverse order for the other half of the participants.

The participants for this experiment are all required to have had experience using the original Demi System. The sample population, despite being a sample of convenience, must be representative of the actual population. Therefore, both male and female students from a variety of ethnic groups must be selected. Furthermore, both undergraduate and postgraduate student assistants must be included in the sample.

Background information such as system experience and biographical information will be collected to understand the demographics of the sample population. The comparative evaluation will consist of five tasks to be completed on each modernized system (Appendix C). The five tasks include:

- Application to be a student assistant using a provided user persona;
- Selection of sessions to assist;
- Removal of a session;
- Viewing the current list of selected sessions after additions and removals; and
- Updating personal information on the system.

These five tasks will be repeated with slightly different information for the modernized system second in order.

The NMMU Department of Computing Sciences has a formal usability lab fitted with observation and eye-tracking hardware. The participants will evaluate the two modernized systems in this lab so that usability data can be gathered throughout the evaluation. The performance metrics that will be gathered include:

- Time on task;
- Task completion;
- Task success; and
- Error rates.

The times recorded per task are captured by the eye-tracking software on the usability lab machine. Screen recordings of all of the participants' actions are performed by the eye-tracking software. The task completion, success and error rates are gathered by the evaluator's observation of these screen recordings.

To determine a participant's satisfaction with a software product, self-reported metrics need to be collected (Tullis and Albert 2008). The participants will complete the System Usability Scale (SUS) questionnaire after the use of each modernized system (Appendix C). The SUS will therefore be used as a post-task questionnaire.

After both modernized systems have been evaluated by the participant and each system's SUS questionnaire has been completed, a modified SUS post-test questionnaire will be given to the participant (Appendix D). This modified instrument consists of the same ten statements as the original instrument. The difference, however, is the alteration of the agreement scale. The purpose of this post-test SUS is to ask the participants to compare the first system to the second system. The results of these post-task and post-test questionnaires can be normalised and compared to one another for both the white-box and black-box modernized systems. Combining the results from all three evaluation components will suggest the most suitable modernization approach for the generation of data services in this case study.

Descriptive statistics will be applied to the biographical information gathered from the participants to describe the sample population. The data collected during the usability evaluation will need to be compared for the two modernized systems. Comparative statistics such as the Analysis of Variance (ANOVA) test will be used to validate the null hypothesis that the effectiveness of the modernized data services are equivalent (Devore and Farnum 2005).

To compare the two modernization approaches in terms of task completion times during use of the modernized systems, the ANOVA test will aid in testing a null hypothesis of the form:

H_{0,2,1}: Mean task completion times for black-box and white-box modernized data services are not equivalent.

H_{1,2,1}: Mean task completion times for black-box and white-box modernized data services are equivalent.

Similarly, when comparing the number of errors observed during use of the white-box modernized and black-box modernized data services the null hypothesis will take the form:

H_{0,2,2}: Mean error rates for black-box and white-box modernized data services are not equivalent.

H_{1,2,2}: Mean error rates for black-box and white-box modernized data services are equivalent.

Lastly, the scores obtained from the SUS questionnaires will be compared with regard to the following null hypothesis:

H_{0,2,3}: Mean SUS scores for black-box and white-box modernized data services are not equivalent.

H_{1,2,3}: Mean SUS scores for black-box and white-box modernized data services are equivalent.

ANOVA will be used to compare all performance and user satisfaction metrics collected during the user evaluation. The results of each of these hypotheses will be analysed in order to determine whether the following null hypothesis (Section 1.5) can be rejected:

H_{0,2}: Effectiveness of black-box and white-box modernized data services is not equivalent

H_{1,2}: Effectiveness of black-box and white-box modernized data services is equivalent

6.3. Results

The analysis of the data gathered during the application of the comprehensive evaluation framework to the modernized Demi systems is necessary to formulate results. The framework was applied in a comparative analysis of the two modernization approaches to determine which approach is more suitable for the generation of data services in the case of the Demi System.

The developer effort results are discussed after analysis by the use of software metrics on the modernized code of the two resulting systems (Section 6.2.1). The generation of good quality data services is analysed for each modernization approach (Section 6.2.2). The performance of the data services is compared for the two modernized systems using a sample of calls to

each data service method. Lastly, the results of the user studies on each modernized system are compared to one another to determine if there is a significant difference in perception or performance between the two systems (Section 6.2.3). The results of the usability evaluation give rise to the effectiveness of the data services created from modernization.

The combination of results from the three branches of the evaluation framework will be interpreted to determine if one modernization approach is more suitable than the other for modernization of a system to data services. The use of more than one metric to evaluate the comparison of the modernization approaches aims to provide a more holistic understanding of the entire modernization approach and its success (Section 5.2).

6.3.1. Developer Effort Results

Software metrics to measure code complexity were applied to the white-box and black-box modernized systems. The WMC, DIT and NOC metrics were calculated for all client and service classes for the white-box modernized code. The WMC metrics for the client classes of the white-box and black-box modernized systems may be compared (Figure 6.1).

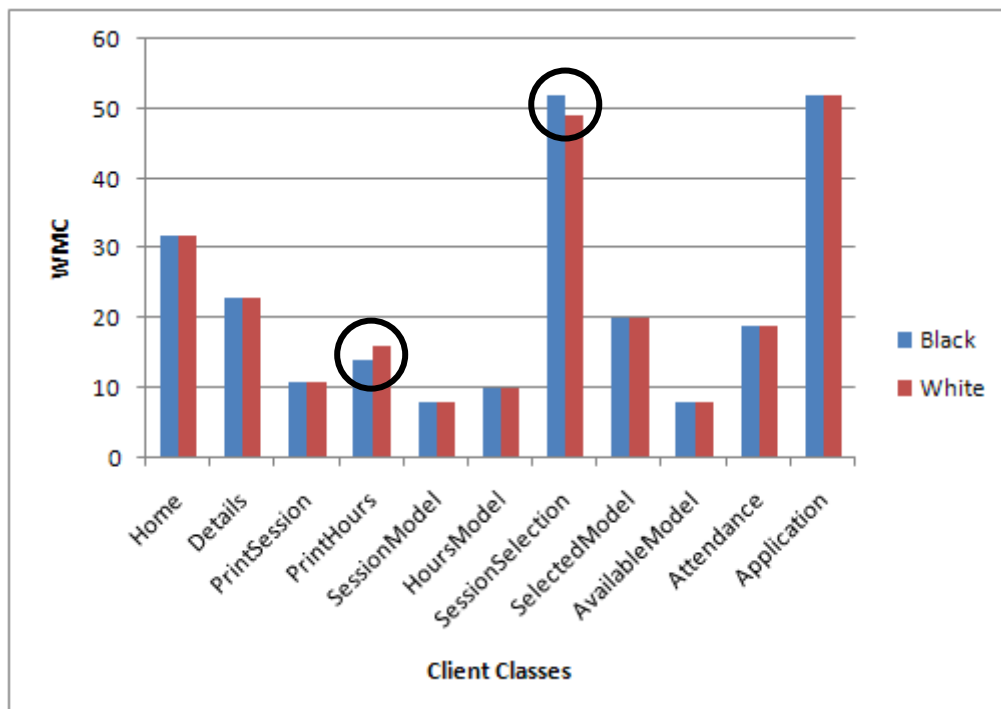


Figure 6.1. WMC Comparison for White-Box and Black-Box Modernization

The implementations of the white-box and black-box client classes differed slightly. The number of methods for the client classes was also calculated for both the white-box and black-box modernized systems (Figure 6.2). The increase in the WMC metric for the black-box `SessionSelection` class implies that the class required more effort to develop (Figure 6.1). The WMC value was larger due to an extra method required during development of the class (Figure 6.2). The extra method was created to process the data retrieved from the Session Selection service as the data was packaged as a serialized `Session` object in the SOAP response (Section 4.4.2).



Figure 6.2. Comparative Number of Methods of Client Classes

Similarly for the `PrintHours` white-box class, the increase in number of methods initiated an increased WMC value. The code migration phase in the white-box modernization approach applied the migration of code from VB to Java and certain implementation challenges were met (Section 4.3.2). The development of methods to capture the selected date from the Java date picking tool resulted in the increased methods and increased WMC values for the `PrintHours` class in the white-box modernized system. The same functionality was catered for differently in development of the black-box modernized system and this had a positive impact on the developer effort for the `PrintHours` class (Figure 6.2).

The rest of the classes in both the white-box and black-box modernized client-side code showed the same results for number of methods and WMC. The invasive code migration of

the interface functionality was the cause for the similar class structures and complexities in the two modernized systems. The calculation of effort required by the developer to modernize the black-box system by measuring WMC was not necessary for the data service classes. The data service classes for the black-box modernization were simply wrappers for original legacy methods (Section 4.4.2). The migration of existing system code to data services for the white-box modernization approach, however, resulted in the collection of complexity metrics for these classes (Table 6.1).

	Login	Checks	PrintSession	PrintHours	SessionSelection	Attendance	Details	Application
WMC	30	20	12	12	24	42	20	31
#Methods	6	4	2	2	5	6	4	6
DIT	0	0	0	0	0	0	0	0
NOC	0	0	0	0	0	0	0	0

Table 6.1. Comparison of WMC and Number of Methods for White-Box Services

The complexity of the web methods in the white-box modernized data service classes is evident (Table 6.1). The migration of the data operations from original system code to Java code resulted in the development of Java database connectivity code. This migration of data access from the original Demi System to the modernized system increased the effort required by the developer to modernize the system in the white-box approach. The recreation of these data operation methods was not necessary for the black-box modernization approach.

The interpretation of NOC and DIT metrics also provides insight into the design complexity of the classes being analysed. Similarly to the WMC metric, the NOC and DIT metrics are applied to the classes containing modernized code only to show evidence of developer effort required during modernization. The NOC metric analysis of the client-side classes on both the white-box and black-box system revealed that none of the classes had any subclasses. Furthermore, all of the data service classes developed during white-box modernization also yielded values of zero for NOC. The DIT metric values for the white-box and black-box modernized client classes are compared (Figure 6.3).

The Java table model classes, namely `HoursModel`, `SessionModel`, `SelectedModel` and `AvailableModel` showed a DIT value of one for all the white-box and black-box client classes (Figure 6.3). This DIT value is a result of the table model implementing the

abstract Java Table Model interface. The `PrintHours` class implements the abstract Java Dialog interface and thus also has a DIT value of one. All other classes developed during both modernization approaches showed DIT values of zero.

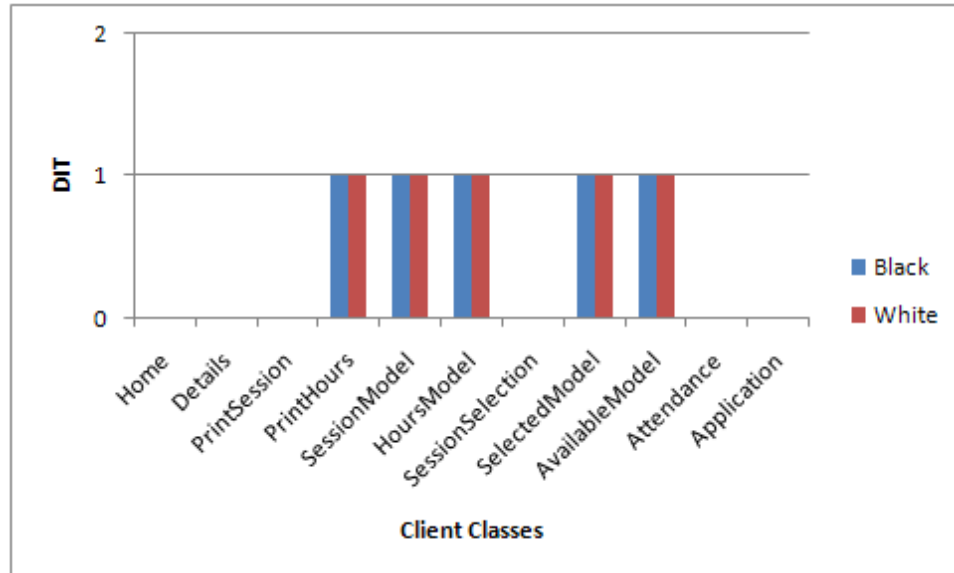


Figure 6.3. DIT Comparison for White-Box and Black-Box Modernization

Low NOC and DIT values indicate reduced complexity of the classes and easier prediction of the behaviour of classes (Chidamber and Kemerer 1994). This indication is beneficial in the argument that modernization to generate data services results in the development of classes whose behaviour is easy to predict. The developer would benefit from this result when testing of code is required. Both the DIT and NOC metrics analysis of the white-box developed data services yielded zero values for all classes (Table 6.1). Thus, the complexity in behaviour prediction of these data services is perceived as being low.

6.3.2. Quality of Service Results

Not only were software metrics used to measure developer effort required to modernize the legacy code, but the CBO metric was a useful measure for code reusability. The CBO analysis was applied to both the white-box and black-box data services to measure the relationships between services. The CBO for the data services developed by the two modernization approaches were then compared (Table 6.2).

The data services developed for the white-box and black-box modernization differed slightly. The white-box data services did not make use of a `Session` class (Table 5.6). The black-

box modernization approach did not lead to the generation of the Print Session data service. The functionality for the Print Session data service was obsolete as the method to retrieve selected sessions was catered for by the Session Selection data service during black-box modernization (Section 4.4.2). In the case of the white-box modernization approach, the data operation to retrieve selected sessions was developed in the Print Session data service (Section 4.3.3).

	Login	Details	PrintHours	SessionSelection	Attendance	Application	Checks	Session	PrintSession
Black	0	0	0	1	1	0	0	2	No service
White	0	0	0	0	0	0	0	No Class	0

Table 6.2. CBO Comparison for White-Box and Black-Box Data Services

The `SessionSelection` and `Attendance` classes showed coupling relationships with the `Session` class in the black-box modernized services. This coupling was due to the existence of the serializable `Session` class used to retrieve session data from the database to send in a serialized format by SOAP response to the black-box service client (Section 4.4.2). The `Session` class itself was not a data service, but an auxiliary class existing in the original Demi System to make data access simpler. As a result, none of the data services couple with one another in either the black-box or white-box data services. These low CBO values promote code modularity and reusability (Chidamber and Kemerer 1994). The data services generated by both modernization approaches have been successful in terms of reusability. This result is encouraging as reusability of the web methods in data services is one of the QoS guidelines to which should be adhered.

The measurement of the data service performance was the next QoS metric to be analysed. The performance was measured in terms of latency. Latency was defined as the difference in system time from the transmission of the service request to the retrieval of the service response (Section 6.2.2). The latency was measured in milliseconds using the `Java System.currentTimeMillis()` method. Each data service method invoked by the modernized client application was analysed in terms of its latency. The same data service methods existed for both white-box and black-box developed services. There were 22 service methods to be analysed. The latencies of the service responses were gathered during use of

the system by 30 participants in the user evaluation (Section 6.2.3). The results of the latencies were recorded in a log file during runtime of the systems.

A network issue was encountered during the first six participants' evaluations. This network problem was a result of a faulty network port in the switch. As a result of this network problem, the recorded latencies for the first six participants had to be omitted from the investigation. These omitted results were identified as clear outliers during data pre-processing. The network issue was resolved and the evaluations continued for the remaining 24 participants. Due to a sample size of less than thirty, the Analysis of Variance (ANOVA) test could not be used to compare the performance results for the white-box and black-box data services. The non-parametric Wilcoxin matched pairs test was used to compare the performance results for the two modernization outputs (Table 6.3).

Method	Valid n	T	Z	p-value	Better Approach
fillNoticeBoard1	23	1.00	4.17	.000**	White
fillNoticeBoard2	12	23.00	1.26	.209	Black
getMessages	20	62.50	1.59	.113	Black
checkStudent	15	27.50	1.85	.065	White
getStudentInfo	14	49.50	0.19	.851	White
checkWorkshop	11	17.50	1.38	.168	Black
readFirstInfo	24	27.00	3.51	.000**	White
readInfo	11	30.00	0.27	.790	Black
readLastInfo	24	2.50	4.21	.000**	Black
getLecturers	16	67.00	0.05	.959	Black
storePersonalDetails	22	36.00	2.94	.003**	White
storeExtraDetails	16	31.50	1.89	.059	White
getSelectedSessions	24	19.00	3.74	.000**	White
getSelectedColumns	21	62.50	1.84	.065	Black
getAvailableDetails	24	0.00	4.29	.000**	Black
getAvailableColumns	23	24.00	3.47	.001**	Black
getDemis	23	22.50	3.51	.000**	White
addSession	20	48.00	2.13	.033*	Black
insertIntoChanges	23	82.00	1.70	.089	Black
removeSession	13	40.50	0.35	.727	Black
getDetails	24	3.00	4.20	.000**	White
setDetails	16	47.50	1.06	.289	White

Legend: * Significant where $p=0.05$

** Significant where $p=0.01$

Table 6.3. Wilcoxin Matched Pairs Test on Performance

The difference between the latency recorded for each method was calculated by subtracting the black-box method latency from the white-box method latency. The valid n represents how many non-zero differences were tested, as the Wilcoxin test only considers differences of a non-zero value (Table 6.3). The p -values highlighted in bold indicate significant differences identified in the service method latencies.

The mean value determines whether the white-box or the black-box data service performed better (Table 6.3). Six methods, namely `storePersonalDetails`, `getDetails`, `readFirstInfo`, `fillNoticeBoard1`, `getDemis` and `getSelectedSessions` performed significantly better for the white-box developed web services, whilst four methods, namely `readLastInfo`, `getAvailableSessions`, `addSession` and `getAvailableColumns` performed significantly better for the black-box developed services. Since six methods for white-box modernized services and four methods for black-box modernized services performed significantly better than their counterparts, the null hypothesis for mean latencies could be rejected (Section 6.2.2). The null hypothesis for the mean latencies is (Section 1.5):

$H_{0.1.1}$: Mean service method latencies for black-box and white-box modernized data services are not equivalent

The alternate hypothesis is therefore accepted, namely:

$H_{1.1.1}$: Mean service method latencies for black-box and white-box modernized data services are equivalent.

6.3.3. Effectiveness of Data Services Results

A within-subjects design for the usability evaluation was conducted with a sample of 30 participants. Each participant used both white-box and black-box modernized systems, but the order of the systems was counterbalanced (Section 6.2.3). That is, fifteen participants evaluated the black-box modernized system first followed by the white-box modernized system. The other fifteen participants evaluated the systems in reverse order.

The participants involved in the usability evaluation were required to be familiar with the existing Demi System (Section 6.2.3). The participants selected consisted of both male and female students from undergraduate and postgraduate levels of study (Table 6.4). All participants had completed at least one year of undergraduate studies.

		Gender			Home Language				
Study Level	Count	Male	Female	Sum	English	Afrikaans	Xhosa	Other	Sum
Undergraduate	16	13	3	16	8	2	3	3	16
Postgraduate	14	11	3	14	9	5	0	0	14
Total	30	24	6	30	17	7	3	3	30

Table 6.4. Biographical Statistics of Participants

Participants were selected from a variety of ethnic groups, as indicated by the various home languages spoken (Table 6.4). The participants were surveyed on their level of familiarity with the Demi System and how often they use the system per week (Table 6.5).

Occurrences per Week	Students Assisting Sessions	Students Using System
Less than Once	0	2
Once	7	7
Twice	8	7
More than Twice	15	14
Total	30	30

Table 6.5. Participants Familiarity with the Demi System

The breakdown of familiarity with the system in relation to the amount of sessions the students assist with per week is reported (Table 6.5). It is evident that some students do not use the system every week, despite the fact that all students have at least one session every week. In terms of specific functionalities that the system provides, participants were asked to select the functionalities for which they used the system. The count of participants using a certain system process is tabulated in Table 6.6. The most well used system process is session attendance. Printing one's list of selected sessions is the least used system functionality.

System Functionality	Students
Application	15
Attendance	30
Session Selection	21
Session Removal	19
Print Session List	11
Change Details	15

Table 6.6. Frequency Distribution of System Functionality

During the usability evaluation, participants were each given five tasks (Appendix C) to complete for each system (Section 6.2.3). The participants did not know which system was modernized by the white-box approach or the black-box approach. The systems were identified as System 1 and System 2. The performance metrics collected for each task in System 1 and System 2 need to be compared. Task completion results indicated that one participant was unable to complete the fourth task for the white-box modernized system and one participant was unable to complete the fourth task for the black-box modernized system. Task success results indicated that a slightly higher task success rate was achieved for the white-box modernized system than the black-box system (Figure 6.4).

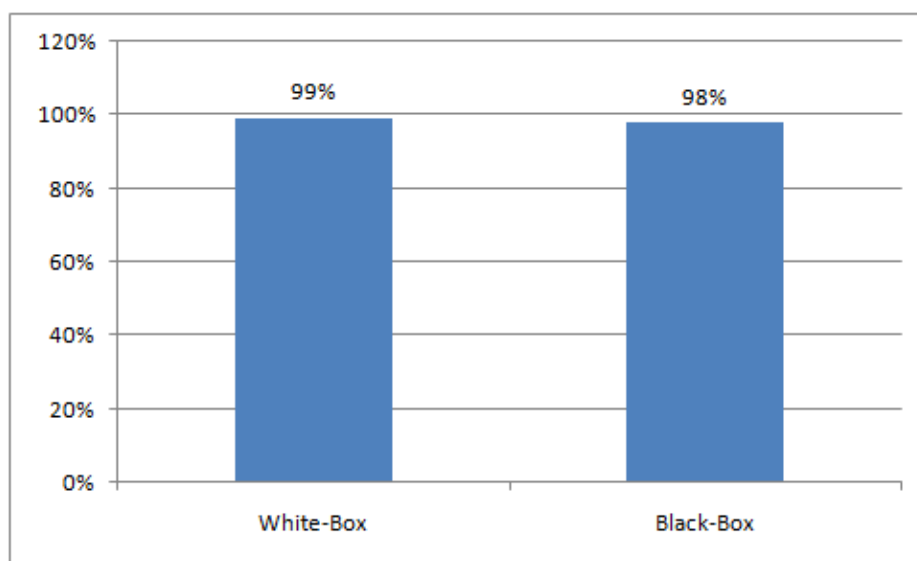


Figure 6.4. Success Rate Comparison over All Five Tasks

Three tasks were unsuccessfully completed for the black-box modernized system, whereas only one task was unsuccessfully completed for the white-box modernized system (Figure 6.4). Error rates were recorded per task and totalled for each system that the users evaluated (Table 6.7).

Number of Errors	Black-Box System	White-Box System
0 Errors	12	13
1 Error	14	15
2 Errors	3	1
3 Errors	1	1

Table 6.7. Frequency Distribution of Errors across Both Systems

The error rates for each system were compared using the ANOVA test (Section 6.2.3). The result yielded from this analysis was that no significant difference existed ($p = 0.092$). Since no significant difference was determined between the two modernized systems for error rates, the respective null hypothesis $H_{0.2.2}$ (Section 1.5) can be rejected:

$H_{0.2.2}$: Mean error rates for black-box and white-box modernized systems are not equivalent

The alternate hypothesis $H_{1.2.2}$ is therefore accepted:

$H_{1.2.2}$: Mean error rates for black-box and white-box modernized systems are equivalent

An overall task time performance test was conducted to see which system obtained better results. The task times were calculated for each of the five tasks for the two systems. The time values for each task were normalised to a z-score (Kranzler and Moursund 1999). The z-scores for each task were then averaged for the five tasks to obtain a total z-score for task times for each modernized system. An ANOVA test was applied to the total z-scores to assess if a significant difference in system performance existed between the two systems. A p-value of 0.051 was obtained for this test, indicating that no significant difference between the two systems exists. The p-value, however, does tend towards a significant difference. When assessing the mean of the total z-score value ($M=-0.19$), it signifies that the white-box modernized system performed faster than the black-box modernized system.

A more detailed analysis of each task time (Appendix E) compared between the two systems shows which tasks achieved better performance for each system (Table 6.8). The times recorded for all participants for Task 1 were compared using ANOVA. A significant difference in means was determined. The further analysis of the mean value indicates that the white-box modernized system performed better than the black-box modernized system for the first task.

Analysis of Task 2 completion times revealed that a significant difference was calculated in the task times of the two systems. Similarly, on analysis of the mean, Task 2 performed better in the white-box modernized system. Thus, the results achieved for task completion times for both Tasks 1 and 2 cannot reject the null hypothesis (Section 1.5):

$H_{0.2.1}$: Mean task times for black-box and white-box modernized data services are not equivalent

Task	Mean	p-Value	Better Times Achieved
Task 1	-20.83	.003	White-box system
Task 2	-8.00	.038	White-box system
Task 3	2.40 (white-box first) -5.73 (black-box first)	.033	Both systems
Task 4	9.60 (white-box first) -18..60 (black-box first)	.004	Both systems
Task 5	-4.53 (white-box first) 14.67 (black-box first)	.008	Both systems

Table 6.8. Comparative Statistics for Task Completion Times

The analysis of Task 3 times using the ANOVA test illustrated that the order in which the participants used the systems made a significant difference in task times recorded (Table 6.8). Analysis of the means showed that when the white-box modernized system was used first, participants took longer to complete the task using the white-box system than the black-box system. Similarly, when participants used the black-box system first they took longer to complete the third task in the black-box system than the white-box system. The better performance of the task in the system used second in order is an indication that the participants had familiarised themselves with the task and were able to complete it in a shorter time using the second system.

A similar significant difference in the order in which the systems were used was obtained from the ANOVA test when applied to Task 4 (Table 6.8). The same observation applies to the fourth task as the third task. Participants obtained better task times on the second system they used as they had learned from completing the task on the first system.

The analysis of Task 5 revealed a significant difference in the order in which the systems were used (Table 6.8). The analysis of the means showed this time, however, that when participants obtained better tasks times on the first system used than on the second system. This result was achieved for both the cases when the white-box modernized system was used first and when the black-box modernized system was used first. This result appears incongruent with the results obtained from analysis of Tasks 3 and 4. Clearly, the learning effect was not the cause of the worsened task times on the second system used. Further inspection of the tasks conducted revealed that the nature of the fifth task itself was the cause of the worsened performance using the second system. The fifth task for the second system

required participants to enter more information into the system than the fifth task for the first system. Thus, the times taken to complete the task on the second system were greater.

The conclusion drawn from the comparative analysis of the third, fourth and fifth task is that the null hypothesis $H_{0.2.1}$ (Section 1.5) can be rejected for each of these tasks:

H_{0.2.1}: Mean task times for black-box and white-box modernized data services are not equivalent

No significant difference in the overall mean times for these three tasks between the two modernized systems could be found. Only the order in which they were performed affected the mean task completion times of the two systems. Thus, the alternative hypothesis is accepted for Tasks 3, 4 and 5:

H_{1.2.1}: Mean task times for black-box and white-box modernized data services are equivalent

Not only was the performance data analysed by comparative statistics to compare the two modernized systems, but the self-reported metrics collected were compared as well. The results gathered from the SUS questionnaire completed by each participant for each modernized system were summed to achieve overall SUS scores for each system. The SUS scores for the white-box and black-box modernized systems were compared for each participant (Figure 6.5).

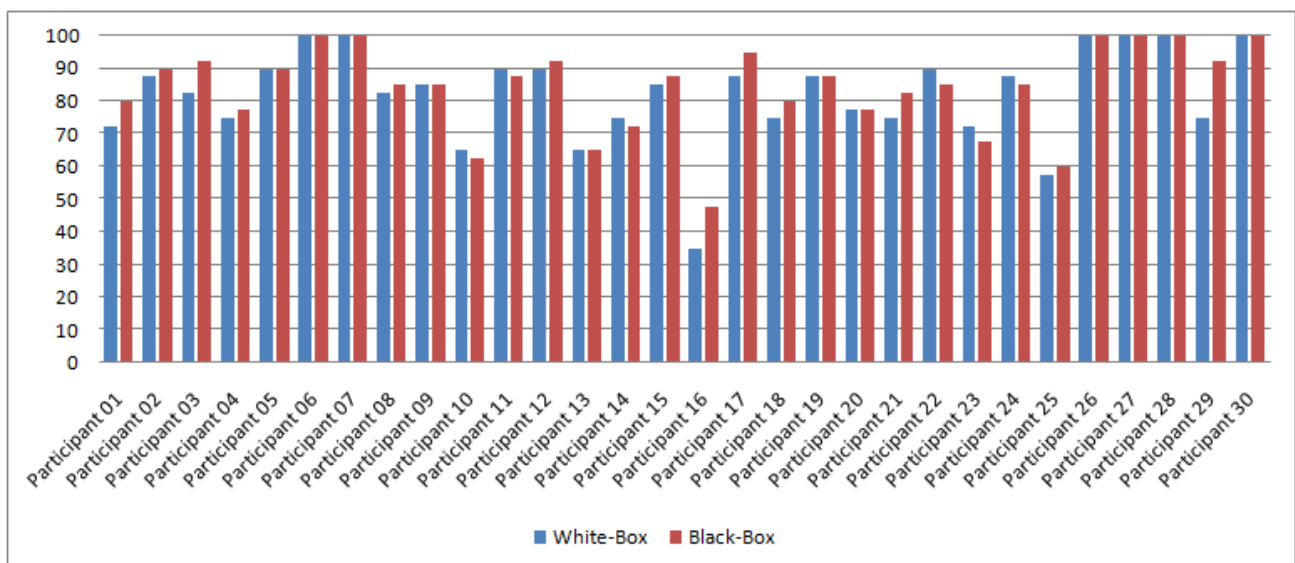


Figure 6.5. Comparison of SUS Scores per Participant

A comparative analysis using ANOVA was conducted to determine whether a significant difference exists in the user's satisfaction of the two systems. This further analysis supports the calculation of the SUS scores and provides a more detailed understanding of the user satisfaction with the two systems. The positive statements of the SUS questionnaire were grouped together and compared over the two systems by performing the ANOVA test on the difference between scores for the white-box and black-box systems. The black-box scores for the positive statements were subtracted from the white-box scores for the positive statements. The ANOVA test presented a significant difference between the two systems in terms of the positive statements ($p=0.016$). The analysis of the mean ($M=-0.05$) indicated that the participants answered the SUS more positively for the black-box system.

Similarly, the negative statements of the SUS were grouped together and their scores summed for each of the two systems. The difference between the negative statements scores were calculated by subtracting the black-box scores from the white-box scores. ANOVA revealed that no significant difference between the two systems existed in the comparison of the negative statements scores of the SUS. Finally, the overall SUS scores for the two modernized systems were compared to one another. The ANOVA result for this comparison yielded a significant difference in overall SUS scores ($p=0.033$). The mean indicated that the preference was towards the black-box modernized system ($M=-0.18$). Thus, overall, the participants showed a higher level of satisfaction with the black-box modernized system. These results can therefore not reject the null hypothesis (Section 1.5):

H_{0.2.3}: Mean SUS scores for black-box and white-box modernized data services are not equivalent

The results obtained during the usability evaluation indicated no significant difference between the two modernized systems in terms of error rates. The task times compared to each other between the two modernized systems showed that two of the tasks (Task 1 and 2) were performed faster in the white-box modernized system than the black-box modernized system. This increased performance was observed regardless of the order in which the systems were used. Two of the tasks, when compared (Task 3 and 4), indicated that the order in which they were completed affected the time the participants took to complete the tasks. The task completed on the first system took longer to complete than the task completed on the second system. The fifth task took longer to complete on the system used second in order than on the system used first due to the nature of the task. Lastly, upon comparative analysis of the SUS

scores obtained from the participants, the black-box modernized system received better satisfaction ratings than the white-box system.

6.4. Conclusion

The comprehensive evaluation framework has been applied to the comparison of the black-box and white-box modernization approaches used to modernize the existing Demi System. The experiments devised for each branch of the comprehensive evaluation strategy were executed to gather data for a comparative analysis (Section 6.2).

The experiment to measure developer effort to modernize the systems was conducted by analysing the code created by the developer during modernization (Section 6.2.1). The code analysis was performed using identified software metrics (Section 5.2.1). The results yielded from this experiment indicated that far more effort was required by the developer to apply the white-box modernization approach to the existing system than the black-box approach (Section 6.3.1).

The experiment to measure QoS metrics was conducted by analysis of the code of the data services generated as well as measurement of the performance of the data services (Section 6.2.2). The quality of the data services was evaluated based on the identified QoS metrics specifically relevant to data services (Section 5.2.2). The comparative analysis of service latency showed that in six methods the white-box data services outperformed the black-box data services whilst in four methods the black-box data services outperformed the white-box data services (Section 6.3.2). The null hypothesis $H_{0.1.1}$ (Section 1.5) could therefore be rejected. Since the hypothesis $H_{0.1.1}$ was rejected and no other significant differences in QoS were discovered by inspection, the more general hypothesis $H_{0.1}$ could be rejected as well:

$H_{0.1}$: QoS of black-box and white-box modernized data services is not equivalent.

The experiment to measure the effectiveness of the data services was performed through user studies (Section 6.2.3). The user studies were carried out in a formal controlled environment where usability data was gathered through recognised techniques (Section 5.2.3). Results showed that high completion and success rates were achieved for both modernized systems (Section 6.3.3). The results of the usability evaluation yielded a similarity in the number of errors made by participants in both systems, thus rejecting the null hypothesis $H_{0.2.2}$.

Comparative analysis of the task completion times showed that participants completed two of the five tasks more quickly using the white-box modernized system, thus unable to reject the null hypothesis $H_{0.2.1}$ for these two tasks. Analysis of the SUS ratings of each modernized system showed, however, that the participants preferred the black-box modernized system to the white-box modernized system, thus unable to reject the null hypothesis $H_{0.2.3}$.

The more general hypothesis for the data service effectiveness component of the evaluation framework $H_{0.2}$ (Section 1.5) could therefore be rejected as a result of the rejection of two of its sub-hypotheses. The rejected hypothesis for data service effectiveness is:

$H_{0.2}$: Effectiveness of black-box and white-box modernized data services is not equivalent.

The results of these three evaluation components are tabulated in Table 6.9.







Evaluation Component	White-Box Approach	Black-Box Approach
QoS		
Developer Effort		
Effectiveness of Data Services		

Table 6.9. Results Obtained from Comprehensive Evaluation Framework

The rejection of the hypotheses $H_{0.1}$ and $H_{0.2}$ for the comparison of modernized data services in terms of QoS and effectiveness therefore indicate that neither modernization approach is preferable in these two evaluation components as they appear to perform equivalently. The overall hypothesis regarding the comparison of modernized data services (Section 1.5) could therefore be rejected, namely:

H_0 : Black-box and white-box modernized data services are not equivalent.

It can be seen that if only one of the evaluation branches had been used to compare the two modernization approaches a different result would have been achieved for each evaluation (Tabl 6.9). The independence of the evaluation strategies of the three branches allows them to be combined (Chapter 5). The combination of the three strategies has supplied an understanding of the comparison of modernization approaches from different perspectives.

For the purposes of determining which modernization approach is more suitable for the production of data services, it seems that the black-box modernization approach is favourable (Table 6.9). This selection is made by inspection, due to the fact that the developer effort results showed significantly reduced effort than the white-box approach (Section 6.3.1). Furthermore, the satisfaction of the users with the modernized system was higher than with the white-box system (Section 6.3.3). In addition to these results, the QoS evaluation showed that the black-box data services were of no less quality than the white-box data services (Section 6.3.2). Thus, the combination of the three identified evaluation branches of the comprehensive framework has produced a more holistic comparison of the two modernization approaches.

Chapter 7: Conclusion

7.1. Introduction

Systems which have begun to resist maintenance require modernization. Various modernization approaches exist, each with different characteristics. This investigation applied the white-box and black-box modernization approaches to a case study to obtain data services for an SOA. It was unclear how to compare these two modernization approaches in order to determine which approach was more appropriate for the creation of data services from an existing system.

This investigation has proposed a comprehensive evaluation framework for the evaluation of a modernization approach that generates data services (Section 1.3). The comprehensive evaluation framework presented consists of three distinct evaluation branches derived from existing evaluation strategies. The framework evaluates the developer effort to modernize a system, the quality of the data services developed through modernization and the effectiveness of these data services.

Each of these evaluation strategies exists independently, but the need for a combination of these approaches when evaluating the modernization of a system to data services has arisen. The framework developed for this investigation required an understanding of system modernization and the various modernization approaches that exist (Chapter 2). An

understanding of SOA and data services in particular was also required to formulate the evaluation framework (Chapter 3). The modernization approaches were required to be applied to a case study to generate data services (Chapter 4). The outcomes of these modernization attempts provided a platform on which to apply the proposed evaluation framework as a comparison tool (Chapter 6).

The comprehensive evaluation framework (Chapter 5) aims to provide a specific evaluation tool to address the need to evaluate modernization approaches for the development of data services (Section 1.2). It was proposed that the combination of three evaluation strategies could provide a more detailed understanding of the modernization approach in terms of its design, process and outputs.

7.2. Achievements

The development of a comprehensive evaluation framework first required an understanding of modernization approaches and the target architecture for data services. Modernization is an alternative to system maintenance or system replacement (Section 2.2). Two modernization approaches were identified to be applied to the case study in this investigation, namely white-box modernization and black-box modernization. These two modernization approaches form the basis for the use of the proposed evaluation framework as a comparison tool as they are opposite in nature. The comparison of these two approaches at a conceptual level was required in order to apply each of them to the case study. The use of these approaches on the case study achieved the research objective (Section 1.4).

White-box modernization is an invasive approach requiring knowledge of the existing system code and functions (Section 2.3). On the other hand, black-box modernization is reported as being a non-invasive approach where only the inputs and outputs of the system require analysis (Section 2.4). Various implementation techniques exist for each of these modernization approaches (Section 2.5). The most appropriate techniques were selected to be applied to the case study for each modernization approach.

The outcome of the modernization approaches required the development of web services that provide data as a service to achieve another research objective (Section 1.4). Various service architectures exist for the development of these data services. The SOA was identified as the

target platform in which to develop the data services. SOA is recognised for its provision of software as a service to consumers as opposed to specific tailor-made applications (Section 3.2). Two well-known service architectures were reviewed for use in this investigation. SOAP style web services use specific standards and protocols in a XML based communication environment (Section 3.3). Alternatively, REST style web services are more lightweight and rely on HTTP for transmission of responses and requests (Section 3.4). When comparing the advantages and disadvantages of each of these service styles, it was discovered that SOAP services would be preferable for the development of data services due to their secure and reliable transmission of messages (Section 3.5).

The primary research objective for this study was the development of a comprehensive evaluation framework to evaluate a modernization approach (Section 1.4). This research objective was met through the investigation of existing independent evaluation strategies, namely:

- Quality of Service guidelines for data services;
- The measure of a developer's effort from software metrics; and
- The measure of effectiveness of a modernized system by user studies.

The investigation of each of these existing evaluation strategies achieved further secondary research objectives to understand the metrics derived from each strategy (Section 1.4). These existing evaluation strategies formed the three components of the comprehensive evaluation framework proposed for this investigation (Chapter 5). The evaluation framework required validation, thus it was applied to a white-box modernization approach (Section 5.3). The modernization approach was evaluated in terms of the three evaluation components. Results of the validation showed how the results of each evaluation leg complemented the results of each of the other evaluation legs (Section 5.4). The combination of results from all three evaluation components lead to the determination of modernization success.

The final secondary research objective included the application of this evaluation framework to the comparison of the two modernization approaches applied to the case study (Section 1.4). This comparison involved the development of experiments to compare each of the modernization approaches on the metrics for each component of the evaluation framework (Section 1.5). Specific hypotheses were devised for each of the quantitative comparative experiments. Each of these hypotheses for QoS and for effectiveness of the modernized data

services were designed to support a more general hypothesis for the comparison of modernized data services, namely:

H_0 : Black-box and white-box modernized data services are not equivalent

H_1 : Black-box and white-box modernized data services are equivalent.

The specific sub-hypotheses for each of the experiments conducted in the other two components of the evaluation framework (Section 6.3.2, Section 6.3.3) were tested in order to determine if they could be rejected or not (Figure 7.1).

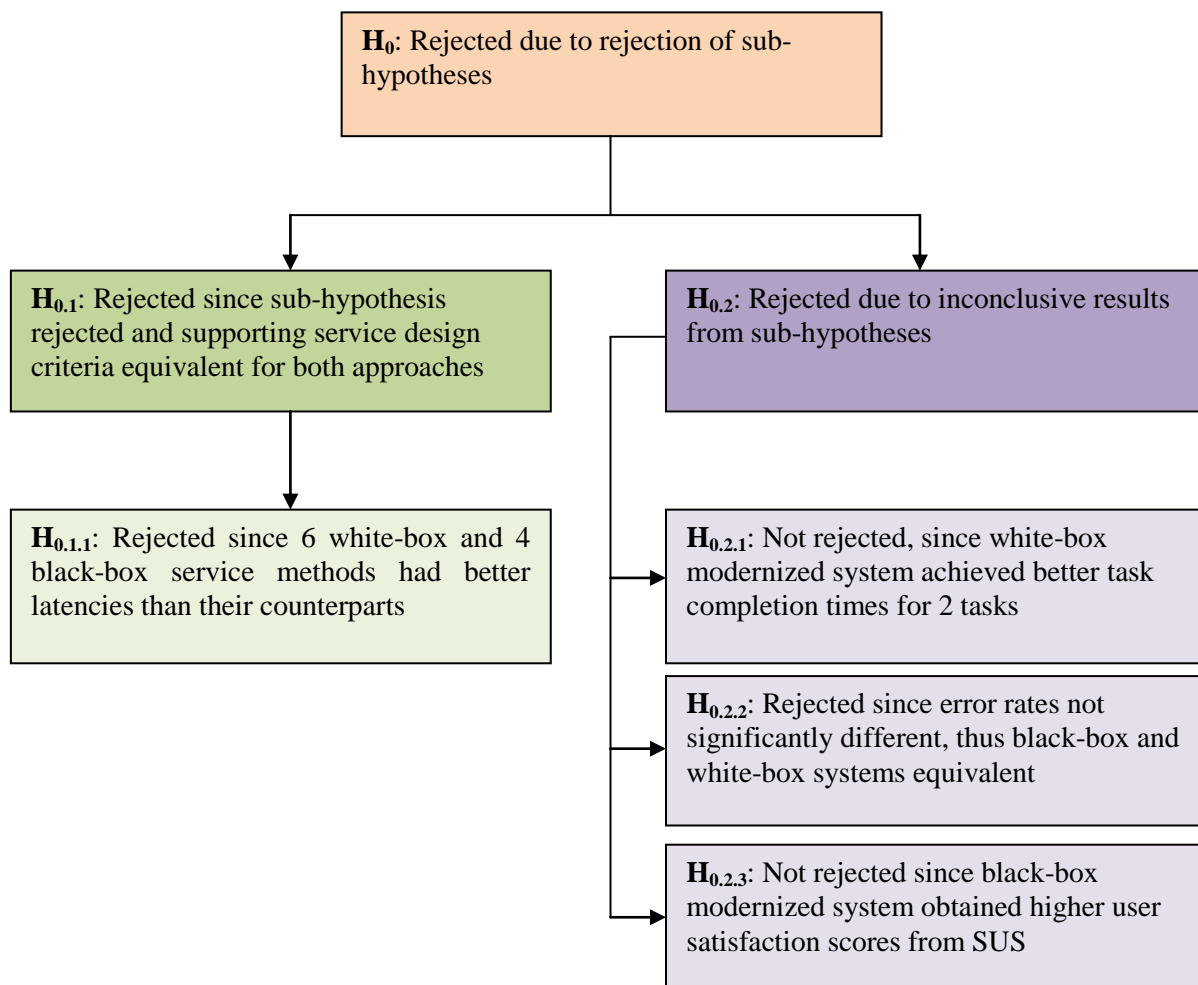


Figure 7.1. Effects of Experiments on Hypotheses

The evaluation component for measuring developer effort did not contain a hypothesis based on statistical significance. The results of this evaluation component, however, yielded that far more effort was required by the developer to apply the white-box modernization approach to the case study (Section 6.3.1). The proposal and application of this comprehensive evaluation framework formed the theoretical and practical contributions, respectively, of this research.

7.3. Contributions

The contributions of this study may be divided into theoretical contributions and practical contributions. The theoretical contribution involved the proposal of a comprehensive evaluation strategy for modernization of systems to data services (Section 7.2.1). The practical contributions include the application of two modernization approaches to the case study as well as the application of the evaluation framework to the comparison of the modernization approaches (Section 7.2.2).

7.3.1. Theoretical Contribution

The proposed comprehensive evaluation framework for modernization towards data services consists of three recognised evaluation strategies (Section 5.2). The three evaluation branches of the framework are:

- Quality of the services resulting from the legacy system modernization process will be measured against acknowledged Quality of Service (QoS) standards;
- A measure of the effort required by the developer to create the modernization approach; and
- A measure of the effectiveness of the data services generated will be measured by the use of empirical evaluations with system users.

The quality attributes of web services were identified and cross referenced to determine which attributes were most appropriate for data services (Section 5.2.2). Five QoS attributes were selected as the guideline for the development of high quality data services in particular (Table 7.1). For the measurement of effort required by the developer, a well known suite of object-oriented software metrics were analysed to determine which metrics measure complexity of code (Section 5.2.1).

It was determined that three of the software metrics could be used to measure code complexity of developed code by analysis of the methods in the code and the hierarchy of the class (Table 7.1). To measure the effectiveness of the data services generated by modernization, usability studies were presented (Section 5.2.3). The collection of performance and self-reported metrics through user evaluation in a formal controlled environment were reported as an appropriate measure for the effectiveness of software (Table 7.1).

QoS Component
Security
Reliability
Interoperability
Reusability
Performance
Developer Effort Component
Weighted Methods per Class (WMC)
Number of Children (NOC)
Depth in Inheritance Tree (DIT)
Service Effectiveness Component
Task Completion
Task Success
Time on Task
Error Rates
Self-Reported Feedback (SUS)

Table 7.1. Comprehensive Evaluation Framework Metrics

This validated framework was composed of the combination of these three independent evaluation strategies (Section 5.3). Each strategy, when applied on its own, produces a result as to the success of a modernization attempt. The combination of the three evaluation branches to form a comprehensive framework used these complementary evaluation strategies to obtain a deeper understanding of the success of the modernization approach.

7.3.2. Practical Contribution

To support the development of the comprehensive evaluation framework, it was necessary to apply two modernization approaches to a case study in order to compare these two approaches. The practical contributions of this investigation thus are:

- Application of the white-box and black-box modernization approaches to the case study; and
- Application of the comprehensive evaluation framework to the two modernization approaches applied to the case study.

The white-box and black-box modernization approaches were applied to the Demi System to generate data services for a SOA (Chapter 4).

7.3.2.1. Application of Modernization Approaches

The Demi System was the existing system suitable for modernization by the two identified approaches. The system was developed for use in the Department of Computing Sciences at NMMU (Section 4.2). The Demi System was developed in a programming language that is no longer suitable for future maintenance in the department. The modernization of this system to expose data as a service in a SOA would elongate the lifespan of the system. Therefore, this system was selected as a suitable case study to investigate the effects of the two modernization approaches applied to the system.

The white-box modernization approach applied techniques discovered in related studies, such as program understanding and code migration (Section 2.3). The program understanding phase aided in gaining system understanding through diagrammatic representations of the existing system where documentation was missing (Section 4.3.1). The knowledge of the existing system gained through program understanding brought about the discovery of components for the identification of data services. The next white-box modernization phase, code migration, enabled the migration of existing system code to a new, more maintainable language (Section 4.3.2). Existing VB code was therefore migrated to Java code. Certain challenges were encountered during code migration, where existing system functionalities were not easy to replicate in the new language. These challenges were overcome through the discovery of specific Java API's and components which allowed the same functionality to be catered for.

The component-based analysis (Section 2.5) of the system through program understanding as well as analysis of the existing system classes enabled web service discovery (Section 4.3.3). Eight components were discovered which were each developed as a data service during white-box modernization. All data operations performed on the database following the

CRUD model were developed as individual data service methods within the appropriate data service. The data services were developed using the J2EE platform for communication between the data services and the web service client application. All data operations were migrated from the .NET database connectivity to Java's JDBC. The web service client application was then able to be connected to the data services to perform any data operations on the system's database.

The black-box modernization approach applied to the Demi System consisted of two phases, namely interface modernization and component-based service discovery. The user interface modernization phase allowed the analysis of inputs and outputs of the existing system (Section 4.4.1). This analysis of the inputs and outputs of the system was accomplished during interaction with the system, as an application of the user interface modernization technique (Section 2.5). Through the analysis of the inputs and outputs of the system, the components for data service generation were identified.

Seven components suitable for development as data services were identified through interaction with the existing system. All existing system functionality not related to these data service components was then migrated from the existing VB code to Java code, similarly to the white-box modernization approach. Only the data service components were wrapped as web services due to the specific need to modernize only data services from an existing system (Section 1.3). The component-based web service discovery phase enabled the creation of data services which wrapped the original system's data operation code (Section 4.4.2). This web service wrapping of components was identified as an appropriate non-invasive technique for applying black-box modernization to a system (Section 2.6).

The research objective to apply the white-box and black-box modernization approaches to the case study using the techniques identified in literature was thus achieved (Section 1.4). Furthermore, the development of data services through the application of the two modernization approaches to the case study obtained the respective research objective.

7.3.2.2. Application of the Evaluation Framework

A further practical contribution was the application of the proposed comprehensive evaluation framework (Chapter 5) to the comparison of the modernization approaches. This practical contribution addressed the research objective to perform a comparative analysis of

two distinct modernization approaches (Section 1.4). The comprehensive evaluation framework proposed for this investigation was applied to the white-box and black-box modernization approaches implemented on the case study (Section 6.2). Specific experiments were designed to conduct each of the three evaluation components of the comprehensive framework. The results of the experiments conducted in the comprehensive evaluation are tabulated in Table 7.2.

	White-Box	Black-Box
QoS Component		
Security	SOAP Design	SOAP Design
Reliability	SOAP Design	SOAP Design
Interoperability	J2EE Design	.NET and J2EE Design
Reusability	High	High
Performance	6 methods significantly faster	4 methods significantly faster
Developer Effort Component		
Weighted Methods per Class (WMC)	High	Low
Number of Children (NOC)	Low	Low
Depth in Inheritance Tree (DIT)	Low	Low
Service Effectiveness Component		
Task Completion	99%	99%
Task Success	99%	98%
Time on Task	2 out of 5 tasks significantly faster	No tasks faster
Error Rates	Low	Low
Self-Reported Feedback (SUS)	No significant preference	Significantly more positive

Table 7.2. Detailed Results of Comparison using the Comprehensive Evaluation Framework

The application of the comprehensive evaluation framework to measure effort required by the developer to modernize the Demi System involved the analysis of the code developed during modernization using identified software metrics (Section 6.2.1). The results of the code complexity analysis conducted during the comparative evaluation of the two modernization

approaches yielded that the white-box modernization approach required far greater effort from the developer than the black-box modernization approach (Table 7.2).

A second experiment was designed to compare the data services developed through modernization in terms of their quality (Section 6.2.2). Five guidelines for high quality data services were identified in the comprehensive evaluation framework to be adhered to (Section 5.2.2). Three of the quality attributes were unable to be measured quantitatively, but were catered for in the design of the data services during both modernization approaches. The performance and reusability of the data services were measured, however, by quantitative methods (Section 6.2.2). The results of the comparative analysis of the data services performance and reusability (Table 7.2) showed that the data services developed via both modernization approaches could not be proved significantly different (Section 6.3.2). The hypothesis for data service performance ($H_{0.1.1}$) was thus rejected (Figure 7.1). As a result of the equivalence of the modernized data services in terms of their QoS attributes, the more general hypothesis $H_{0.1}$ was also rejected.

A third experiment was designed to evaluate the effectiveness of the data services by conducting user studies on the two modernized systems (Section 6.2.3). The user evaluation was designed to collect performance and self-reported data. Results obtained from the analysis of these performance metrics gathered showed that the two modernized systems both obtained high task completion and success rates. No significant difference could be found in a comparative analysis of the error rates for each of the modernized systems, thus the hypothesis $H_{0.2.2}$ was rejected (Figure 7.1). The comparative analysis of task completion times yielded that two out of the five tasks were completed significantly more quickly using the white-box modernized system than the black-box modernized system (Section 6.2.3). Thus the null hypothesis $H_{0.2.1}$ could not be rejected (Figure 7.1). The comparison of the other three task completion times did not exhibit a preference to any particular system. For these three tasks, the order in which the systems were used indicated a significant difference in task completion times.

The self-reported data was gathered using a recognised instrument, the SUS (Section 6.2.3). Analysis of the results gathered from the SUS questionnaires for each system indicated an overall preference towards the black-box modernized system (Table 7.2). Furthermore, a detailed analysis of the positively phrased statements in the SUS revealed that participants

were more positive about the black-box modernized system than the white-box modernized system. Thus the hypothesis $H_{0.2.3}$ could not be rejected (Figure 7.1).

The conclusions drawn from this experiment indicated that the white-box modernized system obtained preferable task completion time results, but the black-box modernized system was preferred in terms of user satisfaction. In terms of the effectiveness of the data services, quicker task time completion of two out of the five tasks indicates that the white-box modernized system is favourable. Similarly, the satisfaction of the participants with the black-box modernized system indicates that the users find the black-box modernized system more effective in performing the tasks they require it to perform. Thus, the more general hypothesis, $H_{0.2}$ was rejected due to the inconclusive results obtained from the usability evaluation for effectiveness of the data services (Figure 7.1). The overall hypothesis for the comparison of white-box modernized data services to black-box modernized data services, H_0 was rejected as a result of the results obtained from the sub-hypotheses for the two evaluation components, namely QoS and effectiveness of data services. Despite the equivalent performance of the data services in terms of quality and effectiveness, the inclusion of the results obtained from the developer effort evaluation show that the black-box modernization approach is more suitable due to the reduced effort in comparison to the white-box approach.

The combination of the results obtained from the comprehensive evaluation framework leads to a deeper understanding of each modernization approach from the three perspectives evaluated (Section 6.4). The comparison of which approach is more suitable for each evaluation component made it possible to determine which modernization approach was more suitable for the task of data service development. It was therefore discovered, as a result of inspection of results from the three components of the framework, that the black-box modernization approach was most suitable approach for this case study (Table 6.9).

7.4. Limitations

This investigation was focused on the application of the white-box and black-box modernization approaches only. A hybrid of the two approaches, namely the grey-box modernization approach was identified, but excluded from this investigation in order to clarify the comparison of two distinct and independent approaches (Section 1.6).

To apply these two modernization approaches to a real-world scenario a case study was used (Section 1.5). The case study chosen for this investigation was the Demi System for the Department of Computing Sciences at NMMU (Section 4.2). The Demi System did not exhibit the main characteristic of a legacy system, which is its difficulty to maintain after use for a long time (Section 2.2). The Demi System was, however, developed in a language that is not supported by the Department of Computing Sciences and would thus become difficult to maintain for future use. Consequently, the Demi System was selected as the most appropriate system on which to practice the identified modernization approaches.

The application of the modernization approaches to the case study produced the outcome of these modernization approaches, namely data services for a SOA. Two service-styles were contrasted and compared to decide in which service-style the data services would be implemented (Chapter 3). This study was only concerned with the development of one common service-style for the outputs of both modernization approaches, thus SOAP services were chosen.

The primary objective of this investigation was to develop a comprehensive evaluation framework for the analysis of a modernization approach that enables data service discovery (Section 1.4). The evaluation framework was limited to the inclusion of three evaluation strategies, namely:

- Quality of the services resulting from legacy system modernization measured against acknowledged QoS standards;
- A measure of the developer's effort required to apply the modernization approach; and
- A measure of the effectiveness of the data services generated by modernization through user evaluations.

Each of these components of the evaluation framework were limited to specific metrics suitable for the evaluation of the black-box and white-box modernization approaches as well as the evaluation of data services in particular.

The QoS metrics chosen for this investigation consisted of only the quality attributes related to data services in particular (Section 5.2.2). The measurement of effort required by the developer to modernize the existing system consisted only of software metrics concerned

with code complexity (Section 5.2.1). The metrics chosen were selected from a recognised suite of object-oriented metrics. Other code complexity metrics exist, but were not included in this investigation. The measure of effectiveness of the data services was conducted by the application of user evaluations on the modernized systems (Section 5.2.3). Well known user testing techniques were utilised to combine the collection of performance and self-reported metrics to measure the effectiveness of the modernized systems.

The three evaluation components of the comprehensive framework were equally weighted and not triangulated for this investigation. Instead, the results of each modernization approach were compared for each evaluation component to determine the most suitable modernization approach for that component. Consequently, the most suitable approach was determined by inspection of the overall results of the modernization approaches for the three evaluation components.

7.5. Future Work

The proposed comprehensive evaluation framework consisting of three components was applied to two modernization approaches applied to a case study. Due to the limitations of this framework and its application, the recommendations for future research include:

- The comparison of the object-oriented software metrics with more modern alternatives to refine the measurement of developer effort to modernize systems;
- Discovery of quantitative measures for security, reliability and interoperability of data services to provide a deeper comparative analysis of QoS;
- Weighting and triangulation of the results of the evaluation components to allow ranking of modernization approaches;
- Application of the comprehensive framework to the three modernization approaches, namely white-box, black-box and grey-box modernization; and
- Application of the evaluation framework to more than one case study to further validate its effectiveness.

These recommendations for future research aim to create a more robust and flexible evaluation framework. The application of this framework to more than one case study would obtain more results concerning the effectiveness of the framework independently of the case study used. These future research recommendations aim to address the limitations encountered in this investigation.

7.6. Summary

Research identified the lack of a comprehensive evaluation tool for the evaluation of modernization approaches that generate data services for an SOA. This investigation proposed an evaluation framework combining established evaluation components, namely:

- Quality of services resulting from legacy system modernization measured against acknowledged Quality of Service (QoS) standards;
- A measure of the effort required by the developer to apply the modernization approach; and
- A measure of the effectiveness of data services generated by the process of empirical evaluations with system users.

The comprehensive evaluation framework was applied to a case study on which two distinct modernization approaches had been executed. The two modernization approaches considered for this investigation were the white-box and black-box modernization approaches.

The evaluation framework was used as a tool for the comparison of these modernization approaches to determine which approach was more suitable for the generation of data services. The comparative analysis revealed that had only one of the evaluation components of the framework been applied to the case study, possible inconclusive results may have been obtained. The combination of the results, however, from the components of the evaluation framework revealed that black-box modernization was the more suitable approach for the generation of data services.

In future, further research into the components of this framework and the triangulation of results from each of the three components will serve to refine the framework. Furthermore, the application of this framework to more case studies will serve as a validation of the framework in terms of its effectiveness to determine the suitability of a modernization approach in the generation of data services.

References

- Barnes, M. and Cilliers, C. (2010): A Comprehensive Evaluation Strategy for the Modernization of Legacy Systems. *South African Telecommunications Network and Applications Conference*. Stellenbosch.
- Bianco, P., Kotermanski, R. and Merson, P. (2007): *Evaluating a Service-Oriented Architecture*. Carnegie Mellon University.
- Borkar, V., Carey, M., Lychagin, D., Westmann, T., Engovatov, D. and Onose, N. (2006): Query Processing in the AquaLogic Data Services Platform. *International Conference on Very Large Data Bases (VLDB'06)*.
- Canfora, G (2004): Software Evolution in the Era of Software Services. *International Workshop on the Principles of Software Evolution*.
- Canfora, G., Fasolino, A., Frattolillo, G. and Tramontana, P. (2006): Migrating Interactive Legacy Systems to Web Services. *Conference on Software Maintenance and Reengineering*.
- Canfora, G., Fasolino, A., Frattolillo, G. and Tramontana, P. (2008): A Wrapping Approach for Migrating Legacy System Interactive Functionalities to Service Oriented Architectures. *Journal of Systems and Software*, **81** (4):463-480.
- Carey, M (2006): Data Delivery in a Service-Oriented World: The BEA AquaLogic Data Services Platform. *International Conference on Management of Data*.
- Chiang, C. and Bayrak, C. (2006): Legacy Software Modernization. *International Conference on Systems, Man and Cybernetics*. Taipei, Taiwan.1304 - 1309.
- Chiang, R.H.L., Barron, T.M. and Storey, V.C. (1997): A Framework for the Design and Evaluation of Reverse Engineering Methods for Relational Databases. *Data and Knowledge Engineering*, **21**:57-77.
- Chidamber, S. and Kemerer, C. (1994): A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, **20**:476 – 493.
- Chidamber, S.R. and Kemerer, C.F. (1991): Towards a Metrics Suite for Object Oriented Design. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, **26**.

- Chung, S., Byung Chul An, J. and Davalos, S. (2007): Service Oriented Software Reengineering: SoSR. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*. IEEE Computer Society Washington.
- Colosimo, M., De Lucia, A., Scanniello, G. and Tortora, G. (2008): Evaluating legacy system migration technologies through empirical studies. *Information and Software Technology*, **51**:433-447.
- Comella-Dorda, S., Seacord, R.C., Wallnau, K. and Robert, J. (2000): A Survey of Black-Box Modernization Approaches for Information Systems. *International Conference on Software Maintenance*.
- Concas, G., Marchesi, M., Murgia, A., Pinna, S. and Tonelli, R. (2010): Assessing traditional and new metrics for object-oriented systems. In *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*:24-31.
- Devore, J. and Farnum, N. (2005): *Applied Statistics for Engineers and Scientists*. Second Edn, Thomson Brooks/Cole.
- Eckstein, R. and Mordani, R. (2006): *Introducing JAX-WS 2.0 With the Java SE 6 Platform, Part 1* [online]. Available at http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/. [Accessed on 18 November 2010].
- Erradi, A., Anand, S. and Kulkarni, N. (2006): Evaluation of Strategies for Integrating Legacy Applications as Services in a Service Oriented Architecture. *International Conference on Services Computing*.
- Erickson-Connor, B. (2003): Truth or Consequences: Legacy Application Modernization. *Z/Journal*:38-43. August/September 2003. Bob Thomas.
- Fenn, J., et al. (2009): *Hype Cycle for Emerging Technologies, 2009*.
- Fielding, R.T. and Taylor, R.N. (2002): Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, **2** (2):115-150.
- Flanagan, D. (2001): *Java™ Foundation Classes in a Nutshell*, O'Reilly & Associates. Available at http://docstore.mik.ua/oreilly/java-ent/jfc/ch03_20.htm. [Accessed on 18 November 2010].
- Garcia-Rodriguez de Guzman, I., Polo, M. and Piattini, M. (2007): An ADM Approach to Reengineer Relational Databases towards Web Services. *Working Conference on Reverse Engineering*. Vancouver.90-99.
- Gupta, S. (2004): *Metrics for Object Oriented Software Development* [online]. Available at <http://javaboutique.internet.com/tutorials/codemetrics/>. [Accessed on 19 November 2010].
- Hofstee, E. (2006): *Constructing a Good Dissertation: A Practical Guide to Finishing a Master's, MBA or PhD on Schedule*. Johannesburg, EPE.
- Jeong, B., Hyunbo, C. and Choonghyun, L. (2009): On the Functional Quality of Service (FQoS) to Discover and Compose Interoperable Web Services. *Expert Systems with Applications*, **36**:5411-5418.
- Kachru, S. and Gehringer, E.F. (2004): A Comparison of J2EE and.NET as Platforms for Teaching Web Services. In *Proceedings of Frontiers in Education*, **3**.
- Kilov, H. (1990): From Semantic to Object-Oriented Data Modeling. In *Proceedings of the First International Conference on Systems Integration*:385-393.
- Kranzler, G. and Moursund, J. (1999): *Statistics for the Terrified*. Second Edn, Prentice-Hall. 0-13-955410-6.
- Lee, E., Lee, B., Shin, W. and Wu, C. (2003): A Reengineering Process for Migrating from and Object-oriented Legacy System to a Component-based System. *International Computer Software and Applications Conference*.

- Lewis, G., Morris, E. and Smith, D. (2006): Analyzing the Reuse Potential of Migrating Legacy Components to a Service-Oriented Architecture. *Conference on Software Maintenance and Reengineering (CSMR '06)*.
- Li, F., Qian, C., Liu, W. and Chi, C. (2009): N-to-1 SOAP Requests Bundling for Efficient Software Service Delivery. *World Conference on Services*. Los Angeles.539-545.
- Liegl, P. (2007): The Strategic Impact of Service Oriented Architecture. *International Conference and Workshops on the Engineering of Computer-Based Systems*.
- Lim, N., Majumdar, S. and Nandy, B. (2010): Providing Interoperability for Resource Access Using Web Services. *Communication Networks and Services Research Conference*. Montreal.236 - 243.
- Liu, Y. and Connelly, K. (2008): Realizing an Open Ubiquitous Environment in a RESTful Way. *IEEE International Conference on Web Services*.96-103.
- McCabe, T. (1976): A Complexity Measure. *IEEE Transactions on Software Engineering*, 2:308-320. IEEE.
- Mehta, A. and Heineman, G. T. (2002): Evolving Legacy System Features into Fine-Grained Components. *International Conference on Software Engineering (ICSE'02)*.
- Miller, J.R. and Jeffries, R. (1992): Usability Evaluation: Science of Trade-Offs. *IEEE Software*, 9 (5):97-98,102.
- Mulligan, G. and Gračanin, D. (2009): A Comparison of SOAP and REST Implementations of a Service Based Interaction Independence Middleware Framework. *Winter Simulation Conference*.1423-1432.
- Nath, Y. (2009): *Calculation of Cyclomatic Complexity & understanding of its Properties* [online]. Available at <http://www.scribd.com/doc/16095618/Calculation-of-Cyclomatic-Complexity-understanding-of-its-Properties>. [Accessed on 25 November 2010].
- O'Reilly, T. (2005): *What is Web 2.0. Design Patterns and Business Models for the Next Generation of Software* [online]. Available at <http://oreilly.com/pub/a/Web2/archive/what-is-Web-20.html>. [Accessed on 12 November 2010].
- Olivier, M.S. (2004): *Information Technology Research: A Practical Guide for Computer Science and Informatics*. Pretoria, 2 Edn, Van Schaik. 0627025765.
- Pautasso, C., Zimmermann, O. and Leymann, F. (2008): RESTful Web Services vs. 'Big' Web Services: Making the Right Architectural Decision. In *Proceedings of the Fifth International Conference on World Wide Web*, Beijing:805-814.
- Peng, Y., Ma, S. and Lee, J. (2009): REST2SOAP: a Framework to Integrate SOAP Services and RESTful Services. *International Conference on Service-Oriented Computing and Applications*. Taipei.1-4.
- Rajshekhar, A.P. (2005): *Java Mail API: Getting Started* [online]. Available at <http://www.devarticles.com/c/a/Java/Java-Mail-API-Getting-Started/>. [Accessed on 10 August 2010].
- Salste, T. (2008): Aivosto Project Analyzer Ver. 9.0.
- Sanders, D., Hamilton, J. and Macdonald, R. (2008): Supporting a Service-Oriented Architecture. *Spring Simulation Multiconference*.
- Sangeetha, S. and Chinnici, R. (2007): *Using Annotations on the Java EE 5.0 Platform* [online]. Available at <http://today.java.net/article/2007/05/18/using-annotations-java-ee-50-platform>. [Accessed on 18 November 2010].
- Seacord, R. C., Plakosh, D. and Lewis, G. (2003): *Modernizing Legacy Systems Software Technologies, Engineering Processes, and Business Practices*. Addison-Wesley.
- Selçuk Candan, K., Li, W., Phan, T. and Zhou, M. (2009): Frontiers in Information and Software as Services. *International Conference on Data Engineering*.1761-1768.

- Sharp, H., Rogers, Y. and Preece, J. (2007): *Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons.
- Sjøberg, D.I.G., Dybå, T. and Jørgensen, M. (2007): The Future of Empirical Methods in Software Engineering Research. *Future of Software Engineering (FOSE '07)*.
- Sneed, H. M. and Sneed, S. H. (2003): Creating Web Services from Legacy Host Programs. *International Workshop on Web Site Evolution (WSE'03)*.
- Strawmyer, M. (2003): *Serialization/Deserialization in .NET* [online]. Available at <http://www.developer.com/net/csharp/article.php/3110371>. [Accessed on 23 November 2010].
- Stroulia, E., El-Ramly, M. and Sorenson, P. (2002): From Legacy to Web Through Interaction Modeling. In *Proceedings of the International Conference on Software Maintenance*:320 - 329.
- Thiran, P., Hainaut, J., Houben, G. and Benslimane, D. (2006): Wrapper-based Evolution of Legacy Information Systems. *ACM Transactions on Software Engineering and Methodology*:329-359.
- Tullis, T. and Albert, B. (2008): *Measuring the User Experience*. Morgan Kaufmann.
- Tullis, T.S. and Stetson, J. (2004): A Comparison of Questionnaires for Assessing Website Usability. In *Proceedings of the Usability Professionals Association Conference*, Usability Professionals Association.
- Turpin, A. and Scholer, F. (2006): User Performance versus Precision Measures for Simple Search Tasks. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*:11-18. ACM.
- Wang, X., Hu, S.X.K. and Garton, H. (2007): Integrating Legacy Systems within the Service Oriented Architecture. *Power Engineering Society General Meeting*.1-7.
- Whitten, J.L., Bentley, L.D. and Dittman, K.C. (2004): *Systems Analysis and Design Methods*. Sixth Edn, McGraw Hill/Irwin. 0-07-247417-3.
- Zhang, B., Bao, L., Zhuo, R., Hu, S. and Chan, P. (2008): A Black-Box Strategy to Migrate GUI-Based Legacy Systems to Web Services. *IEEE International Symposium on Service-Oriented System Engineering*.
- Zhang, Z. and Yang, H. (2004): Incubating Services in Legacy Systems for Architectural Migration. *Asia-Pacific Software Engineering Conference (APSEC'04)*.

Appendices

A Holistic Evaluation Strategy for the Modernization of Legacy Systems

Meredith A. Barnes and Charmain Cilliers

Department of Computing Sciences

Nelson Mandela Metropolitan University

P.O. Box 77000, Port Elizabeth, 6031

Tel: +2741 – 504 2094, +2741 – 504 2235

email: MeredithAnne.Barnes@nmmu.ac.za; Charmain.Cilliers@nmmu.ac.za

Abstract-Service Oriented Architecture has developed into a current paradigm for business application solutions. This development calls for a need to modernize legacy systems which store the fundamental business procedures for large organisations. The primary goal of this paper is to present a holistic evaluation strategy which can be used to assess the end product of a modernization process applied to a legacy system, specifically a data service in an SOA. The evaluation strategy is comprised of a three-legged approach. The first evaluation leg consists of the analysis of the quality of service of the data service generated by the modernization process. The second leg investigates the effort required by the developer to modernize the original legacy code, depending on the type of modernization approach selected. The third leg measures the efficacy of the data services generated by empirical testing upon completion of the modernization process. The envisaged contribution is the development of an evaluation strategy that ensures successful modernization of legacy systems by assessing each of the major components of the modernization process.

Index Terms – Quality of service (QoS), Software evaluation, Modernization

I. INTRODUCTION

Service Oriented Architecture (SOA) has been identified as an emerging technology that is currently on the rise and will reach maturity within a few years (Figure 1). Consequently, a need for current research in this field has emerged in order to gain enlightenment on the technology.

Legacy systems are used by many organizations because of the data pertinent to the organizations' business processes that are still maintained by them. These legacy systems become outdated after years of use and require more than maintenance to keep them relevant [16]. The modernization of these legacy systems becomes necessary, therefore allowing organizations continued access to their core business functionalities in a more modern architecture.

The current technology climate (Figure 1) implies that a need to modernize legacy systems into web services conforming to SOA requirements will benefit organizations. Services typically consist of a collection of software elements, each of which executes a particular business process [13].

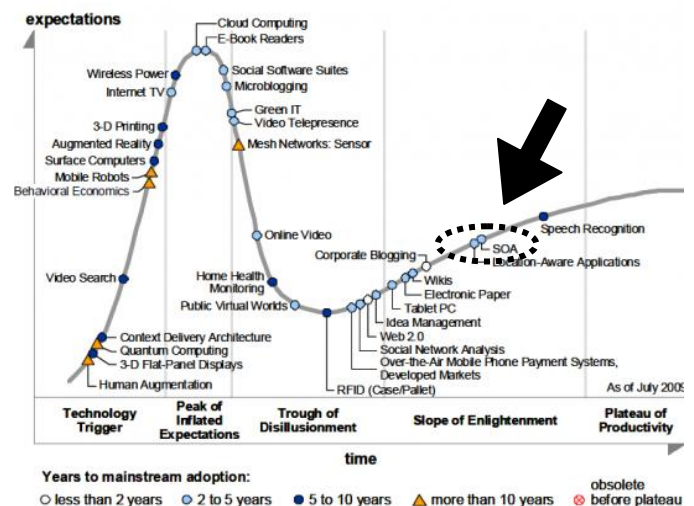


Figure 1. Adapted from Emerging Technologies Hype Cycle [10]

The development of services for an SOA involves adherence to certain protocols and guidelines defined for the specific architecture (Figure 2). The provision of services that provide core business functionality is the responsibility of a Service Provider.

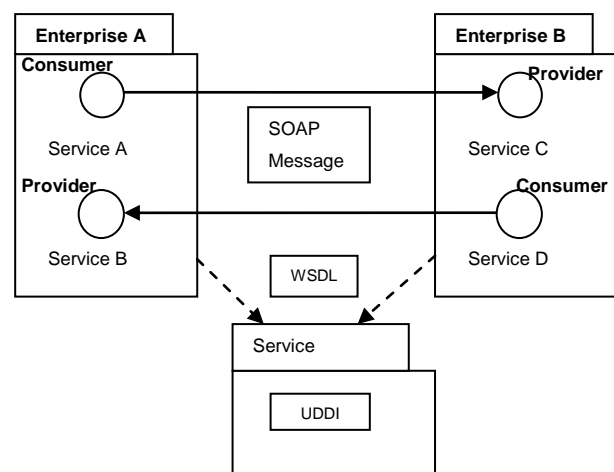


Figure 2. Adapted from "A Sample SOA Environment" [12]

These services are also required by other services or subcontractors, who are Service Consumers (Figure 2). Thus the SOA is a combination of service providers and services consumers providing and requesting services, for which all information is stored, and made accessible to actors in a Service Registry. The architecture of an SOA environment depicts the relationship between the service provider, service

consumer and service registry (Figure 2). Services share information amongst one another via Simple Object Access Protocol (*SOAP*) messages and information stored in the service registry is based on Universal Description, Discovery and Integration (*UDDI*). The interface between service providers, or consumers, and the service registry can be built based on the Web Service Description Language (*WSDL*) standards.

The functional attributes of services are deciding factors for service consumers when requesting services from service providers [11]. These attributes need to be considered along with non-functional Quality of Service (QoS) attributes. QoS metrics are essential when selecting the best possible execution plan for legacy system modernization with respect to budget and time constraints. QoS metrics define a useful measure for service comparison during the service provisioning process. During the development of services, the effort required by the developer to modernize legacy code to conform to SOA requirements is a useful measure. Software metrics for Object-Oriented (OO) applications are useful for the analysis of the developer's effort to create the service [6].

The modernization of a legacy system into services that perform the same business functionality, but operate in a modern architecture leads to another possible metric for evaluation. Services need to be assessed as a feasible replacement option for the legacy system [5]. An empirical approach can be used to study users' interactions with the services to determine their efficacy [18].

The combination of the QoS evaluation, developer effort evaluation and end product efficacy evaluation leads to a holistic evaluation strategy. The outcomes of the application of this comprehensive evaluation strategy potentially provide results that can be correlated to provide evidence for successful legacy system modernization.

II. BACKGROUND

Understanding the process of and necessity for migrating legacy code to a more modern service-related architecture requires the recognition of concepts such as legacy system modernization and data services. Data is provided as a service, thereby increasing the longevity of the core functionalities of the legacy system.

A. Legacy System Modernization

Since legacy systems contain functionality that is of utmost importance for business longevity, it is essential that they are evolved to integrate with modern platforms and architectures, such as SOA [9]. Modernization covers a broader range of changes to an existing system than the process of system maintenance and is a possible solution for the elongation of a legacy system's lifespan [16]. These changes include restructuring the system, improving system functionality or modifying system attributes. Modernization must, however, conserve a sizable portion of the existing legacy system to conserve the original business rules contained in the system [8]. Modernization approaches typically fall into two distinct categories [9], namely:

- Legacy Integration and Service Enablement; and
- Legacy Transformation.

Legacy integration uses non-invasive wrapping of legacy systems to hide complexity and emphasises modern interfaces to improve interoperability (Figure 3). This category of modernization is used to lengthen the lifetime of legacy systems by exposing the integral functionality of these systems. Consequently, legacy wrapping reduces the cost of integration whilst requiring less immediate planning and design.

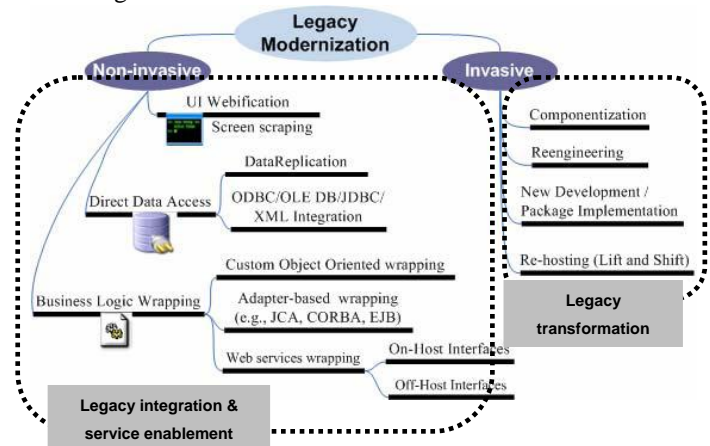


Figure 3. Adapted from “Taxonomy of Legacy Modernization Approaches” [9]

Legacy transformation follows an invasive re-engineering approach to convert legacy systems (Figure 3). A detailed analysis of the existing legacy code and an understanding of the system functionality and data structures lead to the extraction of data definitions and business rules.

Modernization can be classified by the degree of knowledge of system internals required to sustain the modernization approach [8]. Modernization that requires understanding of the functionality and structure of a legacy system is known as white-box modernization, and modernization that only requires knowledge of the legacy system interfaces is called black-box modernization.

White-box modernization requires an initial reverse-engineering process to gain understanding of the legacy system internal structure and operation [8]. This process is called *program understanding* and involves modelling the domain, extracting information from the legacy code and creating representations of the system hierarchy [16]. After program understanding, restructuring of the system defines a transformation from one representation of the system to another at the same level of abstraction [8]. This process typically alters quality attributes of the system including maintenance and performance. White-box modernization is beneficial for simplified future maintenance and extension [9].

Alternatively, black-box modernization is concerned with the evaluation of the inputs and outputs of the legacy system during operation to gain knowledge of the interfaces [16]. Black-box modernization is usually a less complex task than white-box modernization as all it involves is the wrapping of the original system. Wrapping entails encasing the legacy

system in a software layer that masks the unnecessary complexities of the legacy code whilst generating a modern interface. The wrapper interfaces with the legacy system during execution of every possible interaction instance [3]. A wrapping approach is termed black-box since after the legacy interface is analysed, the internals are disregarded [16].

There are many advantages for incorporating wrappers into legacy applications [17]. Firstly, the legacy database is not altered in any way. Secondly, the wrapper allows for more functionality to be added to existing legacy applications, such as statistics collection and performance visualization. Lastly, wrappers allow for an incremental modernization process of complex legacy systems. Alternatively, a disadvantage of black-box modernization is the manual recording of interaction scenarios and screen templates [3]. A black-box solution is not always realistic as it sometimes requires knowledge of some system internals by use of white-box techniques [8].

Each of the modernization approaches has benefits in specific domains. However, there is a lack of a comprehensive evaluation of the modernization process that identifies which approach is successful in the modernization of a legacy system into data services.

B. Data Services

Services that provide data in a uniform structure are known as Data Services [4]. These data services have the essential functionality of a distinct business object. Thus, the structure of the data service is determined by and describes the information stored within it for its specific business object type.

Data services have a collection of read methods which are calls to the service to request access to instances of business object types in various ways [2]. Similarly, data services have write methods which allow for the insertion, modification, or deletion of different business object instances. Data services also have navigation methods which are service requests to navigate the relationships between business objects returned via different services.

Importance of quality metrics of services, such as reusability and performance, is stressed for improved service discovery [11]. Several quality attributes are described in existing evaluation strategies that need to be adhered to when generating services for an SOA [1]. These non-functional attributes serve as a guideline for the development of high quality services. Software metrics are used to quantitatively measure quality attributes [6].

III. EVALUATION STRATEGIES

Numerous evaluation strategies exist in the form of guidelines for the development of services for an SOA as well as for the evaluation of the end products of development. These strategies include the adherence to QoS requirements [1], measurement of software metrics to assess code complexity and developer effort [6], and empirical

evaluation to gauge the acceptance of the end product with users of the software [18].

A. Quality of Service Attributes

QoS metrics are essential in selecting the best possible execution plan for service discovery and composition within budget and time limitations [11]. A number of quality attributes are relevant to the evaluation of the end product of the modernization process of a legacy system [1, 11], namely:

- Performance;
- Availability;
- Security;
- Testability;
- Interoperability; and
- Reliability.

Performance is measured as the time taken for responses to be generated from requests sent to a service [1], and is indicative of the cost of invoking a service [11]. Availability is defined as a service's ability to be available despite server maintenance or system overload. Security is defined in terms of the confidentiality and protection of data when accessed by a service. A service is required to be testable for modified versions of the service. A service is required to be interoperable on various platforms, the level of interoperability being defined in terms of its reusability. Software metrics can be used to measure reusability of code in a system [6]. The reliability of the service is measured by the amount of system errors encountered and the recovery of the system as a result [11].

B. Object-Oriented Software Metrics

A suite of metrics for object-oriented (OO) code exist based on measurement theory [6]. These metrics were developed to be high level and therefore independent of any programming language. The metrics that will be focused on for the purposes of this study include:

- WMC (Weighted Methods per Class);
- DIT (Depth of Inheritance Tree);
- NOC (Number of Children); and
- CBO (Coupling Between Objects).

The use of these candidate metrics provides insight into the complexity of the code, the effort required by the developer to create and maintain the OO system, as well as effects on quality attributes pertaining to the system.

The effort required by the developer to create and maintain a class is described by the WMC metric [6]. The number of methods per class in an OO system, combined with the complexity of these methods is a clear indication of the amount of time and effort required by the developer to create the class. With regard to the DIT metric, the deeper a class is in a hierarchy of classes, the greater the number of methods it inherits from its hierarchy [6]. This inheritance factor increases the complexity of the class and decreases the ability to predict the behaviour of the class. Alternatively, a greater depth in the inheritance tree implies better reusability of methods.

The NOC metric measures the number of subclasses inheriting the methods of a parent class, namely the scope of

its properties [6]. The greater the number of subclasses, the better the reuse of methods from the parent class due to the inheritance property.

The CBO metric provides an indication of the comprehension of the actions of one object on another object due to the former object's calls or references to the latter [6]. If the CBO measure is too high, the reusability and modularity of the class is negatively affected. By minimizing coupling, it is possible to improve maintainability of the code. Thus, reduced changes are required to other sections of the system code.

C. Efficacy of Data Services

Many guidelines for the development of high quality services have been suggested in order to assist the developer in the conversion process associated with the modernization of legacy systems. Once the service has been deployed, however, the user requires an effective and efficient means of carrying out the core functionality for which they require the service. Effectiveness is defined as a general goal as to how well a product performs the task for which it has been created [14].

Empirical research in the form of *usability studies* [14] can be used to determine the efficacy of a service, since the service is software with which the user interacts. This strategy is apparent in the case where empirical evaluations are used to support the comparative analysis between an existing legacy system modernization tool and MELIS, a proposed legacy system migration tool [7]. Usability testing is defined as the evaluation approach that involves measuring users' performance and analysing their satisfaction with the system being tested in a formal controlled environment [18].

Quantitative methods are defined as collecting numerical data and analysing it by statistical procedures [15]. Qualitative data is collected as text or images from observations, interviews or questionnaires. Results from statistical analysis of data collected from usability experiments are combined with the participants' feedback to validate the results, seen in similar studies [7].

IV. COMPREHENSIVE EVALUATION STRATEGY

Various evaluation strategies exist and are employed to measure the quality of software produced [1, 5, 6, 11] as well as the satisfaction that the end product provides the customer [18]. A comprehensive evaluation strategy is proposed. The approach evaluates the design, implementation and outputs of a modernization process to aid in the assurance of a thorough modernization process and the precise performance of the outputs after modernization [5]. The holistic modernization evaluation strategy is comprised of three different criteria:

1. QoS attributes of the services generated;
2. Developer effort required to modernize legacy system; and
3. Efficacy evaluation of the service generated.

To validate this evaluation strategy, the three evaluation legs will be applied to a case study. The legacy system in the

case study allows student assistants at a university to maintain their information, select courses to assist and mark their attendance for sessions in the courses they assist. This legacy system went through a white-box modernization process to expose the data as a service, using the design principles shown in similar studies [2]. The data services were created to retrieve data from the database with read methods such as *getStudentInfo* and *fillDetails* (Table 1). Data services also contained write methods such as *updateDemi* (Table 1).

A consolidated list of QoS attributes can be drawn from combining those presented (Section III). The most appropriate and frequently used QoS metrics are used as a guideline for service generation from the modernization approach [1, 2, 11]:

6. Performance;
7. Reliability;
8. Interoperability; and
9. Reusability.

Testability is excluded from the four final attributes as it is identified as a minor quality attribute [11]. Security and availability are excluded from the elected QoS metrics due to the nature of the case study. The student assistant system operates within a closed network where data security is not a necessity. The availability of the services despite server overload is not a concern in this study as it is not likely that the data services would be queried simultaneously by several users.

Service Methods	Mean Time (s)	Std Dev
<i>checkStudent</i>	0.417	0.051
<i>getStudentInfo</i>	0.045	0.002
<i>fillNoticeBoard1</i>	0.045	0.005
<i>fillNoticeBoard2</i>	0.074	0.005
<i>getMessages</i>	0.044	0.002
<i>fillDetails</i>	0.062	0.005
<i>updateDemi</i>	0.067	0.009
<i>getSessionTableData</i>	0.069	0.026
<i>getSessionColumnData</i>	0.030	0.003
<i>getHoursTableData</i>	0.064	0.004
<i>getHoursColumnData</i>	0.035	0.006

Table 1. Mean Response Times for Service Calls

Performance of the data services created during modernization was measured in seconds as each service method was called by the client application. The service methods were tested by seven participants and the mean and standard deviations were recorded (Table 1). Considering that the services are called with a SOAP request, perform a SQL query to retrieve data and finally return the data via a SOAP response to the client, the performance results are consistently low, with the exception of the *checkStudent* method. This method is called first during login to the system and the connection to the data source is first made here, explaining the excess time taken for the service to respond. Participants noted that the response times of the data services were no different than the performance of the legacy system (n = 3).

Reliability of the data sent to and retrieved by the services was dealt with in the service client. Checks are performed on all data to ensure that it is meaningful and the system provides users with responsive feedback for the actions which they perform. Finally, the interoperability of the services describes the reusability of the services on any platform. To ensure reusability of the services generated from modernization, the services were written with Java and deployed on a Glassfish application server using SOAP standards. The nature of SOAP services is inherently platform independent due to the exchange of messages in SOAP format using XML. The service client classes maintain the states of the interface during the application's execution (Table 2), whilst the data services contain the methods to access and modify data and the service methods are called by the service client classes as needed.

	Home	Details	PrintSession	PrintHours	SessionModel	HoursModel
WMC	22	22	11	16	8	7
# Methods	11	6	6	8	6	6
DIT	0	0	0	1	1	1
NOC	0	0	0	0	0	0
CBO	2	1	2	1	0	0

Table 2. Metrics for Classes in the Service Client

The DIT and NOC metrics are useful measures for the verification of code reusability. The calculation of the depth of a class in a class hierarchy reveals the level of method reuse due to inheritance [6]. The reusability of code in the client application classes is low; however, low DIT and NOC values also indicate reduced complexity and increased prediction of behaviour of the classes (Table 2). The CBO metrics collected also prove useful in determining the reusability of classes of code in the application. The *Home* and *PrintSession* classes show the highest values for coupling, thus increasing their reusability slightly.

The WMC metric provides a measure of effort required by the developer to create the data services from the legacy code during modernization. Thus, the number of methods per class and their perceived complexity provides an indication of the time and effort to create and maintain the code [6]. The WMC metrics for the classes as well as the number of methods per class are indicated in Table 2. The complexities of the methods are relatively low amongst the classes in comparison to the number of methods per class, thus improving the quality of the system. This reduced complexity of the service client is due to the incorporation of data services for any data retrieval or insertion (Table 1).

Reverse-engineering rules applied to legacy systems require validation by empirical research [5]. The modernization of legacy systems of various designs produces various end results. Empirical research performed in a similar environment to which the results will be applied with realistic subjects is relevant [15]. Admittedly, this form of research may be time-consuming and costly, but the industrial benefit lies in the implication of the results. After modernization and software metric evaluations were completed, a usability study was performed to obtain quantitative and qualitative feedback from representative users of the system. To identify any usability issues, a

sample of seven participants was selected to perform three tasks on the modernized student assistant service. The participants consisted of six male and one female student assistant, who were familiar with the legacy system as they had used it at least once a week previously ($n = 7$). Participants' times taken per task were recorded to assess the efficiency of the data services. The times per task were consistently low for all tasks, at less than one minute (Table 3). The mean times per task are an indication of the efficiency of the services to provide timely responses to the users.

	Mean	Std Dev
Task 1	38.51	7.14
Task 2	20.36	7.15
Task 3	35.96	10.83

Table 3. Participants Mean Times per Task

	Frequency
Task 1	0
Task 2	0
Task 3	5

Table 4. Error Rates per Task

The effectiveness of the system to provide users with the data they requested was assessed by retrieving task success and task completion measures. All participants were able to complete each of the three tasks, although task three produced high error rates (Table 4). Whilst executing task three, participants struggled to navigate to the interface to print the total hours of service. During modernization, the interface was directly translated from the legacy system, thus indicating design flaw in the original system as well. A recommendation from this observation is to improve the visibility of the option to print the total hours of service, thus improving navigability of the modernized system.

Lastly, a post-test System Usability Scale (SUS) was performed to determine system acceptance as a whole by participants, identified by unique participant ID's (Table 5). The scores for each of the 10 questions were normalized and collated into a final score.

	P01	P02	P03	P04	P05	P06	P07	Mean
Score	92.5	92.5	90	100	82.5	75	77.5	87.14

Table 5. SUS Scores per Participant

Consistently high scores were given by each of the participants, indicating a high acceptance rate for the modernized system. Combined with encouraging user comments and the system's efficiency and effectiveness to provide users with the data they requested, it can be seen that the data services are a feasible replacement for the legacy system.

V. CONCLUSION

Due to the growing enlightenment on SOA technologies [10], there is an opportunity for organisations to use these emerging technologies as a platform to modernize legacy systems. The benefit of the modernization process is the lengthened lifespan of core business processes within existing legacy systems. Once data services have been

generated by modernization, it is necessary to evaluate these services. A holistic evaluation strategy for legacy system modernization consisting of three evaluation legs is proposed. Firstly, QoS attributes are defined as a guideline for development of data services. The second leg involves the prediction effort required by the developer to generate the data services during modernization. Thirdly, the data services must be evaluated by empirical studies to gauge performance of the service as well as user satisfaction.

Existing evaluation strategies include the service's adherence to QoS guidelines by the identification of major quality attributes of services for an SOA [11]. Also, the use of software metrics during the development process to ensure software of a high quality is identified [6]. Lastly, user studies have been performed to assess performance metrics as well as gain feedback regarding qualitative measures of software end products [7]. By combining these existing evaluation strategies to produce a comprehensive evaluation framework, the overall success of a modernization approach can be gauged.

The application of this evaluation approach to the case study in this paper demonstrated how to use QoS metrics as a guideline for the development of high quality data services. The extent to which the guidelines were met was measured by the use of software metrics (Section IV). These analytical metrics validated the proper design of the modernized data services. These metrics alone, however, don't give an overall view of the usefulness of the new system. To measure the effectiveness of the data services, usability tests were conducted. The results obtained from the usability tests indicated a high acceptance rate by participants. The combination of the analytical evaluation results and the usability testing results illustrate the success of the modernization approach applied to the case study.

Perceived benefits of this combined modernization evaluation approach include the collection of data throughout all phases of development. Guidelines have been recognised that need to be adhered to. These guidelines provide rules for the development of high quality services. Software metrics provide a quantitative measure for the evaluation of the modernization outputs. The developer's effort measure could be beneficial in the determination of which modernization approach is more suitable for the migration of legacy code to data services. Lastly, user studies of the services generated allow for the assessment of the final product's efficacy and the user's satisfaction. This holistic evaluation approach aims to ensure the successful outcome of legacy system modernization.

The comprehensive evaluation strategy was applied to a white-box modernization case study. The results from this pilot study were feasible, showing that the modernization of the legacy system into data services was an efficient and acceptable process. The analytical evaluation of the software metrics and quality of the service also yielded positive results regarding the design of the data services. The success

of the modernization effort has served as a validation of the selection of the criteria for this evaluation strategy.

REFERENCES

- [1] Bianco, P., Kotermanski, R. & Merson, P. (2007). Evaluating a Service-Oriented Architecture. Carnegie Mellon University.
- [2] Borkar, V., Carey, M., Lychagin, D., Westmann, T., Engovatov, D. & Onose, N. (2006). Query Processing in the AquaLogic Data Services Platform. International Conference on Very Large Data Bases (VLDB'06).
- [3] Canfora, G., Fasolino, A., Frattolillo, G. & Tramontana, P. (2006). Migrating Interactive Legacy Systems to Web Services. *Conference on Software Maintenance and Reengineering*.
- [4] Carey, M. (2006). Data Delivery in a Service-Oriented World: The BEA AquaLogic Data Services Platform. International Conference on Management of Data.
- [5] Chiang, R.H.L., Barron, T.M., Storey, V.C. (1997). A Framework for the Design and Evaluation of Reverse Engineering Methods for Relational Databases. *Data and Knowledge Engineering*, **21**, 57-77.
- [6] Chidamber, S., Kemerer, C. (1994). A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, **20**, 476 – 493.
- [7] Colosimo, M., De Lucia, A., Scanniello, G. & Tortora, G. (2008). Evaluating legacy system migration technologies through empirical studies. *Information and Software Technology*, **51**, 433-447.
- [8] Comella-Dorda, S., Seacord, R.C., Wallnau, K., Robert, J. (2000). A Survey of Black-Box Modernization Approaches for Information Systems. *International Conference on Software Maintenance*.
- [9] Erradi, A., Anand, S., Kulkarni, N. (2006). Evaluation of Strategies for Integrating Legacy Applications as Services in a Service Oriented Architecture. *International Conference on Services Computing*.
- [10] Gartner Group. (2009). Hype Cycle for Emerging Technologies, 21 July 2009.
- [11] Jeong, B., Hyunbo, C., Choonghyun, L. (2009). On the Functional Quality of Service (FQoS) to Discover and Compose Interoperable Web Services. *Expert Systems with Applications*, **36**, 5411-5418.
- [12] Liegl, P. (2007). The Strategic Impact of Service Oriented Architecture. *International Conference and Workshops on the Engineering of Computer-Based Systems*.
- [13] Sanders, D., Hamilton, J. & Macdonald, R. (2008). Supporting a Service-Oriented Architecture. *Spring Simulation Multiconference*.
- [14] Sharp, H., Rogers, Y., Preece, J. (2007). Interaction Design: Beyond Human-Computer Interaction, John Wiley & Sons.
- [15] Sjøberg, D.I.G., Dybå, T., Jørgensen, M. (2007). The Future of Empirical Methods in Software Engineering Research. *Future of Software Engineering (FOSE '07)*.
- [16] Seacord, R. C., Plakosh, D. & Lewis, G. (2003). Modernizing Legacy Systems Software Technologies, Engineering Processes, and Business Practices, Addison-Wesley.
- [17] Thiran, P., Hainaut, J., Houben, G., Benslimane, D., (2006) Wrapper-based Evolution of Legacy Information Systems. *ACM Transactions on Software Engineering and Methodology*. 329-359.
- [18] Tullis, T., Albert, B. (2008). Measuring the User Experience, Morgan Kaufmann.

Meredith A. Barnes received her BSc and BSc (Hons) in Computer Science and Applied Mathematics from Nelson Mandela Metropolitan University (NMMU). She is currently pursuing a MSc in Computing Sciences at NMMU.

Demi Service Task List (Framework Validation)

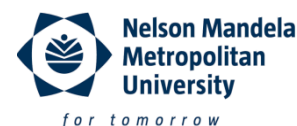
Task 1
<p>Log in to the Demi System using your student number</p> <p>Select the “Change Details” Option</p> <p>Increase the value for your year level by one</p> <p>Select the “Save” button and follow the prompts to save your changes</p> <p>Once back on the Home interface, reopen the “Change Details” interface to see if your changes were saved</p> <p>Select the “Close” button</p>
Task 2
<p>Select the “Print Session List” button</p> <p>Select the option to print your session list and follow the prompts to use the default printer</p>
Task 3
<p>Select the “View Total Hours” button</p> <p>Change the first date to 1 May 2009 and view the total hours for your date selection</p> <p>Select the option to print your table and follow the prompts to use the default printer</p> <p>Close the Page and return to the Home page</p> <p>Close the homepage</p>

Nelson Mandela Metropolitan University

Department of Computing Sciences

This questionnaire is part of research towards a MSc in
Computing Sciences

Contact Information: Email: Meredith.Barnes@nmmu.ac.za; Phone: 041 504 2094



SYSTEM 1 Task List

Task 1 : Student number is 209012345

1.1	Use the Demi System to complete the Demi Application Form (Use the provided information below)	
	ID Number	8701010083980
	Title	Mr
	Name	Peter
	Surname	Smith
	Phone number	083 123 4567
	Address	123 Green Street Parkville Port Elizabeth
	Correspondence language	English
	Can speak fluently	English
	Qualification	BSc
	Year level	3
	Period available	Entire Year
	Previous experience	None
	Have you applied for SI leader?	No
	What courses?	Leave blank
	Comments	Leave blank
1.2	Once the application form is complete, send the verification email to: charmain.cilliers@nmmu.ac.za	
1.3	Save your application and close the form	
1.4	Close the Demi System	
1.5	Press F10 to move to the next task	

Task 2 : Student number is 209012345

2.1	Use the Demi System to select the WRFE101 session from the list of available sessions	
2.2	Select the WRA102 session from the list of available sessions	
2.3	Save the session selection	
2.4	What labs are your chosen sessions held in?	
	WRFE101	<i>Answer:</i>
	WRA102	<i>Answer:</i>
2.5	Close the Demi System	
2.6	Press F10 to move to the next task	

Task 3 : Student number is 209012345

3.1	Use the Demi System to remove the WRFE101 session from your list of selected sessions	
3.2	Save the session removal	
3.3	What lab is your remaining session held in?	
	WRA102	<i>Answer:</i>
3.4	Close the Demi System	
3.5	Press F10 to move to the next task	

Task 4 : Student number is 209012345

4.1	Use the Demi System to view your list of selected sessions	
4.2	From the list of sessions please answer the following questions:	
4.2.1	What session are you registered to demi?	<i>Answer:</i>
4.2.2	What day do you need to demi this session?	<i>Answer:</i>
4.3	Close the Demi System	
4.4	Press F10 to move to the next task	

Please complete this section after you have completed the tasks for System 1

Put a cross [X] in the appropriate block

1.	I think that I would like to use System 1 frequently					
Strongly disagree	1	2	3	4	5	Strongly agree
2.	I found System 1 unnecessarily complex					
Strongly disagree	1	2	3	4	5	Strongly agree
3.	I thought System 1 was easy to use					
Strongly disagree	1	2	3	4	5	Strongly agree
4.	I think that I would need the support of a technical person to be able to use System 1					
Strongly disagree	1	2	3	4	5	Strongly agree
5.	I found the various functions in System 1 were well integrated					
Strongly disagree	1	2	3	4	5	Strongly agree
6.	I thought there was too much inconsistency in System 1					
Strongly disagree	1	2	3	4	5	Strongly agree
7.	I would imagine that most people would learn to use System 1 very quickly					
Strongly disagree	1	2	3	4	5	Strongly agree
8.	I found System 1 very cumbersome to use					
Strongly disagree	1	2	3	4	5	Strongly agree
9.	I felt very confident using System 1					
Strongly disagree	1	2	3	4	5	Strongly agree
10.	I needed to learn a lot of things before I could get going with System 1					
Strongly disagree	1	2	3	4	5	Strongly agree
11.	Please list any features that you liked the <i>most</i> about System 1 and explain why:					
12.	Please list any features that you liked the <i>least</i> about System 1 and explain why:					

SYSTEM 2 Task List

Task 1 : Student number is 209055667

1.1	Use the Demi System to complete the Demi Application Form (Use the provided information)	
	ID Number	8701010084680
	Title	Miss
	Name	Mary
	Surname	Miller
	Phone number	083 765 4321
	Address	10 Long Street Greenfields Port Elizabeth
	Correspondence language	English
	Can speak fluently	English, Afrikaans
	Qualification	BComm
	Year level	3
	Period available	Entire Year
	Previous experience	None
	Have you applied for SI leader?	No
	What courses?	Leave blank
	Comments	Leave blank
1.2	Once the application form is complete, send the verification email to: jean.greyling@nmmu.ac.za	
1.3	Save your application and close the form	
1.4	Close the Demi System	
1.5	Press F10 to move to the next task	

Task 2 : Student number is 209055667

2.1	Use the Demi System to select the WRFC102 session from the list of available sessions	
2.2	Select the WRA202 session from the list of available sessions	
2.3	Save the session selection	
2.4	What labs are your chosen sessions held in?	
	WRFC102	<i>Answer:</i>
	WRA202	<i>Answer:</i>
2.5	Close the Demi System	
2.6	Press F10 to move to the next task	

Task 3 : Student number is 209055667

3.1	Use the Demi System to remove the WRA202 session from your list of selected sessions	
3.2	Save the session removal	
3.3	What lab is your remaining session held in?	
	WRFC102	<i>Answer:</i>
3.4	Close the Demi System	
3.5	Press F10 to move to the next task	

Task 4 : Student number is 209055667

4.1	Use the Demi System to view your list of selected sessions	
4.2	From the list of sessions please answer the following questions:	
4.2.1	What session are you registered to demi?	<i>Answer:</i>
4.2.2	What day do you need to demi this session?	<i>Answer:</i>
4.3	Close the Demi System	
4.4	Press F10 to move to the next task	

Please complete this section after you have completed the tasks for System 2

Put a cross [X] in the appropriate block

1.	I think that I would like to use System 2 frequently					
Strongly disagree	1	2	3	4	5	Strongly agree
2.	I found System 2 unnecessarily complex					
Strongly disagree	1	2	3	4	5	Strongly agree
3.	I thought System 2 was easy to use					
Strongly disagree	1	2	3	4	5	Strongly agree
4.	I think that I would need the support of a technical person to be able to use System 2					
Strongly disagree	1	2	3	4	5	Strongly agree
5.	I found the various functions in System 2 were well integrated					
Strongly disagree	1	2	3	4	5	Strongly agree
6.	I thought there was too much inconsistency in System 2					
Strongly disagree	1	2	3	4	5	Strongly agree
7.	I would imagine that most people would learn to use System 2 very quickly					
Strongly disagree	1	2	3	4	5	Strongly agree
8.	I found System 2 very cumbersome to use					
Strongly disagree	1	2	3	4	5	Strongly agree
9.	I felt very confident using System 2					
Strongly disagree	1	2	3	4	5	Strongly agree
10.	I needed to learn a lot of things before I could get going with System 2					
Strongly disagree	1	2	3	4	5	Strongly agree
11.	Please list any features that you liked the <i>most</i> about System 2 and explain why:					
12.	Please list any features that you liked the <i>least</i> about System 2 and explain why:					

Post-Test Questionnaire						
Put a cross [X] in the appropriate block						
1.	Which system would you prefer to use more frequently?					
System 1	1	2	3	4	5	System 2
Why would you prefer to use this system more frequently?						
2.	Which system did you find more complex?					
System 1	1	2	3	4	5	System 2
Why did you find this system more complex?						
3.	Which system did you find easier to use?					
System 1	1	2	3	4	5	System 2
Why was this system easier to use?						
4.	Which system do you think you would need support of a technical person to use?					
System 1	1	2	3	4	5	System 2
Why did you feel you needed the support of a technical person for this system?						
5.	Which system's various functions did you find were better integrated?					
System 1	1	2	3	4	5	System 2
Why did you feel this system was better integrated?						
6.	Which system did you think was more inconsistent?					
System 1	1	2	3	4	5	System 2
Why did you think this system was more inconsistent?						
7.	Which system do you think most people would learn to use more quickly?					
System 1	1	2	3	4	5	System 2
Why do you think people will learn this system more quickly?						
8.	Which system did you find more cumbersome to use?					
System 1	1	2	3	4	5	System 2
Why did you find this system more cumbersome?						
9.	Which system did you feel more confident using?					
System 1	1	2	3	4	5	System 2
Why did you feel more confident using this system?						
10.	Which system did you feel required you to learn a lot more before you started?					
System 1	1	2	3	4	5	System 2
Why did you feel that you needed to learn more about this system first?						

Task 1 ANOVA

	SS	D.F.	MS	F	p
Intercept	13020.83	1	13020.83	10.46	.003
BW	1140.83	1	1140.83	0.92	.347

	Level	n	Mean	S.D.	95%CI.lo	95%CI.hi
Total		30	-20.83	35.24	-33.99	-7.68
BW	W-B	15	-14.67	31.35	-32.03	2.70
BW	B-W	15	-27.00	38.83	-48.50	-5.50

Task 2 ANOVA

	SS	D.F.	MS	F	p
Intercept	1920.00	1	1920.00	4.73	.038
BW	1228.80	1	1228.80	3.03	.093

	Level	n	Mean	S.D.	95%CI.lo	95%CI.hi
Total		30	-8.00	20.84	-15.78	-0.22
BW	W-B	15	-14.40	24.26	-27.84	-0.96
BW	B-W	15	-1.60	14.93	-9.87	6.67

Task 3 ANOVA

	SS	D.F.	MS	F	p
Intercept	83.33	1	83.33	0.85	.365
BW	496.13	1	496.13	5.04	.033

	Level	n	Mean	S.D.	95%CI.lo	95%CI.hi
Total		30	-1.67	10.59	-5.62	2.29
BW	W-B	15	2.40	7.57	-1.79	6.59
BW	B-W	15	-5.73	11.82	-12.28	0.81

Task 4 ANOVA

	SS	D.F.	MS	F	p
Intercept	607.50	1	607.50	1.00	.326
BW	5964.30	1	5964.30	9.80	.004

	Level	n	Mean	S.D.	95%CI.lo	95%CI.hi
Total		30	-4.50	28.16	-15.02	6.02
BW	W-B	15	9.60	21.50	-2.30	21.50
BW	B-W	15	-18.60	27.47	-33.81	-3.39

Task 5 ANOVA

	SS	D.F.	MS	F	p
Intercept	770.13	1	770.13	2.26	.144
BW	2764.80	1	2764.80	8.13	.008

	Level	n	Mean	S.D.	95%CI.lo	95%CI.hi
Total		30	5.07	20.59	-2.62	12.75
BW	W-B	15	-4.53	14.45	-12.54	3.47
BW	B-W	15	14.67	21.72	2.64	26.69

**Positive SUS
ANOVA**

	SS	D.F.	MS	F	p
Intercept	3.01	1; 28	3.01	6.62	.016
BW	0.01	1; 28	0.01	0.02	.893

	Level	n	Mean	S.D.	95%CI.lo	95%CI.hi
Total		30	-0.3167	0.6628	-0.5642	-0.0692
BW	W-B	15	-0.3000	0.5278	-0.5923	-0.0077
BW	B-W	15	-0.3333	0.7943	-0.7732	0.1065

SUS ANOVA

	SS	D.F.	MS	F	p
Intercept	1.01	1; 28	1.01	5.06	.033
BW	0.03	1; 28	0.03	0.17	.686

	Level	n	Mean	S.D.	95%CI.lo	95%CI.hi
Total		30	-0.1833	0.4401	-0.3477	-0.0190
BW	W-B	15	-0.1500	0.3510	-0.3444	0.0444
BW	B-W	15	-0.2167	0.5250	-0.5074	0.0741