A novel model for improving the maintainability of web-based systems.

Emad Ghosheh

School of Electronics and Computer Science

Ph.D. Thesis

A Novel Model for Improving the Maintainability of Web-Based Systems

EMAD GHOSHEH

A thesis submitted in partial fulfilment of the requirements of the University of Westminster for the degree of Doctor of Philosophy

June 2010

Acknowledgements

I would like to thank all people who have helped and inspired me during my doctoral study. I would like to express my deep and sincere gratitude to my Director of Study, Professor Sue Black, Head of Department of Information and Software Systems, University of Westminster. Her wide knowledge and her logical way of thinking have been of great value for me. Her understanding, encouraging and personal guidance have provided a good basis for the present thesis.

I am deeply grateful to my supervisors, Professor Vladimir Getov, and Professor Epaminondas Kapetanios for their support in my research.

I wish to express my warm and sincere thanks to Professor Jihad Qaddour, School of Information Technology, Illinois State University who helped me in conducting the empirical experiments at Illinois State University in the USA.

I am deeply thankful to Mark Baldwin, Principal Lecturer in the Department of Information at the University of Westminster for his help and review of the statistical results reported in this research.

I cannot end without thanking my family, on whose constant encouragement and love I have relied throughout my time working on my PhD. Special thanks for my parents, wife Rana, son Jawad, daughter Ayisha, and son Noorideen. It is to them that I dedicate this work.

Portions of this thesis have appeared in the following publications

- **Journals**

  [1] E. Ghosheh and S. Black and E. Kapetanios and M. Baldwin. Exploring the Relationship between UML Design Metrics for Web Applications and Maintain-

ability. Journal of Object Technology JOT, published by ETH Swiss Federal Institute of Technology, vol. 9, no. 3, May- June 2010, pp. 125-144.

[2] E. Ghosheh and S. Black. Empirical validation of UML class diagram metrics through an industrial case study. Journal of Electronics & Computer Science $JECS$, vol. 10, no. 4, pp. 63-74, 2008.

[3] E. Ghosheh and S. Black and J. Qaddour. An introduction of new UML design metrics for web applications. International Journal of Computer & Information Science, 8(4):600-609, 2007.

- **Book Chapters**

[1] E. Ghosheh, S. Black, and J. Qaddour. An industrial study using UML design metrics for web applications. In Computer and Information Science, volume 131 of Studies in Computational Intelligence, chapter 20, pages 231- 241. Springer-Verlag, 2008.

- **Conferences**

[1] E. Ghosheh, S. Black, Wapmetrics: a tool for computing UML design metrics for web applications, in Proceedings of the 7th IEEE/ACS International Conference on Computer Systems and Applications, pages 682-689. IEEE Computer Society Press, 2009.

[2] E. Ghosheh, S. Black, and J. Qaddour, A general evaluation criteria for web applications maintainability models, in Proceedings of the IEEE Region 5 Technical, Professional, and Student Conference, pages 1-6. IEEE Computer Society Press, 2008.

[3] E. Ghosheh, S. Black, and J. Qaddour. Design metrics for web application maintainability measurement. In Proceedings of the 6th ACS/IEEE International Conference on Computer Systems and Applications, pages 778-784. IEEE Computer Society Press, 2008.

[4] E. Ghosheh, J. Qaddour, M. Kuofie, and S. Black. A comparative analysis of maintainability approaches for web applications. In Proceedings of the 4th AC-

S/IEEE International Conference on Computer Systems and Applications, page 247. IEEE Computer Society Press, 2006.

- **Workshops**

[1] E. Ghosheh, S. Black, and J. Qaddour. An introduction of UML design metrics for web applications. In Proceedings of the Annual Mphil-PhD Research Workshop, pages 38-41. Harrow School of Computer Science University of Westminster, 2007.

Abstract

Web applications incorporate important business assets and offer a convenient way for businesses to promote their services through the internet. Many of these web applications have evolved from simple HTML pages to complex applications that have a high maintenance cost. This is due to the inherent characteristics of web applications, to the fast internet evolution and to the pressing market which imposes short development cycles and frequent modifications. In order to control the maintenance cost, quantitative metrics and models for predicting web applications' maintainability must be used. Maintainability metrics and models can be useful for predicting maintenance cost, risky components and can help in assessing and choosing between different software artifacts. Since, web applications are different from traditional software systems, models and metrics for traditional systems can not be applied with confidence to web applications. Web applications have special features such as hypertext structure, dynamic code generation and heterogenousity that can not be captured by traditional and object-oriented metrics.

This research explores empirically the relationships between new UML design metrics based on Conallen's extension for web applications and maintainability. UML web design metrics are used to gauge whether the maintainability of a system can be improved by comparing and correlating the results with different measures of maintainability. We studied the relationship between our UML metrics and the following maintainability measures: Understandability Time (the time spent on understanding the software artifact in order to complete the questionnaire), Modifiability Time(the time spent on identifying places for modification and making those modifications on the software artifact), LOC (absolute net value of the total number of lines added and deleted for components in a class diagram), and nRev (total number of revisions for components in a

class diagram). Our results gave an indication that there is a possibility for a relationship to exist between our metrics and modifiability time. However, the results did not show statistical significance on the effect of the metrics on understandability time. Our results showed that there is a relationship between our metrics and LOC(Lines of Code). We found that the following metrics NAssoc, NClientScriptsComp, NServerScriptsComp, and CoupEntropy explained the effort measured by LOC(Lines of Code). We found that NC, and CoupEntropy metrics explained the effort measured by nRev(Number of Revisions). Our results give a first indication of the usefulness of the UML design metrics, they show that there is a reasonable chance that useful prediction models can be built from early UML design metrics.

Keywords and Phrases : Maintainability, metrics, web applications, empirical studies

Table of Contents

1 Introduction **2**

2 Related Research **5**

2.1 Web Modeling 6

2.2 Web Metrics 7

2.3 Software Maintainability 8

2.4 Software Metrics 10

2.4.1 Validating Software Metrics 11

2.4.2 Theoretical Validation of Metrics 12

2.4.3 Empirical Validation of Metrics 14

2.5 Architecture Views 16

2.5.1 The 4 + 1 View Model 17

2.6 Web Applications 18

2.6.1 Web Application Basics 21

2.6.2 Web Application Modeling using the Unified Modeling Language(UML) 23

3 Maintainability Models **28**

3.0.3 Hierarchical Multidimensional Assessment Model 28

3.0.4 Regression Modeling Techniques 31

3.0.5 WebMO 37

4 Research Methodology **39**

4.1 Problem Description 39

4.1.1 Problem Statement 43
4.1.2 Why are Web Applications Different? 43
4.1.3 Problem Solution 44
4.1.4 Benefits of Maintainability Models 44
4.2 Solution Methodology 45
4.2.1 Planning Phase 45
4.2.2 Definition of UML class diagram metrics for Web Applications 47
4.2.3 Maintainability Measurement 52
4.2.4 Statistical Modeling Phase 54
4.2.5 Post Modeling Phase 55
4.2.6 Model Validation 56

5 Theoretical Validation 61
5.1 Kitchenham's Validation Framework 61
5.2 Briand's Validation Framework 62
5.2.1 Size Metrics 63
5.2.2 Complexity Metrics 65
5.2.3 Coupling Metrics 70
5.2.4 Cohesion Metrics 75

6 Empirical Validation 83
6.1 PetStore Experiment 84
6.2 Telecom Web Applications 91
6.3 Industrial Provisioning Web Application 97

7 Conclusion and Future Work 113
7.1 Conclusion 113
7.1.1 Original Contribution to Knowledge 118
7.1.2 Future Work 119

Appendices 130

A Empirical Surveys **131**

B Empirical Case Studies **141**

B.1 ProvisionApp Interfaces 142

B.2 ProvisionApp Functional Modules 143

B.3 Industrial Provisioning Web Application Data 150

C WapMetrics Tool **158**

C.1 WapMetrics Tool 158

C.1.1 Presentation Component 159

C.1.2 Controller Component 160

C.1.3 Business Component 160

C.2 Case Study 161

C.2.1 Introduction 161

C.2.2 Data Collection 163

C.2.3 Results 163

D WapMetrics Source Code **166**

D.1 Installation Instructions 166

D.2 XMLFileParser 166

D.3 MetricProcessor 180

D.4 MetricComputation 187

List of Figures

2.1 A Structure Model for Measurement [63] 13
2.2 4 + 1 View Model 17
2.3 Web Site Evolution 19
2.4 Service Oriented Web Application [52] 20
2.5 HTTP Protocol and Web Applications 22
2.6 Web Applications Model 26

3.1 WAMM Control Structure Metrics Tree 30
3.2 WAMM Information Structure Metrics Tree 31
3.3 Web Objects Components 38

4.1 Maintenance in Ideal World [29] 41
4.2 Lehman's Laws of Software Evolution [29] 41
4.3 Software Metrics and Maintainability [36] 44
4.4 Web Applications Reference Model 48
4.5 Sample Class Diagram 50

5.1 Sample Class diagram S 66
5.2 Class diagram S with $m1$, $m2$ disjoint modules 67
5.3 Class diagram S showing Module Monotonocity 68
5.4 Class diagram S showing Module Monotonocity for Coupling 72
5.5 Class diagram S showing Modules m1, m2 81
5.6 Class diagram S showing Merging of Modules m1, m2 82

6.1 Comparative Analysis between Cart and Create Customer Understandability Time . . . 89
6.2 Comparative Analysis between Cart and Create Customer Modifiability Time . . . 90
6.3 ScatterPlot LOC Model . . . 109

A.1 welcomeScreen Design . . . 135
A.2 Cart Design . . . 136
A.3 Create Customer Design . . . 137
A.4 Create User Design . . . 138

B.1 ProvisionApp Interfaces . . . 142
B.2 Login Module . . . 143
B.3 Search Module . . . 144
B.4 Current Transactions Module . . . 145
B.5 Service Transaction Module . . . 146
B.6 Device Transaction Module . . . 147
B.7 UserName Module . . . 147
B.8 Retrigger IOTA Module . . . 148
B.9 Error Queue Module . . . 148
B.10 Vision Password Module . . . 149
B.11 Network Provisioning Module . . . 150
B.12 Help Module . . . 151

C.1 WapMetrics Tool Architecture . . . 160
C.2 WapMetrics MainScreen . . . 161
C.3 WapMetrics Results Screen . . . 161
C.4 Claros Home Screen [23] . . . 162
C.5 Claros Contacts Screen [23] . . . 162
C.6 Claros Contacts Class Diagram . . . 165

List of Tables

3.1 WAMM Metrics at System Level 29
3.2 WAMM Metrics at Component Level 29
3.3 Length Metrics 34
3.4 Complexity Metrics 34
3.5 Functionality Metrics 35
3.6 Information Model Metrics 36
3.7 Navigation Model Metrics 36
3.8 Presentation Model Metrics 37

4.1 Web Sites Growth 40
4.2 Number of Web Users in the United States 40
4.3 Web Application Class Diagram Metrics 49
4.4 Measurements of Sample Class Diagram 51

5.5 Web Application Class Diagram Metrics 62
5.6 Metrics Definition based on Kitchenham's model 78
5.7 Metrics Validation based on Kitchenham's model 79
5.8 Metrics Validation based on Kitchenham's model 80

6.9 Web Application Class Diagram Metrics 85
6.10 Results & Analysis 91
6.11 ANOVA Analysis 92
6.12 Characteristics of Web Applications 92
6.13 Web Application Design Metrics 93
6.14 Results 95

6.15 Descriptive Statistics 101
6.16 Univariate Analysis Results 102
6.17 Size Metrics Model 104
6.18 Complexity Metrics Model 105
6.19 Coupling and Cohesion Metrics Model 106
6.20 All Metrics Model 107
6.21 Goodness of Fit: Values of MREs for All Models 108
6.22 Goodness of Fit: Values of MEAN MREs for All Models using Bootstrapping 108

A.1 Data For PetStore Experiment 139
A.1 Data For PetStore Experiment 140

B.1 Characteristics of ProvisionApp 141
B.2 Data for Dependent Variables for the Industrial Provisioning Web Application 152
B.3 Data for Independent Variables for the Industrial Provisioning Web Application 153
B.3 Data for Independent Variables for the Industrial Provisioning Web Application-Continued 154
B.3 Data for Independent Variables for the Industrial Provisioning Web Application-Continued 155
B.3 Data for Independent Variables for the Industrial Provisioning Web Application-Continued 156
B.3 Data for Independent Variables for the Industrial Provisioning Web Application-Continued 157

C.4 Claros Contacts Component Results 164

1

Introduction

Many World Wide Web applications incorporate important business assets and offer a convenient way for businesses to promote their services through the Internet. A large proportion of these web applications have evolved from simple HTML pages to complex applications which have a high maintenance cost. The cost of software maintenance accounts for a large portion of the overall cost of a software system [97]. For example, a problem in the Amazon.com web site in 1998 put the site down for several hours which cost the company an estimated $400,000. This is due to the laws of software evolution [67] and to some special characteristics of web applications. Two software evolution laws which affect the evolution of web applications are: firstly, the law of continuing change where software that is used in real world must change or it will become less useful in the changing world. Secondly, the law of increasing complexity where software becomes more complex as it evolves and more resources are needed to maintain it.

In addition to this, web applications have some characteristics that make their maintenance costly: heterogeneity, speed of evolution, and dynamic code generation. A survey on Web applications conducted by the Cutter Consortium in 2000 revealed that 79% of web projects presented schedule delays. Also, 63% of web projects exceeded their budgets [74]. Therefore, it is important to control the maintenance cost of web applications by using quantitative metrics that can predict web applications' maintainability. Maintainability metrics and models can be useful in many ways: predicting the maintenance cost to provide accurate estimates in a project lifecycle [43], comparing different

design documents to select the documents that have the highest maintainability, identifying risky components to mitigate risks early in the project by reengineering or allocating experienced developers to the risky components [36]. Web applications are different from traditional software systems, models and metrics for traditional systems can not be applied to web applications. The reason for that is that web applications have special features such as hypertext structure, dynamic code generation and heterogenousity that can not be captured by traditional and object-oriented metrics. Another difference is the difference in the unit of measurement of a metric for each application domain. For traditional systems, the unit of measurement of a metric can be a file, procedure or an attribute. For object-oriented systems, the unit of measurement can be a class, interface or attributes. For web applications, the unit of measurement is a web object which can be either an HTML file, JSP, Servlet or client scripts.

The main aims and objectives of this research are to:

- identify and define new UML design metrics based on an extension of Conallen's model [26].
- provide theoretical validation for the metrics based on a validation framework proposed by Kitchenham [63] and another one proposed by Briand [16].
- study the relationship between the UML design metrics and Understandability Time and Modifiability Time (the time spent on understanding and modifying a software artifact).
- study the relationship between the UML design metrics and Lines of Code (absolute net value of the total number of lines added and deleted for components in a class diagram).
- study the relationship between the UML design metrics and nRev (Number of Revisions for classes in a class diagram).
- provide an environment for the maintainability prediction model that includes tools and procedures so that it can be used in an industrial environment.

This thesis is further organized as follows: Chapter 2 discusses related research in the area of web metrics and maintainability models. It gives a general background on software maintainability. It discusses software metrics and architecture views. Then, it gives an overview on the structure of web applications. Finally it discusses how to model web applications using UML. Chapter 3 discusses related research in the area of maintainability models. It describes the following topics, Hierarchical Multidimensional Assessment, Regression Analysis, WebMo. Chapter 4 focuses on the problem description and the proposed solution. It starts by identifying the outstanding problem and the solution. It presents an approach for modeling web applications using an extension of Conallen's model [26]. Chapter 5 provides a classification of the metrics based on the measurement structure proposed by Kitchenham [63] for theoretical metrics validation. In addition, the metrics are validated based on the measurement properties proposed by Briand [16]. Chapter 6 describes the empirical studies used to validate the UML design metrics. This research uses industrial web applications, and the Sun Pets Store application [96]. Chapter 7 provides a conclusion and discusses future work.

2

Related Research

Companies want to know how to assess and predict the quality of their software before it is used; one of the most desirable quality attributes is maintainability [44]. Measures of software maintainability can be taken either late or early in the development process. Late measurements of software maintainability can be used for assessing the software system, planning for future enhancements, and identifying risky software components. On the other hand, early measures of software maintainability can help in allocating project resources efficiently, predicting the effort of maintenance tasks and controlling the maintenance process. Maintainability can be measured by using some of the sub-characteristics of maintainability such as understandability, analyzability, modifiability and testability. Some studies have measured maintainability by measuring both modifiability and understandability [13, 70, 59]. In some studies the maintainability has been quantified in the Maintainability Index (MI) [24, 66, 32]. Other studies have used effort for measuring maintainability [43]. Most of the studies related to maintainability measurements have been carried out using structured and object-oriented systems with little research looking at web applications. In this chapter we discuss related research in the web modeling and web metrics area. In addition, we provide an introduction to the four main areas related to the research described in this thesis: software maintainability, software metrics, architecture views and web applications.

2.1 Web Modeling

There are different models proposed to describe web application structure. WebML [76, 12, 25] is an XML language for modeling web applications. WebML generates four models: Structural Model which models the data content in terms of entities and relationships, Hypertext Model which models the navigation of users, Presentation Model which models the layout and graphic appearance of pages, and Personalization Model which models the user specific contents in the application. WebML is difficult to use with web applications that are not data intensive. Also, WebML is not UML compliant which means it takes time for the users to accept the notation.

Relation Management Methodology (RMM) [56, 55] is a methodology for the design of web applications. RMM is only applicable to web applications with navigational design. Object Oriented Hypermedia Design Method (OOHDM) [91] is a model based approach to the development of web applications. It attempts to use object-oriented methods in the design of web applications [91]. The main limitations of these methods is the concentration on navigational design and the non-compliance with existing approaches in designing web applications such as UML.

ReWeb [87, 89] is a tool to analyze web applications based on the model proposed by Ricca and Tonella [87]. Their model presents a web application using the main web application components: HTML pages, forms, server programs, frames and the relationships between the different components such as link, submit and frame loading relationships. Their model is similar to Conallen's model in representing web applications using UML. The main difference is that Conallen's model captures and defines more relationships between different web application components such as forward, include, redirect and builds relationships. This makes Conallen's model more representable of the actual web application.

We chose Conallen's notation for representing web applications in this research because of its popularity and compliance with UML. Another advantage of using Conallen's model is that Rational Rose Web Modeler [53] and WARE [31] can be used to reverse engineer web applications to the Conallen model. Conallen's model has been referenced

and used widely [88, 49, 30, 31, 26].

2.2 Web Metrics

Most research related to maintainability measurement has been carried out on structured and object-oriented systems. Little work has been done in this regard using web applications.

The Web Application Maintainability Model (WAMM) [32] uses source code metrics and the maintainability was measured using the Maintainability Index. In WAMM new metrics were defined but there is still a need to validate those metrics empirically and theoretically. There is also a need to prove how practical WAMM will be in an industrial environment. WAMM captures many metrics which might make it impractical to implement unless there is a tool which can simply and quickly capture all the metrics and provide a single Maintainability Index. The most common approach used is Regression Analysis. There is research which uses Regression Analysis to define and validate metrics and models for web applications. In [71] design and authoring effort were the dependent variables. The independent variables were based on source code metrics. There is still a need for more empirical studies to validate these newly defined metrics in order to make general conclusions. In [5] design metrics were introduced based on W2000 [5] which is a UML like language. In the study the dependent variables were variations of design effort. The independent variables were measured from the presentation, navigational and information models. Some data for the presentation model was discarded in the study due to lack of participation from all subjects. It is not known how useful this approach would be, since it is not known if the W2000 language is used outside the educational environment and if it will become popular in industrial environments. In [1] Maintenance Time is used as the dependent variable and some metrics based on the Navigational model are used as independent variables. It is not known how practical these approaches are. WebMo [85] introduces the notion of Web Objects as size measures for predicting the effort of developing web applications. Case Based Reasoning [73, 72] is an approach that uses a number of projects' features stored

in a database to predict the effort of the current project.

Genero *et al* [44, 46, 45] use object-oriented UML metrics to measure maintainability for object-oriented systems. Their approach is similar to our approach in using UML class diagram metrics. Our approach is different in the type of metrics used which are based on an extension of Conallen's model. Also in our approach we measure different dependent variables for maintainability. We decided to use UML design metrics since most of the studies use source code metrics for measuring maintainability despite the fact that many studies have shown that early metrics are much more useful [18, 20]. Many other researchers have used design metrics for measuring the quality of their software system [13, 70, 59, 95, 51].

2.3 Software Maintainability

Quality of software is a goal that all stakeholders try to achieve. Software maintainability is an important quality indicator of software in the maintenance phase. In the maintenance phase software professionals spend at least half of their time analyzing software in order to understand it [27]. The cost of software maintenance accounts for a large portion of the overall cost of a software system [97]. Therefore, it is important to have a maintainable software, that is easy to understand, correct and enhance. Software maintenance can be categorized as follows [54]:

- *Perfective Maintenance:* perfective maintenance improves the functionality of the software system by expanding requirements.
- *Adaptive Maintenance:* adaptive maintenance deals with porting a software system to a new hardware or software environment.
- *Corrective Maintenance:* corrective maintenance deals with modifications associated with errors in the software system.
- *Preventive Maintenance:* preventive maintenance deals with software modifications that prevent possible future errors.

One of the main concerns of system stakeholders is to increase the maintainability of the software system. Maintainability can be defined as:
The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment [10].
Maintainability is measured as a function of directly measurable attributes A_1 through A_n as shown in Equation 2.1:

$$M = f(A_1 + A_2 + ... + A_n) \tag{2.1}$$

The measure (M) is a maintainability index which can differ depending on the attributes being used in the measurement. For example, Sneed [93] proposed a software quality assessment environment called SOFTING. SOFTING uses the following design attributes: modularity, portability, integrity, redundancy, complexity, generality, time and span-utilization with associated metrics to calculate the maintainability index.

The following are the different approaches used to quantify maintainability using software metrics [24]:

- *Hierarchical Multidimensional Assessment:* in this technique the attributes are defined in a hierarchy. The top level is divided into three levels control structure, information structure and documentation. Each level is assigned to certain metrics. The total maintainability index is calculated by adding up all the metrics in the hierarchy [82].

- *Aggregate Complexity Measure:* in this technique the maintainability is calculated using a function of entropy [77].

- *Regression Analysis Models:* in this technique a polynomial equation is constructed to measure maintainability using a function of metrics [82].

- *Factor Analysis:* a statistical technique where metrics are grouped into clusters where each cluster has metrics that are highly correlated to each other and lowly correlated to metrics in other clusters. Each group of metrics presents a single underlying factor [77].

- *Principal Components Analysis:* a statistical technique that reduces the collinearity between independent variables. It will reduce the number of independent variables used to construct a maintainability regression model [100].

All these models were tested and validated on Hewlett-Packard systems [24]. They showed reasonably accurate measures of maintainability from simple metrics. The Pearson Product-Moment correlation coefficients (PMCCs) for all models ranged from .60 to .90. The PMCCs showed the correlation between the maintainability of the models and the subjective maintainability rating provided by HP engineers [100]. Regression Analysis and Hierarchical Multidimensional Assessment were the simplest ones to use and to calculate maintainability in the industrial environment at HP [24]. This research is using the regression analysis model technique to build a maintainability model for web applications from UML design metrics.

2.4 Software Metrics

Software metrics are units of measurement that quantitatively characterize some aspects of a software system or process [39]. Software metrics can help in gauging the maintainability of web applications. They can provide support for monitoring, controlling, predicting and evaluating the quality of software systems [40]. Metrics can be categorized into may ways. The main categories are:

- *Product metrics*: measure some aspects of the software structure such as requirements artifacts, design artifacts and source code. Product metrics are important for software engineers to have a better understanding of the software system. [39].

- *Process metrics*: measure the activities of analysis, design and coding during a project lifecycle in order to improve them. Time, effort and cost are the most relevant processes attributes. They can be estimated using Boehm's constructive cost model (COCOMO) [84]. Process metrics are important for team leaders and project managers [39].

- *People metrics*: describe the available resources and their skills. For example, people metrics measure the efficiency of designers, testers and developers during a project lifecycle. Project metrics are important for project managers and system stakeholders. [39].

The research described in this thesis relates to product metrics. Product metrics are defined based on UML class diagrams. Early design metrics are very important since they provide an early indication of any future risks that can happen in the software. Early design metrics can be useful in the following ways. First, predicting the maintenance and cost of maintenance tasks which helps in providing accurate estimates that can help in allocating the right project resources to maintenance tasks [43]. Second, comparing design documents which can help in choosing between different designs based on the maintainability of the design. Third, identifying the risky components of a software since some studies show that most faults occur on only few components of a software system [38, 75]. Fourth, establishing design and programming guidelines for software components. This can be done by establishing values that are acceptable or unacceptable and take actions on the components with unacceptable values. This means providing a threshold of software product metrics to provide early warnings of the system [36]. Fifth, making system level prediction where the maintainability of all components can be predicted by aggregating maintainability of single components. This can be used to predict the effort it will take to develop the whole software system [36].

2.4.1 Validating Software Metrics

Many software metrics have been proposed but not all have been validated. Metrics must be validated for us to have confidence in them. Validation of a metric is accomplished by satisfying the following conditions:

- The software metric is a proper numerical characterization of the property it claims to measure [63].
- The software metric is associated with some external metric such as maintainability or any other quality attribute [7].

The first condition is the theoretical validation, while the second condition is called empirical validations.

2.4.2 Theoretical Validation of Metrics

Software metrics play an important role in controlling software maintenance practices and products. It is important that these measures are valid. Kitchenham proposes a validation framework for software metrics [63] with the goal of showing researchers:

- How to validate a measure.
- How to evaluate the validation of others.
- When to apply a certain measure.

In [63] a structure model of software measurement is proposed, which comprises:

- *Entities, Attributes and Relationships:* entities are objects such as products, and processes. Attributes are the properties of the entities. For example, if we say John is shorter than Mary, the relationship we want to capture and describe formally is the "is shorter than". In Figure 2.1 the relationship between entities and attributes is "many to many". This means that an entity can have many attributes and an attribute can describe many entities. For example, weight attribute applies to many entities such as human beings, furniture and animals.
- *Units and Scale Types and their relationships:* a measurement unit determines how an attribute is measured. An attribute can be measured by more than one measurement such as temperature which can be measured using the Celsius or Fahrenheit measurement unit. A measurement unit can be used to measure more than one attribute. For example, the number of faults can be used to measure the correctness of a program, and it can be used to measure the test case effectiveness. A measurement unit has one to one relationship with the scale type. The scale type can be nominal, ordinal, interval and ratio.

- *Values and their properties:* values are most of the time numerical. They can also be in a set of values such as faulty code, non-faulty code. It is important to know the entity, attribute and the unit to interpret the value correctly. For example, the attribute price for an item can not be interpreted correctly unless we have the unit of currency such as dollars or pounds. The properties for the values has also to be specified. For example, the price of an item can not be negative.

- *Measurement Instrument:* a measurement instrument can optionally be used to measure the unit. An instrument usually measures a single unit but it can measure more than one unit. For example, a thermometer can be used to measure two units of temperature Fahrenheit and Celsius.

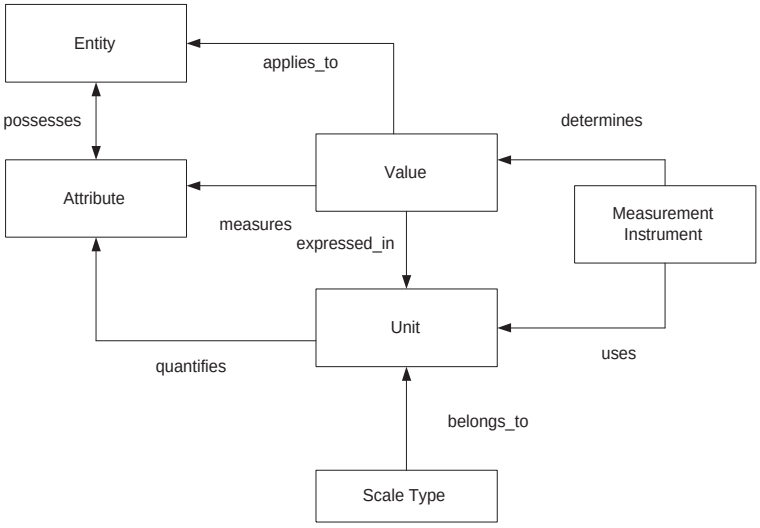


Figure 2.1: A Structure Model for Measurement [63]

Figure 2.1 shows the different elements of the structure model. In addition to kitchenham's [63] framework, this research also uses the general framework proposed by Briand *et al* [16] for theoretical validation for the software metrics. The framework defines properties for several measurement concepts such as size, length, complexity, cohesion, and coupling. The Framework is generic and not specific to any software artifact. In addition to that, it is based on precise mathematical concepts. The objects of study in

the framework are defined as a system which consists of a set of elements and a set of relationships between them.

2.4.3 Empirical Validation of Metrics

Empirical validation of product metrics involves validating that the metric is associated with an external metric and it is an improvement over other product metrics. This is accomplished by carrying out empirical studies with the metrics. An empirical study is a test to compare what is observed to theory. An empirical study can take many forms: eg. surveys, case studies and controlled experiments. An empirical study has the following structure [62]:

- *Experimental Context*
- *Hypotheses*
- *Experimental Design*
- *Threats to Validity*
- *Data Analysis*
- *Results and Conclusions*

Experimental Context

The experimental context defines two elements: Firstly, Background Information which will include information on the problem. The definition of the problem and it's importance. Secondly, Related Research which will discuss about what has been done in the research field and what is still missing and requires further research.

Hypotheses

The hypotheses are important. They state the research question to be answered. The null hypothesis must be stated clearly. It is usually the reverse of what the experimenter believes. The alternative hypothesis is a statement of what the experimenter believes.

Experimental Design

It is important to identify the data that will be used in testing the hypothesis. We also need to identify the subjects and the process by which subjects are selected and assigned to groups. We need to identify two variables dependent and independent variables. The dependent variables are outputs whose values will be predicted based on changes of the independent variables. The independent variables are the predictors or explanatory variables which are used to determine the value of the dependent variable.

Threats to Validity

There are three types of threats that can limit us to draw conclusions from the results: Firstly, Construct Validity which is the degree to which the independent variables and the dependent variables are accurately measured in the study. Secondly, Internal Validity which is the degree to which conclusions can be drawn about effect of independent variables on the dependent variables. Thirdly, External Validity which is the degree to which results can be generalized to other research settings.

Analysis

There are two main approaches of results analysis. Firstly, Classical Analysis which deals with comparing numerical data. The goal of the experiments is to reject or not reject the null hypothesis. Hypothesis testing and power analysis are tools that can be used in this approach. Hypothesis testing determines the confidence level at which the null hypothesis can be rejected. Power analysis determines the magnitude of the effect and the amount of data we have. Secondly, Bayesian Analysis which uses prior information such as data obtained from previous studies or from expert opinions. This approach is not usually used in software engineering studies. Another thing that must be checked is whether to use parametric or nonparametric tests. If the distribution of variables can be identified parametric tests are used, otherwise, nonparametric tests are used [62].

Results and Conclusions

The significance of the results must be reported clearly and they should have enough information for the readers to repeat the experiment. There are some guidelines that must be followed when presenting results.

- All statistical procedures used must be described and citation of references must be provided.
- It is important to report the statistical package used.
- It is good to present the raw data when possible so that it is easier to do a replication experiment.
- It is important to provide descriptive statistics for all dependent and independent variables used in the study.

Conclusions made must follow directly from the results of the study. It is important to define the population to which the results apply. This is important to understand where and how a predictive model can be used [62]. In this research we will be working on design product metrics for web applications. Our work will include validating the proposed product metrics.

2.5 Architecture Views

Different audiences require different views. For example a builder requires the floor plan to view the architecture of a house while a customer needs a view of the actual house architecture. Different views contain different kinds of information and should be described using the most appropriate technique for each view. Each viewpoint has a different collection of metrics that reflect the characteristics of the viewpoint. There are three different architecture views for a software system: Firstly, Conceptual architecture which includes an architecture diagram and CRC cards. The main goal of the view is to identify major components and to allocate responsibilities to components.

Secondly, Logical architecture which identifies the interfaces between components. It is used by designers and developers to build the system. Thirdly, Physical architecture which shows the whole process flow of the system. It is used to assign resources to different processes according to their needs. Importance of architecture views can be found in [65, 19]. The next section will discuss the 4 + 1 view model.

2.5.1 The 4 + 1 View Model

The 4 +1 View model shown in Figure 2.2 defines five different views according to [65].

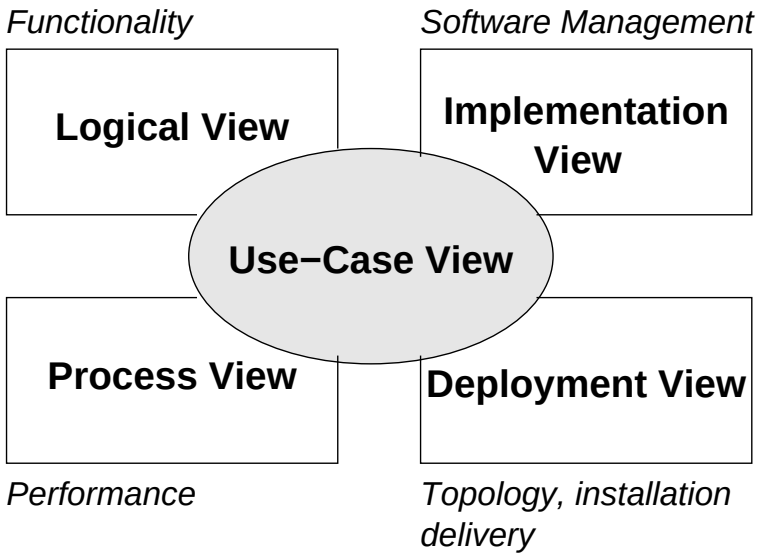


Figure 2.2: 4 + 1 View Model

Each view shows a set of concerns which is of interest to certain stakeholders such as customers, developers and managers. The logical view is created by designers and it supports the functional requirements of the system. Functional requirements are the services that the system is providing to its users. The process view considers non-functional requirements such as performance, fault tolerance and availability of resources. It considers concurrency, system integrity and distribution. In the process view a process is composed of several tasks that form an executing unit. The development view addresses the development environment and how software modules are organized in that environment. The software is grouped into small subsystems that can be worked on by a small

number of developers. The subsystems are organized into a set of layers where each layer provides a well-defined interface to the layer above it. The development view is represented by module and subsystem diagrams that show the import and export relationships of the system. The physical view addresses non-functional requirements such as performance and scalability. It maps software architecture defined in the logical and development view into physical hardware. This mapping needs to have minimal impact on the source code. Scenarios represent use cases. Use cases show that all other views are working together. Use cases are an abstraction of the most important requirement. The scenario view is redundant with other views, that is why we have the +1. In this research we will examine the system from the design point of view which maps to the logical view in the 4 + 1 view.

2.6 Web Applications

The world wide web is the largest distributed application in the world. Sir Tim Berners-Lee originally proposed the world wide web to the European Laboratory for Particle Physics (CERN) in 1990. His goal was to make the web a shared information space through which people and machines could communicate. CERN management approved Sir Berners Lee's proposal in November 1990. He started working on his proposal and developed the first prototype on the NeXT platform. There has been a tremendous growth in web sites since its inception. In 1993 there were around 130 web sites while in early 2001 there were around 27 million web sites [8] as shown in Figure 2.3. This great success of the web resulted in interest from businesses in the web [42], with web applications evolving from simple document sharing applications to complex transaction based applications.

According to Conallen [26] web applications can be categorized in to two types :

1. *Presentation-oriented:* a simple document sharing web site, consisting of hyper-linked text documents that provide information to users.

2. *Service-oriented:* a complicated web application that provides some service to the

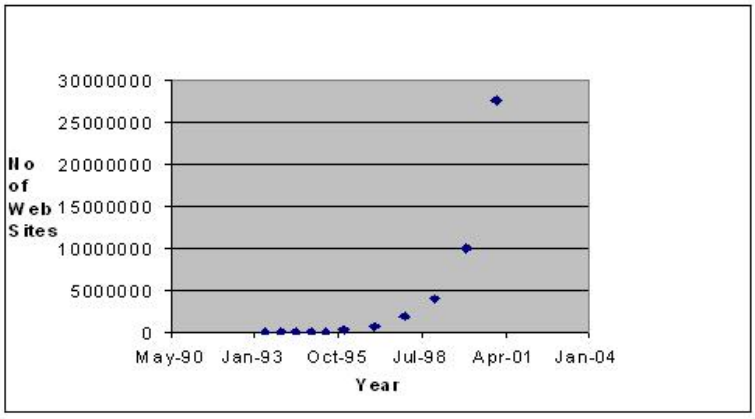


Figure 2.3: Web Site Evolution

user, example e-commerce applications.

In this research we use Conallen's [26] definition of service-oriented applications. Service-oriented web applications are becoming highly complex software systems that run large-scale software applications for e-commerce, information distribution, entertainment, collaborative working, surveys, and numerous other activities. They run on distributed hardware platforms and heterogeneous computer systems. This research will be on service oriented web applications which consist of four tiers [52]. Figure 2.4 shows a service oriented web application based on the J2EE model.

- *Client Tier:* consisting of dynamic Web pages containing various types of markup language (HTML, XML), and a Web browser, which renders the pages received from the server.

- *Web Tier:* consisting of Servlets and JSP pages. Servlets are Java programming language classes that dynamically process requests and construct responses. JSP pages are text-based documents that execute as servlets but allow a more natural approach for creating static content.

- *Business Tier:* containing business code, the logic that solves or meets the needs of a particular business domain such as banking, retail, or finance. It is handled

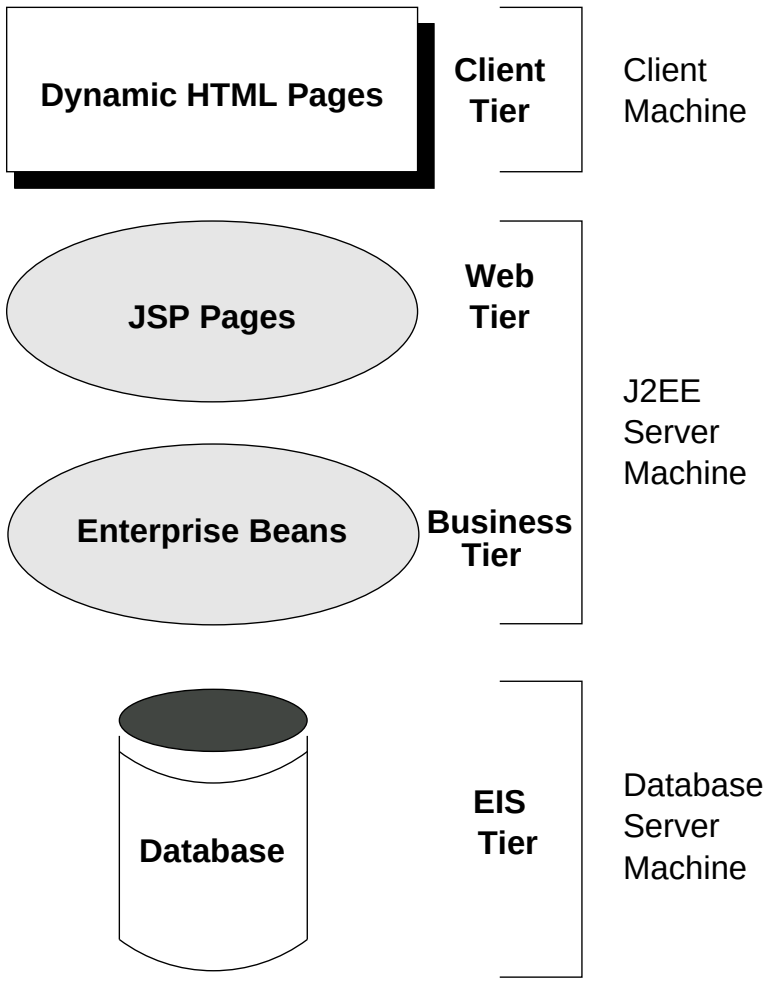


Figure 2.4: Service Oriented Web Application [52]

by enterprise beans running in the business tier. An enterprise bean receives data from client programs, processes it, and sends it to the enterprise information system tier for storage. An enterprise bean also retrieves data from storage, processes it, and sends it back to the client program.

- *Enterprise Information System Tier:* handling enterprise information system software and including enterprise infrastructure systems such as enterprise resource planning (ERP), mainframe transaction processing, database systems, and other legacy information systems.

Service-oriented web applications have several requirements: Firstly, a web site

might have millions of hits per day. If the response time is low customers might leave the web site and go to a different one. Secondly, a web site can grow dramatically due to increase in business so the architecture must be highly scalable. Thirdly users might use sensitive data such as credit cards and other personal and private data. The web site must ensure that they are secure otherwise the web site may lose customers. Fourthly, it is important that service-oriented web sites are easy to change since in the Internet world things change very quickly.

2.6.1 Web Application Basics

This section discusses web applications basics, including HTTP, HTML, Sessions, Dynamic Clients, Scripting, Applets, and ActiveX.

HTTP

The HyperText Transfer Protocol(HTTP) is the protocol used to communicate between the web browser and the web server. The Uniform Resource Locator (URL) is used to access a document on the web server. Documents are located in a directory on the web server that is relative to the web site's root directory. HTTP runs over TCP which is a connection-oriented protocol that implements the OSI model of the networking layer. HTTPS protocol runs HTTP with Secure Socket Layer(SSL) which makes sure that data is protected and encrypted when it is passed between the browser and the web server. In HTTP a client sends a request to the server and the server sends a response back to the client. If the same client sends another request to the server, the server does not know that it is the same client. Therefore, HTTP is called a stateless protocol. This statelessness makes developing web applications more difficult, since in most cases the client makes several calls to the server, and the server needs to keep track of the client that called it. A good example of that is an online shopping web application where the client makes several calls to the server and the server must be aware of that.

Figure 2.5 shows how the HTTP protocol interacts with the web server and web browsers:

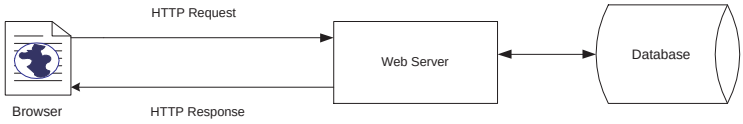


Figure 2.5: HTTP Protocol and Web Applications

HTML

HyperText Markup Language(HTML) is a tag language that is built on the Standard Markup Language(SGML). HTML displays and formats the text files stored on the web server. Style-sheets can be used to get a special formatting of font, color and size. HTML contains an anchor element to connect to other HTML files. Data can be passed to other HTML files using the href attribute in the anchor element. Another important element is the form element which allows the user to enter data and pass it to the web server. The frameset divides the browser into separate layouts where each layout provides a separate display of HTML contents [26].

Sessions

A session associates a user with a single use of the system. An example of that is a shopping-cart session associated with a single user. The session must store some data for the user on the web- server. The data can be stored using cookies or URL parameters [26].

Scripting

Scripting is client side code embedded within HTML. The main reason for using scripting is for client validation. This includes validating the form data before sending it to the web-server. Client validation can save some time and gives a quicker response to errors. An example of a scripting language is JavaScript which is one of the most used scripting languages. JavaScript embeds the code inside a script tag that indicates to the browser that this is scripting code. The problem with scripting is that it is not supported by all browsers. Therefore, it is important to make sure it is supported before using it in

a certain environment [26].

Non-Linear Navigation

Non-Linear Navigation is one of the important characteristic of web applications that is specific to hypertext. The main difference between linear and non-linear navigation is that in linear navigation the client will always be directed to the same page, while in non-linear navigation the client can be directed to different pages depending on some control element. The Non-Linear Navigation characteristic does not apply to traditional software applications and it can make the navigation of web applications hard to understand. In addition to that it makes it difficult to find defects due to the non-linear navigation paths [2].

Dynamic Code Generation

Web applications are getting more complex because of the dynamic code generation aspect [99]. This means that the actual source code is not known until run time. Client code such as HTML, forms, frames, links, and relationships between different web components is generated by server code. This means that it is difficult to determine the execution path for a web page. This makes it more difficult to maintain web applications [87].

2.6.2 Web Application Modeling using the Unified Modeling Language(UML)

Modeling is a technique used to represent complex systems at different levels of abstraction, and helps in managing complexity. UML is an object-oriented language [26] that can be used to model object-oriented systems. It is possible to use UML to model web applications by using extensions supported by UML. Conallen proposed an extension of UML for web applications [26].

The following web application components are used in Conallen's model [26]:

- *Pages:* are an important component of a web application. Pages are requested by browsers and distributed by web servers. Pages contain HTML code, client scripts and server scripts. An example of a page is a JSP or ASP web page. The client scripts in the page are executed by the browser such as JavaScript or VbScript. The client scripts in general are event driven and implement client side validation. The server scripts are executed by the web server and can also do some validation and update the business logic or database of the web application.

- *Forms:* are used to collect user input in a web application. The form can contain several fields such as input text, textarea, selection list, checkboxes, and hidden fields. Each of those fields has a unique name and id. The form has an action element that defines the action page that will receive all the fields once the user hits the submit button.

- *Components:* are software objects that can run on the server side or client side. An example of a server side component is a JavaBean that is accessed by server script code in the web page. Server components can be helpful in encapsulating some of the business logic and making it available for server scripts. An example of client side components is ActiveX and Applets which run on the client side and can provide additional functionality to a web page.

- *Frames:* enhance the user interface by providing multiple pages that are active and open at the same time. The browser page is divided into multiple frames. Each frame has a target browser where components in a frame can interact with components in different frames.

Conallen's Extensions to UML

Figure 2.6 shows the various elements of Conallen web application model. The model uses UML extensions such as stereotyped classes to model the different components of a web application. The following components are defined as stereotyped classes: Web Pages, Forms, Scriplets and Framesets.

A web page is the primary element of a web application. It is modeled with two separate stereotyped classes, the client page and the server page. The client page contains client side scripts and user interface formatting. The server page contains server methods and page scoped variables. It makes sense to model the web page as a client and a server page since a web page has functionality on the client and on the server. It is important to define what attributes and objects each page can have. The client and server page both can have attributes and methods. Conallen's model defines several relationships between the client page and server page. The main relationship between a server page and a client page is a builds relationship since the server page builds the client page. This means that the server page generates the dynamic code for the client page. Every client page can have links to other pages which can be either client or server pages. The link represents the anchor element in a client page. A forward relationship is introduced when the server page delegates responsibility to another server page. An include relationship is introduced when a server page reuses a client or another server page.

Forms are modeled as a stereotyped form class. They are defined to separate the form processing from the client page. The form class can not have methods but it has attributes which are the form fields that are passed to the server page. Each client page can have multiple form classes this is modeled through the aggregation relationship in Conallen's model. The form has a submit relationship to the server page. Each form submits to a different action page.

A scriplet is a cached client page. It has references to components and controls that are re-used by client pages. A client page can have multiple scriplets which is modeled through the aggregation relationship in Conallen's model.

A frameset divides the user interface into multiple views each containing one web page. Frames can contain more than one client page, but they must contain at least one client page. A frameset is similar to a client page but it contains information specific for browsers that do not support frames. Thus a frameset, can have all the relationships that a client page has.

It is important to define the different relationships between components in Conallen's

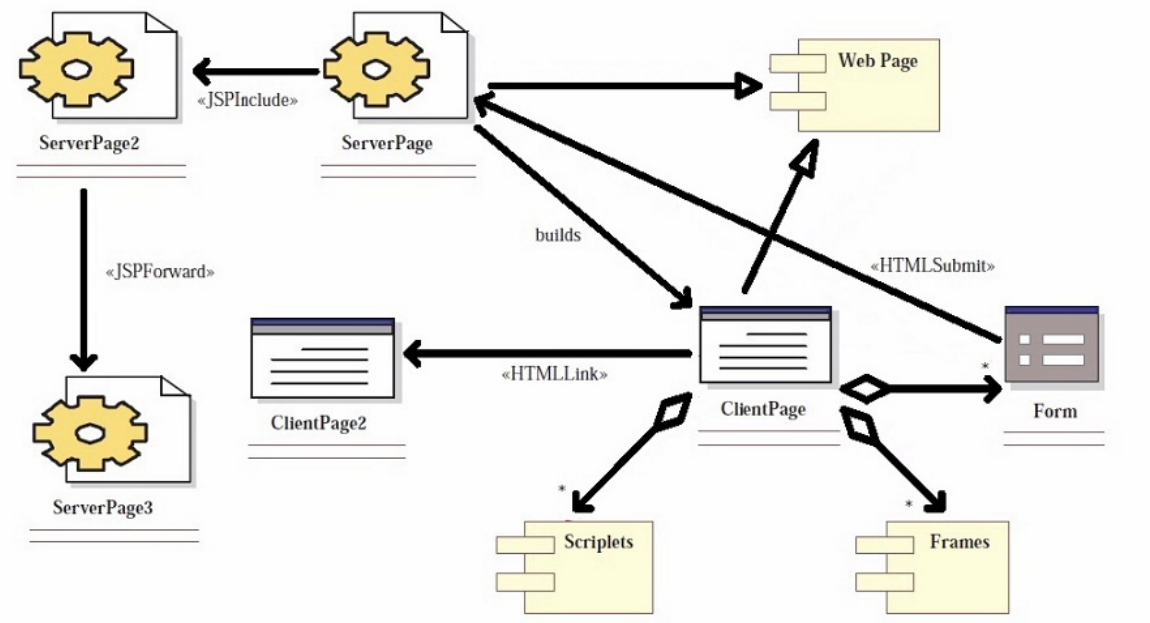


Figure 2.6: Web Applications Model

model. Conallen's model defines the following relations as generic associations between different components: *builds*, *redirects*, *links*, *submit*, *includes*, and *forwards*. The *builds* relationship is a directional relationship from the server page to the client page. It shows the HTML output coming from the server page. The *redirects* relationship is a directional relationship that requests a resource from another resource. The *links* relationship is an association between client pages and server or client pages. It models the anchor element in HTML. The *links* relationship can have parameters which are modeled as attributes in the relationship. The *submit* relationship is a relationship between the form and the server page that processes it. The *include* relationship is a directional association between a server page and another client or server page. The *forward* relationship is a directional relationship between a server page and a client or server page. This presents delegating the server request to another page.

Conallen's model can describe web applications which can be useful during the design phase. Conallen's model has the advantage of being UML compliant and can be extended further to describe web applications in a greater detail. This research will

define a set of metrics based on Conallen's extension of UML. We will study the relationship between these metrics and maintainability.

3
Maintainability Models

In this chapter we will discuss in detail some of the methodologies that have been used for defining metrics and maintainability models for web applications.

3.0.3 Hierarchical Multidimensional Assessment Model

The HPMAS model has been adapted to the context of web applications in the Web Application Maintainability Model (WAMM) [32]. The Web Application Maintainability Model(WAMM) is based on the Oman and Hagemeister model proposed for traditional software systems. WAMM proposes new metrics for web applications in order to predict the maintainability of web applications. WAMM focuses on the source code branch of the HPMAS model. It defines metrics at the component and system level. These metrics are shown in Table 3.1 and Table 3.2:

The Maintainability Index (MI) of the web application is calculated as a function of the metrics defined in the WAMM model. This is shown in Equation 3.1:

$$MI = F(y_i A_i) \tag{3.1}$$

In this equation we have the following variables:

- A_i: The A_i variables is the i-th value of the metric attribute.
- y_i: The y_1 variable is the i-th weight of the metric.

Metric Name	Description
TotalWebPage#	Total number of WA pages
TotalLOC#	Total number of WA LOCs
ServerScript#	Total number of server scripts
ClientScript#	Total number of client scripts
WebObject#	Total number of web objects
InterfaceObject#	Total number of interface objects
TotalData#	Total number of different data identifiers
TotalConnectivity#	Total number of relationships among web pages
TotalLanguages#	Total number of programming languages

Table 3.1: WAMM Metrics at System Level

Metric Name	Description
WebPageTag#	Number of tags in the page
WebPageScript#	Number of scripts in the page
PageWebObject#	Number of web objects in the page
WebPageRelationships#	Number of relationships the page has with other pages
WebPageData#	Number of different data identifiers in the page
WebPageDataCoupling#	Number of data exchanged with other web pages
InnerComponents#	Number of inner components in the page
WebPageControlStructure#	Number of control flow structures
ScriptSize#	Number of source LOCs forming the script

Table 3.2: WAMM Metrics at Component Level

WAMM defines metrics for the source subtree of the Oman model. It specifically defines metrics for the control structure and information structure branches. This is shown in Figure 3.1 and Figure 3.2:

WAMM was applied on two of case studies that incorporate different technologies such as ASP, JavaScript and HTML. These case studies included a freeware discussion forum, customizable portal and a prototype of an e-commerce application. The metrics of these applications were computed using the WARE tool [31] which analyzes the source code and computes the WAMM metrics directly from the source code. The results of these experiments provide a first validation of the maintainability model. There

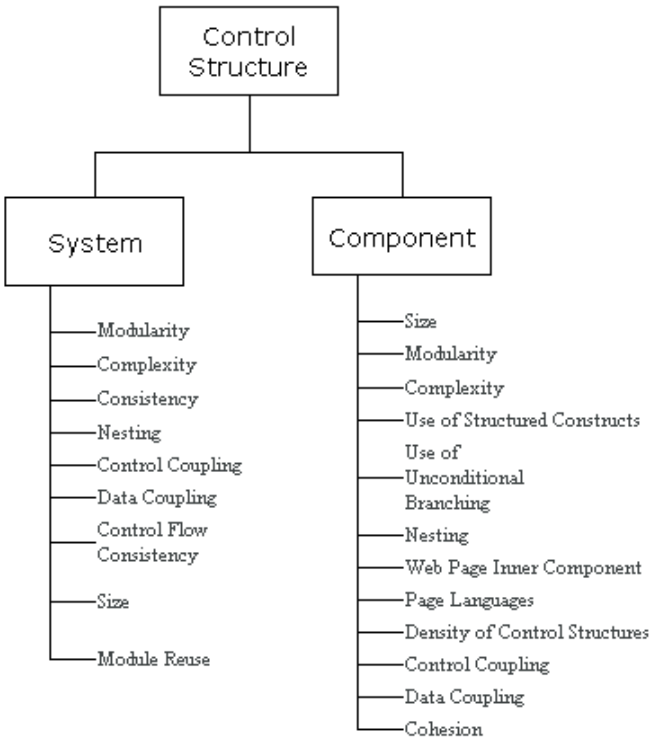


Figure 3.1: WAMM Control Structure Metrics Tree

is still a lot of empirical research needed to validate the model. There is a need to prove empirically that the new defined metrics are effective in calculating the maintainability of web applications. Also the weight of the coefficients needs to be defined through the analysis of historical data and more experiments. WAMM concentrates on source code metrics. One drawback of this approach, is that we have to capture many metrics. This becomes impractical especially when we have metrics which are collinear which means they capture the same underlying attribute.

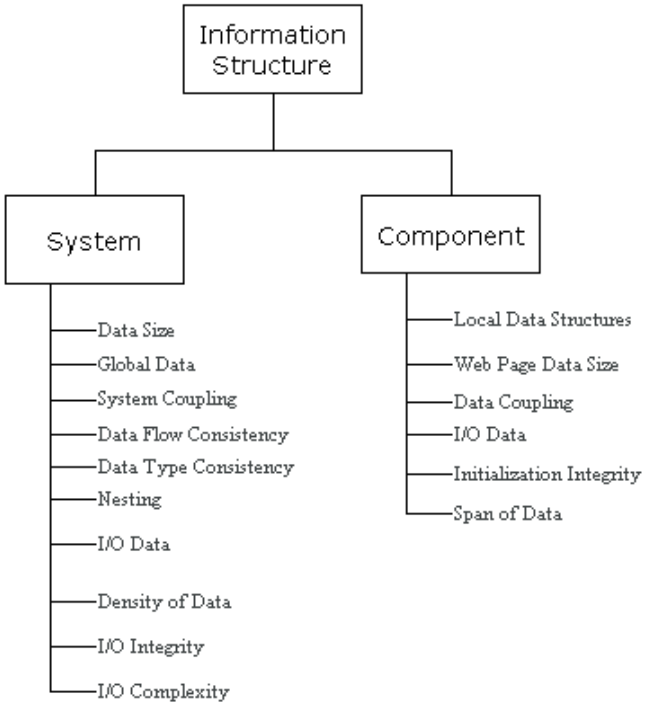


Figure 3.2: WAMM Information Structure Metrics Tree

3.0.4 Regression Modeling Techniques

Regression analysis is applied in many areas of software engineering research. One of those areas is defining measures for software maintainability. Regression analysis defines the dependent variable as a quantitative measure of some condition or behavior. The main goal of regression analysis is to determine the values of parameters for a function that cause the function to best fit the data provided. The linear regression model can be built using several regression techniques such as Relative Least Square (RLS) and Least Square (LS). These regression techniques produce the coefficients for the independent variables. The independent variables have to be selected in a way that reduces the noise in the model. There are a couple of techniques used in selecting the

independent variables such as stepwise regression, backward regression and forward regression [58]. We need to consider the following issues when we apply regression analysis [79]:

- Do we have the right and most important independent variables?
- Do we have the correct specification of the model?
- What is the predictive power of the model?

Choosing the Right Independent Variables

It is very important to choose the right and most important independent variables. In the maintainability model the independent variables are the variables that affect maintainability of the software system. These variables are usually a set of complexity and size metrics which can be obtained from requirements, design or source artifacts. After selecting a set of independent variables a a correlation test can be run to determine the variables that are correlated. We can measure the three types of correlation tests Pearson correlation, Kendall rank correlation and Spearman correlation. Pearson correlation is a parametric test and assumes a normal distribution between the variables. Kendall rank correlation and Spearman correlation are non-parametric tests and do not assume any assumptions related to the distributions. The correlated variables can be removed since they capture the same information. As a result, we will have a reduced set of variables, this makes it easier for practitioners to use.

Correct Model Specification

In regression analysis it is common practice to assume a linear relationship between the independent and dependent variables. It is important to verify the assumptions made regarding the linearity of the model. If the data does not fit a straight line the relation might not be linear. In that case we need to transform the data and use a different equation for fitting the data to the function. There might be some data points that are far from the fitted line, these data points are called outliers. It is important to look for

these outliers and analyze if they are due to an error in data recording. Sometimes these data points can be removed before regression analysis if there is strong evidence that this data is unreliable.

Predictive Power of the Model

It is important to assess the predictive power of the model. This can be done by measuring the goodness-of-fit for the model. The goodness-of-fit measures how much the independent variables explains the dependent variable. The higher the goodness-of-fit the better the predictive power of the model. Another technique to measure the predictive power of the model is the Mean Magnitude of Relative Error (MMRE). MMRE is calculated by taking the mean of the difference between the actual value of the dependent variable and its predicted value divided by the actual value. The lower the MMRE value the better the predictive power of the model [61].

In the next section we will discuss how regression analysis was used to build maintainability models and define metrics for web applications.

Regression Analysis Models for Web Applications

There are few examples of maintainability prediction models for web applications in the literature. Many of these prediction models require more empirical research in order to be accepted and validated. In [71] linear and stepwise regression analysis is used to build an effort prediction model for web applications. Size is used as an independent variable and design and authoring effort is used as the dependent variable. The size is measured in terms of length, functionality and complexity. The case study used a medium-size web application with forty-three computer science students, the data was collected through two questionnaires. The first questionnaire was used to get personal data from the subjects such as their experience in designing and authoring web applications. The second experiment was used to measure the size metrics, confounding factors, and design and authoring effort. Table 3.3 shows the length metrics and Table 3.4 shows the complexity metrics while Table 3.5 shows the functionality metrics.

The functionality metric measures the number of *reads*, *writes*, *entries* and *exits* to

Metric Name	Description
Page Count	Number of html files used in the application
Media Count	Number of non-reused media files used in the application
Program Count	Number of non-reused cgi scripts, Javascript files, Java applets used in the application
Total Page Allocation	Total space allocated for all html pages used in the application
Total Media Allocation	Total space allocated for all media files used in the application
Total Embedded Code length	Total number of lines of code used in the application
Reused Media Count	Number of reused/modified media files
Reused Program Count	Number of reused/modified programs
Total Reused Media Allocation	Total space allocated for all the reused media files in the application
Total Reused Code Length	Total number of lines of code for all the programs reused by an application.

Table 3.3: Length Metrics

Metric Name	Description
Connectivity	Total number of links
Connectivity Density	Connectivity/Page Count
Total Page Complexity	Summation of the number of different media types in a page/Page Count
Cyclomatic Complexity	(Connectivity - Page Count) + 2

Table 3.4: Complexity Metrics

the software system. An example of an *entry* is a request to an I/O device such as a client request to the web server. An example of an exit is a web server response to the client. A read and write is any request that writes or reads from the storage of the application such as a read or write to a database.

In the study the following confounding variables were controlled:

- *Tool Type*: This measures the type of tool used in designing and authoring the web application.
- *Experience*: This is a measure of the design and authoring experience of the subjects.
- *Structure*: This is a measure of the main structure of the web application.

Metric Name	Description
Size	Summation of size entries + Summation of size exits + Summation of size reads + Summation of size writes

Table 3.5: Functionality Metrics

In the study both linear and stepwise regression analysis was used and, the following results were reported:

- *Length Metrics*: both models included Page Count and Total Reused Code Length. The other predictors did not have a statistical significant correlation, so they were removed from the model.
- *Complexity Metrics*: both models included Connectivity since it had the best statistical correlation with effort.
- *Prediction Capability*: the models prediction capability was calculated using the adjusted-R^2. No model presented accurate prediction of effort
- *Results:* both prediction techniques gave similar results in terms of their predictive capabilities. They were compared using the boxplots of residuals which is the difference between the actual and predicted values of the effort.

As a conclusion, replication studies need to be carried out to verify the results. The metrics used in the experiment need to be reviewed to see if there are other metrics that are better predictors of the dependent variable effort.

In another study [5] the effort of designing a web application is the dependent variable. The independent variables are collected from design artifacts. Size, complexity, reuse and decomposition are the software attributes that were used as a basis to define the independent variables for the model. The W2000 modeling language is used in creating UML like diagrams. The W2000 is developed at Politecnico di Milano. The following W2000 models were used in the study:

- *Information Model*: identifies all the data that the web application is using.

- *Presentation Model*: identifies all pages with links and relationships between them.

- *Navigation Model*: structures elements in the Information Model in a new form. It uses nodes to define the main elements and clusters the group nodes together.

Each model in W2000 had its own metrics. Table 3.6, Table 3.7, and Table 3.8 show the metrics that were used in each model:

Metric Name	**Description**
Entities	Number of entities in the model
Components	Number of components in the model
Infoslots	Number of slots in the model
SlotsSACenter	Average number of slots per semantic association center
SlotsCollCenter	Average number of slots per collection association center
ComponentEntity	Average number of components per entity
SlotsComponent	Average number of slots per component
SAssociations	Number of semantic association in the model
SACenters	Number of semantic association centers in the model
Segments	Number of segments in the model

Table 3.6: Information Model Metrics

Metric Name	**Description**
Nodes	Number of nodes in the model
NavSlots	Number of slots in the model
NodesCluster	Average number of slots per cluster
SlotsNode	Average number of slots per node
NavLinks	Number of links in the model
Clusters	Number of clusters in the model

Table 3.7: Navigation Model Metrics

The following were the dependent variables that were measured in the experiments

- *Information Effort*: the effort needed to design the Information Model.

Metric Name	Description
Pages	Number of pages in the model
PUnits	Number of publishing units in the model
PrLnks	Number of links in the models
Sections	Number of sections in the model

Table 3.8: Presentation Model Metrics

- *Presentation Effort*: the effort needed to design the Presentation Model.
- *Navigation Effort*: the effort needed to design the Navigation Model.
- *Total Effort*: the effort needed to design all the models.

This study is an exploratory study that can be used as a basis for additional research. A number of predictors have been identified in the study that have a significant statistical impact on effort. A drawback of this study is using the W2000 modeling language which is not used in the industry or known in educational environments. Therefore, there would be a need for subjects to understand this language to run a replication study.

3.0.5 WebMO

WebMo is a cost estimation model introduced by Reifer. It is an adaptation of the COCOMO II model [85]. WebMo introduces a new size predictor called Web Objects. Web Objects are needed because current size metrics can not capture all attributes of web applications [85]. Web Objects provide an indication of the relative size of a web application. Web Objects add four new components to the function point approach: multimedia files, web building blocks, scripts and links. These new web components are shown in Figure 3.3.

A brief description of these components is given below:

- *Multimedia Files*: this component calculates the effort required to insert audio, video and images into applications.

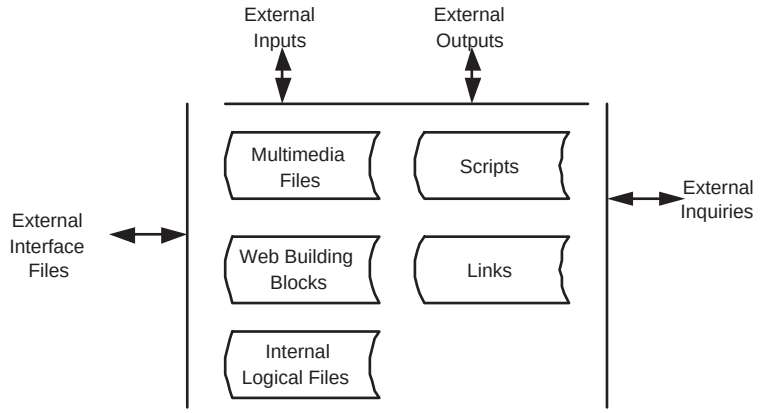


Figure 3.3: Web Objects Components

- *Links*: this component takes into account the effort to link applications and bind them to the database.
- *Scripts*: this component takes into account the effort required to link HTML data with application files.
- *Web building Blocks*: this component takes into account the effort required to develop web blocks such as JSPs and Servlets.

WebMo uses data from 64 web applications in five applications domain to develop accurate estimates for web applications. It forces the collection of certain cost factors such as product reliability, platform difficulty, personnel experience, facilities, and teamwork. Some of these cost factors might not apply in all application domains. As far as we know the only empirical industrial study on web objects is the one done in [90] which compares the performance of objects with function points. The results showed that web objects are better effort predictors. There still is a need to conduct more empirical studies on WebMo to confirm and validate the results.

This chapter discussed some of the models for defining metrics for web applications were described in detail. The next chapter discuses the research methodology applied in this thesis.

4

Research Methodology

This chapter provides a description of the problem, and the proposed research methodology.

4.1 Problem Description

Maintenance presents a major cost factor in the lifecycle of a web application. There are many reasons for the high maintenance cost of web applications such as fast internet evolution, Lehman's laws of software evolution and the particular characteristics of web applications. These reasons are discussed below.

Internet Evolution

The Internet has evolved tremendously in terms of number of web sites and number of usage during the last decade. Table 4.1 shows the growth of the Internet in terms of number of web sites. In 1993 there were 130 web sites, in 2001 the number reached over 27 million web sites [8]. Table 4.2 shows internet growth in terms of number of users in the United States. In 1996 there were 31.9 million users rising to 116.5 million users in 2000 [86]. Amazon.com has a leading e-commerce web application. They started with 0 customers in 1995, by 2003 they had around 20 million customers and the largest online store in 220 countries [8].

The rapid growth of the internet, the enormous evolutionary change with very short

project release cycles and high competition has resulted in many unreliable web applications. According to one study [37], almost 70% of leading e-commerce and government sites exhibit some time of failure when used by a first time user.

Date	Number of Web Sites
June-1993	130
December-1993	623
June-1994	2,738
December-1994	10,022
June-1995	23,500
January-1996	100,000
June-1996	252,000
January-1997	646,162
January-1998	1,834,710
January-1999	4,062,280
January-2000	9,950,491
January-2001	27,585,719

Table 4.1: Web Sites Growth

Year	Internet Usage in the USA
1996	31.9 million
2000	116.5 million

Table 4.2: Number of Web Users in the United States

Lehman's Laws of Software Evolution

In the ideal world of software development growth would be similar to that in Figure 4.1. We would have a perfect software engineer, perfect requirements and a perfect design. Processing all these three input together should give us a perfect system, a happy customer and little work for the maintainer. In real life this is not true even if we have all these three inputs together [29]. The reason for that can be explained by Lehman's laws

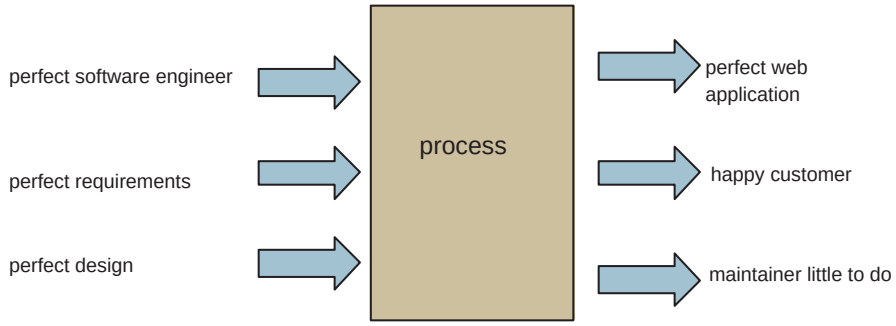


Figure 4.1: Maintenance in Ideal World [29]

of software evolution [67]. There are two laws shown in Figure 4.2 and described as follows:

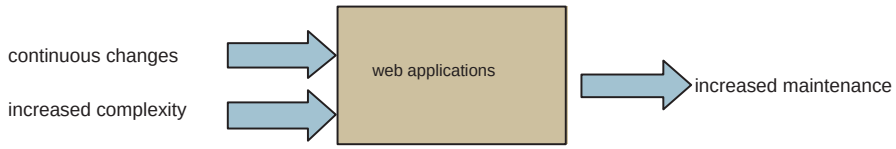


Figure 4.2: Lehman's Laws of Software Evolution [29]

1. *The law of continuing change*: a program used in real world must change or eventually it will become less useful in the changing world.

2. *The law of increasing complexity*: as a program evolves it becomes more complex and extra resources are needed to preserve and simplify its structure.

Web Application characteristics

Web applications have several characteristics that make them difficult to maintain. These characteristics can be summarized as follows:

- *Heterogenousity*: Web applications are usually composed of four tiers each tier uses different programming languages, protocols and technologies. The client tier consists of dynamic web pages containing HyperText Markup Language (HTML),

Extensible Markup Language (XML), JavaScript. The web tier consists of web components such as servlets and Java Server Pages (JSPs) which translate requests between the client and the business tier. The business tier implements the bulk of the application logic. The data layer tier handles enterprise information stored in database systems [6]. In my opinion the heterogenousity characteristic applies more to web applications because they are composed of multiple tiers while traditional software applications are in general composed of one or two tiers.

- *Technology advancement*: There is an increasing demand from customers to incorporate new technologies in their web applications. In the early 90s Tim Berners-Lee created the first web browser and web server [42]. Web pages started as simple static HTML which were used for document sharing. Today web applications are becoming complex software structures which contain many relations between their components [6]. They usually require transaction processing, high security and high performance due to the criticality of the applications and the high number of users. In my opinion the technology advancement characteristic also applies to traditional software applications.

- *Speed of evolution*: Web application have a fast maintenance rate [60] and usually have a short release cycle [28] due to continuous customer demands [33]. All this causes a lack of appropriate and up to date documentation. This characteristic also applies to traditional software systems but in general it is more relevant in web applications [78].

- *Dynamic code generation*: In traditional software applications both the client and server code are static while in web applications the client code and the relationships between different web components may be generated by server code [78]. This means that the actual source code is not known until run time. This makes it more difficult to maintain web applications.

- *Duplicated code*: Web applications tend to have duplicated code since developers tend to cut and paste code from different parts of the system [28]. If an error

happens in one part it will also happen in other parts of the code. Also if there is a need for an enhancement many parts of the web application must be touched. In my opinion duplicated code is found more in web applications but some studies showed that it is also available in traditional software applications [34].

- *Tangled and Scattered Code*: Web applications have functionalities that spread over presentation, control, business and database tiers which leads to tangled and scattered code [64]. This code is difficult to maintain since a simple change involves all parts of the code. In my opinion the tangled and scattered code is found in both web applications and traditional software applications but it is more relevant in web applications.

4.1.1 Problem Statement

Web applications are one of the fastest growing classes of software systems. They have diffused in many and different business domains such as scientific activities, product sale and distribution and medical activities [37]. These web applications have evolved into complex applications that have high maintenance cost. This high maintenance cost of web applications is due to the inherent characteristics of web applications, to the fast internet evolution and to the pressing market which imposes short development cycles and frequent modifications. In order to control the maintenance cost, quantitative metrics and models for predicting web applications' maintainability must be used. The maintainability metrics and models can be useful for predicting the maintenance cost, predicting risky components and, choosing between different software artifacts.

4.1.2 Why are Web Applications Different?

Web applications are different from traditional software systems. Therefore models and metrics for traditional systems can not be applied to web applications. This is because web applications have special features such as hypertext structure, dynamic code generation and heterogenousity. Web applications' features can not be specified by regular metrics that are applied to traditional or object-oriented systems. Another difference is

the difference in the unit of measurement of a metric for each application domain. For traditional systems, the unit of measurement of a metric can be a file, procedure or an attribute. For object-oriented systems, the unit of measurement can be a class, interface or attributes. For web applications, the unit of measurement is a web object which can be either an HTML file, JSP, Servlet or client script.

4.1.3 Problem Solution

Figure 4.3 shows that maintainability and maintenance cost is affected by software complexity. In our research, we will investigate the use of software design metrics that predict the maintainability of web applications. We will present a methodology for assessing, evaluating and selecting software design metrics for predicting web applications' maintainability. In addition, we will propose a maintainability prediction model for web applications.

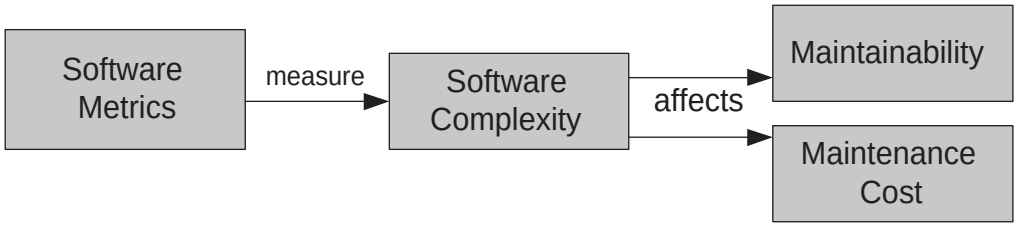


Figure 4.3: Software Metrics and Maintainability [36]

4.1.4 Benefits of Maintainability Models

Maintainability models can be useful for:

- *Predicting Maintenance Cost*: predicting the maintenance and related cost can provide accurate estimates that can help in allocating the right project resources to maintenance tasks [43].

- *Comparing Design Documents*: predicting the maintainability of design documents can help in choosing between different designs based on the maintainability

of the design.

- *Identifying Risky Components*: some studies show that most faults occur on only few components of a software system [81]. Identifying those components early can help in mitigating risk since more resources can be allocated to those components during development or testing. Identifying those components is done through a maintainability model. Some of these models have been used to control the quality of switching software at Alcatel [35].

- *Design and Programming Guidelines*: maintainability model can help in establishing design and programming guidelines for software components. This can be done by establishing values that are acceptable or unacceptable and taking action on the components with unacceptable values. This means providing a threshold of software product metrics to provide early warnings of the software system [36].

- *Making System Level Prediction*: maintainability of all components can be predicted by aggregating maintainability of single components. This can be used to predict the effort it will take to develop the whole software system [36].

4.2 Solution Methodology

This approach uses statistical regression analysis to build the maintainability model for web applications. The proposed approach can be divided into three phases [36], described in the following three sections:

4.2.1 Planning Phase

In the planning phase we will decide on the metrics selection, dependent variable, and data analysis technique. All of these are described below:

Metrics Selection

Measures of software maintainability can be taken either late or early in the development process. Late measurements of software maintainability can be used for assessing the software system, and planning for future enhancements, early measures of software maintainability can help in allocating project resources efficiently, predicting the effort of maintenance tasks and controlling the maintenance process.

There are several design quality attributes that have an effect on the maintainability of software artifacts. As mentioned in the related work section, we decided to use UML design metrics since most studies use source code metrics for measuring maintainability despite the fact that many studies have shown that early metrics are much more useful [18, 20]. Also many other researchers have used design metrics for measuring the quality of their software system [13, 70, 59, 95, 51]. Early measures are very important since they provide an early indication of any future risks that can happen in the software. The following are the design attributes that we will use in our model:

- *Size*: in our model the size of a UML class diagram is measured by counting the number of components in the diagram. The lower the size of a component the higher the maintainability.

- *Complexity*: measured by measuring the number of branches in a component. McCabe Cyclomatic Complexity [43] is a common measure for complexity. We use the number of associations and relations in UML class diagrams to measure complexity in our model.

- *Coupling*: the degree of interaction between two components [13, 14]. It is important to have low coupling in order to have high maintainability. Low coupling can be achieved by reducing the number of messages that can be sent and received by individual components [13]. There are several standard coupling measures defined in the literature. Coupling Between Object Classes (CBO) [22] is measured by the number of other classes it is coupled to. A class is considered to be coupled to another one if it uses methods or attributes of the other class. Response

Set For a Class (RFC) [22] is measured by the number of methods that are called when a message is received by the class. Message Passing Coupling (MPC) [68] is measured by the number of method calls in a class. Data Abstraction Coupling (DAC) [68] is measured by the number of attributes in a class that have as their type another class. In this research coupling is measured using the relationships and components in the UML class diagram.

- *Cohesion*: the degree to which the methods and attributes of a class belong together. [3, 15]. There are a number or metrics proposed in the literature for measuring cohesion. Lack Of Cohesion in Methods (LCOM) [22] is measured by the number of pairs of methods that do not use attributes in common. Tight Class Cohesion (TCC) [11] is measured by the percentage of methods that are connected. Connected methods use common attributes directly or indirectly. Loose Class Cohesion (LCC) [11] is measured by the percentage of pairs of public methods of the class which are directly or indirectly connected. In this research cohesion is measured using the relationships and components in the UML class diagram.

- *Reusability*: taking components of one product in order to facilitate the development of a different product with different functionality [8]. The research measures the reusability by looking at the percentage of web components that are reused in the whole application.

We have identified several metrics for measuring these quality attributes. These metrics are defined in the next section.

4.2.2 Definition of UML class diagram metrics for Web Applications

We have used the model shown in Figure 4.4 to model our web applications and to define our metrics, the main difference between the model in Figure 4.4 and Conallen's model is an additional element called Interface Objects. The Interface Object element is an extension to the current Conallen model, it is a class that has an association relationship

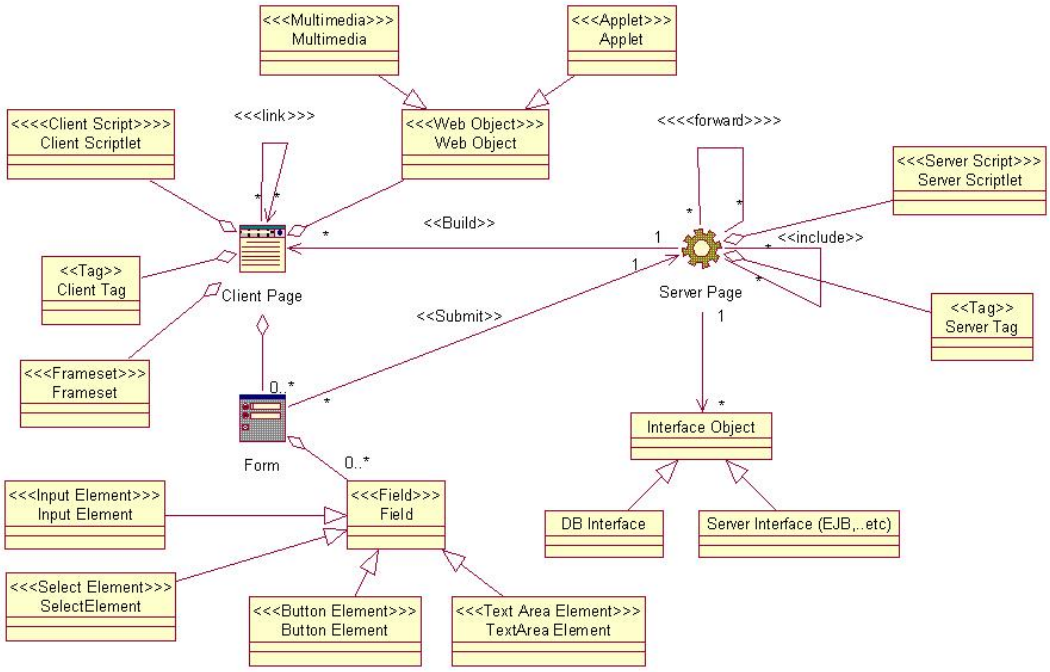


Figure 4.4: Web Applications Reference Model

to the server page in Conallen's model. A server page can have many Interface objects that is why we have the one to many relationship in Figure 4.4 between the server page and the Interface Object. The Interface Object is a class that can have attributes and methods. When modeling a web application it is important to show the Interface Object and all the classes that are associated to it, since these classes give a better picture on the complexity of the class diagram. For example if we have a server page that calls one Interface Object we will show an association relationship between the Interface Object and the server page. If the Interface Object uses other classes to call some server side methods we will show these classes in the class diagram and draw an association relationship between the Interface Object and the other server side classes. The rest of the elements in Figure 4.4 are similar to Conallen's model described in detail in Section 2.6.2.

As mentioned in the related work section, we choose Conallen's notation for representing web applications because of its popularity and compliance with UML. An-

Metric Type	Metric Name	Description
Size	NServerP	Total number of server pages
Size	NClientP	Total number of client pages
Size	NWebP=(NServerP + NClientP)	Total number of web pages
Size	NFormP	Total number of form pages
Size	NFormE	Total number of form elements
Size	NClientScriptsComp	Total number of client scripts components
Size	NServerScriptsComp	Total number of server scripts components
Size	NC	Total number of classes
Size	NA	Total number of attributes
Size	NM	Total number of methods
Structural Complexity	NAssoc	Total number of associations
Structural Complexity	NAgg	Total number of aggregation relationships
Structural Complexity	NLinkR	Total number of link relationships
Structural Complexity	NSubmitR	Total number of Submit relationships times NFormE
Structural Complexity	NbuildsR	Total number of builds relationships times (NServerScriptsComp + NClientScriptsComp)
Structural Complexity	NForwardR	Total number of forward relationships
Structural Complexity	NIncludeR	Total number of include relationships
Coupling	WebControlCoupling = (NLinkR + NSubmitR + NbuildsR + NForwardR + NIncludeR)/ NWebP)	Number of relationships over number of web pages
Coupling	WebDataCoupling = (NFormE/NServerP)	Number of data exchanged over number of server pages
Coupling	EntropyCoupling = $1/n \times (-\log 1/(1+m))$	where n is total number of elements and m is total number of relationships
Cohesion	EntropyCohesion	total entropy coupling of the application / entropy coupling of one class diagram
Resusability	WebReusability = (NIncludeR/NWebP)	Number of include relationships over number of web pages

Table 4.3: Web Application Class Diagram Metrics

other advantage of using Conallen's model is that Rational Rose Web Modeler [53] and WARE [31] can be used to reverse engineer web applications to the Conallen model. Conallen's model has been referenced and used widely [88, 49, 30, 31, 26].

This research defines metrics based on the web application reference model shown in Figure 4.4. The metrics are based on Web Application Extension (WAE) for UML and measure attributes of class diagrams. Table 4.2.1 provides a description of the metrics. The following metrics (NServerP, NClientP, NWebP, NFormP, NFormE, NLinkR, NSubmitR, NbuildsR, NForwardR, NIncludeR, NClientScriptsComp, NServerScriptsComp,

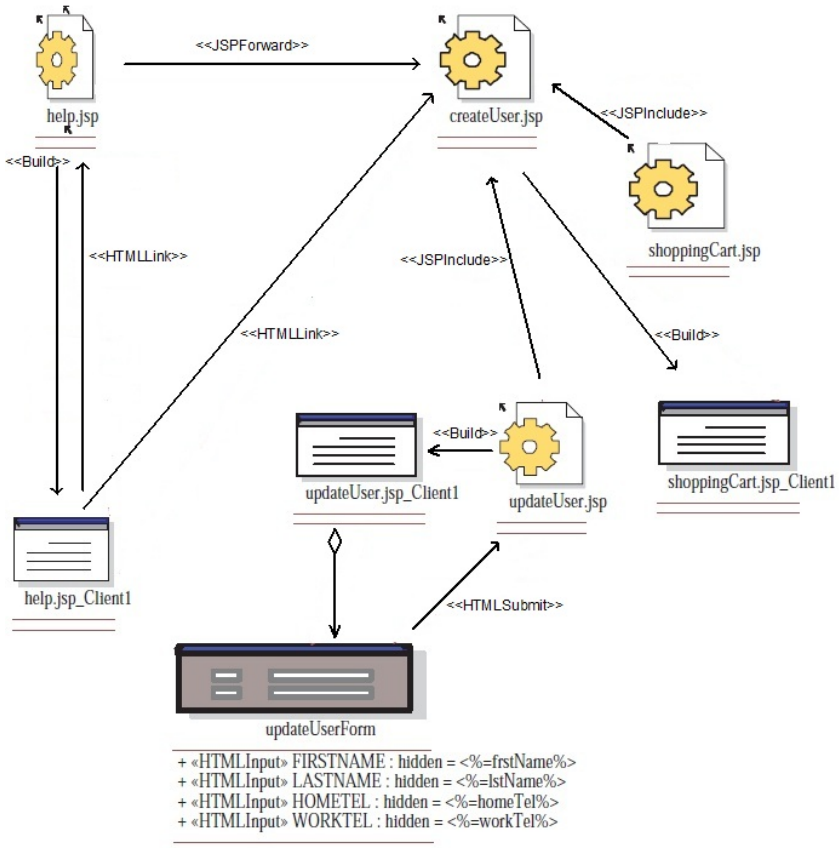


Figure 4.5: Sample Class Diagram

WebControlCoupling, WebDataCoupling, WebReusability) were defined in the author's previous study [48]. The following (NC, NA, NM, NAssoc, NAgg) metrics were defined in the study carried by Genero [44] on class diagram metrics for object-oriented applications. The metrics use the different components of the web application reference model as units of measurement. Figure 4.5 shows a sample class diagram and Table 4.4 shows the results of calculating some of the metrics from the sample class diagram.

Dependent Variable

The next step in the planning phase is to identify the type of the dependent variable and how to measure it. Dependent variables can be binary or continuous. An example of a continuous variable is a count: a non-negative integer such as the number of hours, or the

Metric Type	Metric Name	Measurement
Size	(NServerP)	4
Size	(NClientP)	3
Size	(NWebP)	7
Size	(NFormP)	1
Size	(NFormE)	4
Size	(NClientC)	0
Structural Complexity	(NLinkR)	2
Structural Complexity	(NSubmitR)	4
Structural Complexity	(NBuildsR)	3
Structural Complexity	(NForwardR)	1
Structural Complexity	(NIncludeR)	2
Control Coupling	(WebControlCoupling)	1.7
Data Coupling	(WebDataCoupling)	1
Reusability	(WebReusability)	0.28

Table 4.4: Measurements of Sample Class Diagram

number of faults. Binary variables can have only two values, for example a component can be faulty or non-faulty. Many dependent variables are continuous but sometimes we might decide to use binary variables especially if the data collection technique is not reliable, for example if we have a software system where number of faults is chosen to be the dependent variable. If we do not have a reliable data collection process, counting inaccuracies can occur, eg. a developer might fix many faults in the same ticket which will be counted as one fault. In this case we can say that there was at least one fault fixed for the component, and we should use binary measures for the dependent variable [36].

Data Collection

The design of many web applications is outdated [30], therefore there is a need to reverse engineer the code to come up with the design. In this research IBM Rational

Rose Enterprise edition [53] is used to reverse engineer the web applications used in the empirical studies to produce the class diagrams. Rational Rose has a visual modeling component, which can create the design artifacts of a software system. The Web Modeler component in Rational Rose supports Conallen's extension for web applications, so it will be used in the class diagram generation process. After the class diagrams are created, Unisys Rose XML [53] is used to export the UML class diagrams into XML Metadata Interchange (XMI) [83]. XMI is an OMG standard for exchanging metadata information via Extensible Markup Language (XML). We will develop a tool that reads the XMI files and extracts the data and computes the metrics from the data in the XMI files.

4.2.3 Maintainability Measurement

The following are the dependent variables that we will use to measure maintainability and study the usefulness of the metrics defined in Table 4.2.1:

- *Understandability and Modifiability*: several studies use maintainability characteristics [13, 43, 45] as maintainability measures. We will measure two important characteristics of design maintainability namely understandability and modifiability. We will measure the following variables:
  - *Understandability Time*: represents the time spent on understanding the system in order to complete the questionnaire
  - *Modifiability Time*: represents the time spent on identifying places for modification and making those modifications
- *Lines of Code Changed*: has been defined as the number of all non-blank non-comment lines of code in a class [50]. In past studies, maintainability has been measured by the number of lines of code changed [68, 69]. In our research we will use absolute net value of the total number of lines added and deleted for components in a class diagram to measure maintainability.

- *Number of Revisions of classes in a class diagram*: in past studies, Number of Revisions was not used to measure maintainability. Based on our knowledge, This is the first study to use number of revisions to measure maintainability.

Data Analysis Technique

The data analysis technique will depend on the type of dependent variable. For binary dependent variables we use the logistic regression analysis technique, and for continuous dependent variables we use multiple linear regression. In this research we have several continuous dependent variables including LOC, Modifiability Time, and Understandability Time, therefore we use the multiple linear regression equation shown in Equation 4.1:

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k \tag{4.1}$$

In this equation we have the following variables:

- x: The x variables are the independent variables which present product design metrics. They are also called predictors or regressors.
- y: The y variable is the dependent variable which presents the variables defined in the previous section
- β: The β is the regression coefficient. It is the average amount of dependent increase when the independent variable increase one unit and other independents variables are kept constant.

The equation assumes a linear relationship between the product metric and the dependent variable. The accuracy of the model can be measured by the coefficient of determination R^2 which is the percentage of the variance in the dependent variable explained by the independent variable [4].

4.2.4 Statistical Modeling Phase

In this phase the maintainability model will be built. When building the model we need to look at influential observations. Influential observations are observations that have a large influence on the regression model. We need to make sure that these observations are correct and not due to a data entry error.

We need to provide descriptive statistics such as the mean, median, maximum, minimum and standard deviation for each of the dependent variables. These statistics will be used in correlation and regression analysis.

Correlation Analysis

In this step the product metric is validated by looking into two aspects of the relationship between the product metric and the dependent variable:

- *Statistical Significance*: the probability of getting an estimated parameter as large as the one actually obtained if the true parameter was zero.

- *Magnitude of Relationship*: tells how much influence the product metric has on the dependent variable.

  Both of the above measures concern the coefficient of the independent variables in the regression model. If the coefficient is statistically significant the metric is validated. For example if the dependent variable is effort and the coefficient is positive this means that the independent variable is a cost and causes an increase in effort. On the other hand, if the coefficient is negative this that the independent variable is a cost saving and causes a decrease in effort [4].

Regression Analysis

Correlation and regression are equally important and a complete analysis of the relationship between the dependent and independent variables includes both. In regression analysis the best validated metrics are selected: those that have the strongest relationship

with the dependent variable and weak correlation to each other. There are two selection techniques for selecting the metrics for the regression modes:

- *Forward Selection*: starts with one independent variable, then other independent variables are added as long as they fulfill a certain statistical criteria
- *Backward Selection*: starts with all dependent variables, one is deleted as it complies with certain statistical criteria.

Prediction Model Evaluation

The most common way of evaluating the prediction model is to use the measure of relative error. The relative error measure is shown in Equation 4.2:

$$RE = \frac{x - y}{y} \tag{4.2}$$

where x is the predicted value and y is the actual value. If RE is negative this means that the model underestimates, if RE is positive the model overestimates and if RE is zero the prediction is perfectly accurate. The result can be multiplied by 100 to get the percentage of deviation from the actual value. For example if RE is 20% this means that the model overestimates by 20%. Another measure that can be used is the absolute relative error as show in Equation 6.1:

$$MRE = |\frac{x - y}{y}| \tag{4.3}$$

In the MRE there is no difference between positive and negative measures this might be suitable for some dependent variables where over and underestimation are equivalent [36]. The $MMRE$ is the mean of the MRE, it is one of the most widely used criterion for assessing the performance of software prediction models [80, 61].

4.2.5 Post Modeling Phase

In this phase results will be analyzed and reported. The following shows what is reported and analyzed [36]:

- *Description of the System*: the system must be described clearly. Details of the where the system was used in an educational or professional environment has to be mentioned. The subject size, the type of data, programming language and any tools used must be described.

- *Unit of Observation*: the unit of observation of the metric has to be specified clearly. For example the unit can be a file, a procedure, a class, an attribute.

- *Dependent Variable*: tt is important to describe how the dependent variable was chosen and how it is measured.

- *Descriptive Statistics*: the descriptive statistics for all data sets has to be reported. This includes the mean, minimum, maximum, median and standard deviation.

- *Data Analysis Technique*: the data analysis technique has to be specified and justification why that technique has been chosen must be given.

- *Variable Selection Procedure*: in the results the variable selection technique has to be specified if it is forward or backward selection.

- *Evaluation of Prediction Model*: a description of how the description model is evaluated must be specified,

- *Comparison to Previous Results*: a comparison to previous study should be included to build on knowledge and analyze if the study provided similar or different results.

4.2.6 Model Validation

Software metrics can be acceptable and useful only if they have been proven through a validation process. We will evaluate our model using both theoretical and empirical validation.

Theoretical Validation

We will use the general framework proposed by Briand et al [16] for theoretical validation for the software metrics. The framework defines properties for several measurement concepts such as size, length, complexity, cohesion, and coupling.
A system is defined as a set of elements and a set of relationships between those elements.

Definition 1 *[16]:* ***Representation of Systems and Modules****. A system S will be represented as a pair* $< E, R >$*, where E represents the set of elements of S, and R is a binary relation on* $E(R \subseteq E \times E)$ *representing the relationships between all S elements. Given a system* $S =< E, R >$*, a module* $m =< E_m, R_m >$ *is a module of S if and only if* $E_m \subseteq E$*,* $R_m \subseteq E_m \times E_m$*, and* $R_m \subseteq R$*. The elements of a module are connected to the elements of the rest of the system by incoming and outgoing relationships. The set* $InputR(m)$ *of relationships from elements outside module* m *to those of module* m *is defined as:*

$InputR(m) = \{< e_1, e_2 >\in R \mid e_2 \in E_m$ *and* $e_1 \in E - E_m\}$.

The set $OutputR(m)$ *of relationships from the elements of a module* m *to those of the rest of the system is defined as:*

$OutputR(m) = \{< e_1, e_2 >\in R \mid e_1 \in E_m$ *and* $e_2 \in E - E_m\}$.

For the size metrics three properties are defined: Nonnegativity, Null Value, and Module Additivity as shown in the following definitions:

- **Property Size 1:(Nonnegativity)**. The size of a system $S =< E, R >$ is nonnegative: $Size(S) \geq 0$.
- **Property Size 2:(Null Value)**. The size of a system $S =< E, R >$ is null if E is empty: $E = 0 \Rightarrow Size(S) = 0$
- **Property Size 3:(Module Additivity)**. The size of a system $S =< E, R >$ is equal to the sum of the size of two of its modules $m_1 =< E_{m1}, R_{m1} >$ and $m_2 =< E_{m2}, R_{m2} >$ such that any element of S is an element of either m_1 or m_2:

$(m_1 \subseteq S$ and $m_2 \subseteq S$ and $E = E_{m1} \bigcup E_{m2}$ and $E_{m1} \bigcap E_{m2} = 0) \Rightarrow Size(S) = Size(m_1) + Size(m_2)$.

The size metrics will be defined based on the above definitions. The other metrics will be defined based on their properties defined in [16].

Empirical Validation

We will carry out different empirical studies using industrial and open source web applications. We will conduct the following forms of empirical studies Surveys, Case Studies, and Controlled Experiments [62, 98]:

- *Survey*: research in the large and the past.
- *Case Study*: research in the typical since it studies real projects.
- *Controlled Experiment*: research in the small since the experiment is conducted on a small population. We will use the $MMRE$ to assess performance of the prediction model. The $MMRE$ is one of the most widely used criterion for assessing the performance of software prediction models [80, 61].

This research will carry out some empirical studies in an educational institution. The experiment will contain the following steps:

- *Preparing Materials*: decide on the material of the experiment, eg. design artifacts of web applications will be used for the experiments. They should be a representation of what is used in real life and cover a wide range of web application technologies.
- *Procedure*: the subjects must be chosen carefully, and must have similar estimated performance. Previous academic history and a questionnaire will be used to assign subjects to teams. Clear documentation on all steps of the experiment will be provided to all subjects. Also, a tutorial and sample tasks will be given to subjects before the start of the actual experiment.

- *Tasks*: the experiment will have three tasks. In the first task the subjects will complete a questionnaire about overall understanding, structure of design and specific question about design. In the second task the subjects will do an impact analysis and show the modifications that must be made in the code. In the third step the subjects complete a debriefing questionnaire which includes personal details, experience, opinions with respect to a subjects motivation and the performance approach they adopt to complete the most difficult tasks.

It is important to look into both internal, and external validity to make sure the results are valid. Internal validity is the degree to which conclusions can be drawn about the effect of the independent variables on the dependent variable. A threat to internal validity is the way students are allocated to groups. We might have one group that has excellent students while another has average students. This threat can be eliminated by correctly allocating students to groups. External validity is the degree to which the results of the study can be generalized to other settings. Another threat is that students will not be as experienced as professionals. Another threat is the material used must be a representation to what is used in real life [13].

Ethics in Empirical Studies

There is little attention to ethical issues in the software engineering world [92]. It is important to use ethics in research in order to keep the research going in the right direction. If ethical guidelines are not followed, subjects might quit participation in the experiment. It is encouraged to follow the following three ethical principles when doing empirical research [92]:

- *Informed Consent*: This means providing information to subjects before the experiment starts. This can include, the research purpose, benefits of the research to the world and to subjects. In addition to that, it must be clear that subjects are participating in the experiment with their free choice without any compulsion. Subjects can terminate participation in the experiment any time without giving

any explanation. It should be explained to subjects that there will be no risk of getting a bad grade if they do not participate in the experiment.

- *Scientific Value*: This includes describing the value of the research project and making sure the experimental results are valid. The validity is accomplished by checking internal and external validity.

- *Confidentiality*: This means anonymity of the subjects so that no one can identify them. It is encouraged to have a signed consent form from the subjects before conducting the experiment.

In our experiments we did not have to get a signed consent form from the subjects before the experiments. But we did make sure that subjects participated with their free choice. It was made clear that participating in the experiments will not have an effect on students grade.

5
Theoretical Validation

Software measures play an important role in controlling software maintenance practices and products. It is important that these measures are valid. This chapter will show how to define and validate the UML metrics shown in Table 5.5 using two validation frameworks one proposed by Kitchenham [63] and one proposed by Briand [16].

5.1 Kitchenham's Validation Framework

Kitchenham proposes a validation framework for software metrics [63]. The framework's goal is to show researchers:

- How to validate a measure.
- How to evaluate the validation of others.
- When to apply a certain measure.

In [63] a structure model of software measurement is proposed. The structural model is composed of the following elements: Entities, attributes, relationships, units, scale types, values, and measurement instrument. A classification of the metrics based on the measurement structure proposed by Kitchenham is shown in Table 5.6. Table 5.7 and Table 5.8 validate the metrics based on the following conditions: Attribute Validity, Unit Validity, Instrument Validity, and Protocol Validity. For more details on the framework refer to Section 2.4.2.

Metric Type	Metric Name	Description
Size	NServerP	Total number of server pages
Size	NClientP	Total number of client pages
Size	NWebP=(NServerP + NClientP)	Total number of web pages
Size	NFormP	Total number of form pages
Size	NFormE	Total number of form elements
Size	NClientScriptsComp	Total number of client scripts components
Size	NServerScriptsComp	Total number of server scripts components
Size	NC	Total number of classes
Size	NA	Total number of attributes
Size	NM	Total number of methods
Structural Complexity	NAssoc	Total number of associations
Structural Complexity	NAgg	Total number of aggregation relationships
Structural Complexity	NLinkR	Total number of link relationships
Structural Complexity	NSubmitR	Total number of Submit relationships times NFormE
Structural Complexity	NbuildsR	Total number of builds relationships times (NServerScriptsComp + NClientScriptsComp)
Structural Complexity	NForwardR	Total number of forward relationships
Structural Complexity	NIncludeR	Total number of include relationships
Coupling	WebControlCoupling = (NLinkR + NSubmitR + NbuildsR + NForwardR + NIncludeR)/ NWebP)	Number of relationships over number of web pages
Coupling	WebDataCoupling = (NFormE/NServerP)	Number of data exchanged over number of server pages
Coupling	EntropyCoupling = $1/n \times (-\log 1/(1+m))$	where n is total number of elements and m is total number of relationships
Cohesion	EntropyCohesion	total entropy coupling of the application / entropy coupling of one class diagram

Table 5.5: Web Application Class Diagram Metrics

5.2 Briand's Validation Framework

This research also uses the general framework proposed by Briand *et al* [16] for theoretical validation for the software metrics. The framework defines properties for several measurement concepts such as size, length, complexity, cohesion, and coupling. The Framework is generic and not specific to any software artifact. In addition to that, it is based on precise mathematical concepts. The objects of study in the framework are defined as a system which consists of a set of elements and a set of relationships between them.

Definition 2 *[16]:* ***Representation of Systems and Modules****. A system S will be represented as a pair* $< E, R >$*, where E represents the set of elements of S, and R is a binary relation on* $E(R \subseteq E \times E)$ *representing the relationships between all S elements. Given a system* $S =< E, R >$*, a module* $m =< E_m, R_m >$ *is a module of S if and only if* $E_m \subseteq E$*,* $R_m \subseteq E_m \times E_m$*, and* $R_m \subseteq R$*. The elements of a module are connected to the elements of the rest of the system by incoming and outgoing relationships. The set* $InputR(m)$ *of relationships from elements outside module* m *to those of module* m *is defined as:*

$InputR(m) = \{< e_1, e_2 >\in R \mid e_2 \in E_m$ *and* $e_1 \in E - E_m\}$.

The set $OutputR(m)$ *of relationships from the elements of a module* m *to those of the rest of the system is defined as:*

$OutputR(m) = \{< e_1, e_2 >\in R \mid e_1 \in E_m$ *and* $e_2 \in E - E_m\}$.

We will define some of the operations and modules used in the following sections:
Inclusion: Let module $m1 =< E_{m1}, R_{m1} >$, and $m2 =< E_{m2}, R_{m2} >$, module $m1$ is included in module $m2$ meaning $m1 \subseteq m2$ if $E_{m1} \subseteq E_{m2}$ and $R_{m1} \subseteq R_{m2}$.
Union: Let module $m1 =< E_{m1}, R_{m1} >$, and $m2 =< E_{m2}, R_{m2} >$, the union of module $m1$ and $m2$ is $m1 \bigcup m2$ which is equal to $< E_{m1} \bigcup E_{m2}, R_{m1} \bigcup R_{m2} >$.
Intersection: Let module $m1 =< E_{m1}, R_{m1} >$, and $m2 =< E_{m2}, R_{m2} >$, the intersection of module $m1$ and $m2$ is $m1 \bigcap m2$ which is equal to $< E_{m1} \bigcap E_{m2}, R_{m1} \bigcap R_{m2} >$.

In the next sections the metrics in Table 5.5 are defined based on the size, complexity, coupling, and cohesion properties defined in [16].

5.2.1 Size Metrics

For the size metrics three properties are defined: Nonnegativity, Null Value, and Module Additivity as shown in the following definitions:

- **Property Size 1:(Nonnegativity)**. The size of a system $S =< E, R >$ is nonnegative: $Size(S) \geq 0$.
- **Property Size 2:(Null Value)**. The size of a system $S =< E, R >$ is null if E is empty: $E = 0 \Rightarrow Size(S) = 0$

- **Property Size 3:(Module Additivity)**. The size of a system $S =< E, R >$ is equal to the sum of the size of two of its modules $m_1 =< E_{m1}, R_{m1} >$ and $m_2 =< E_{m2}, R_{m2} >$ such that any element of S is an element of either m_1 or m_2: $(m_1 \subseteq S$ and $m_2 \subseteq S$ and $E = E_{m1} \bigcup E_{m2}$ and $E_{m1} \bigcap E_{m2} = 0) \Rightarrow Size(S) = Size(m_1) + Size(m_2)$.

The following size metrics (**NServerP**, **NClientP**, **NWebP**, **NFormP**, **NFormE**, **NClientScriptsComp**, **NServerScriptsComp**, **NC**, **NA**, **NM**) satisfy properties 1-3. We will take one of the size metrics and prove that it satisfies properties 1-3. The validation of the other size metrics is the same.

NServerP

Definitions: A system $S =< E, R >$ is defined as a class diagram where E are all the classes in the class diagram and R are all the relationships in the class diagram. The size S of the class diagram is a function $Size(S)$ that is defined as the number of server pages in the class diagram. $Size(S)$ is a function that is characterized by three properties Nonnegativity, Null Value, and Module Additivity.

Nonnegativity: The number of server pages is obtained as a sum of all server pages in the class diagram which is a sum of nonnegative numbers so the nonnegativity property holds.

Null Value: When there is no server page in the class diagram the sum of server pages is equal to null.

Module Additivity: Assume S is a class diagram, E are the set of server pages in S, and R are the set of relationships between the server pages. Class diagram S is partitioned in two disjoint class diagrams $m1$, and $m2$. E_{m1} are the set of server pages in $m1$, and R_{m1} are the set of relationships between server pages in $m1$. E_{m2} are the set of server pages in $m2$, and R_{m2} are the set of relationships between server pages in $m2$. Both class diagrams do not have any elements in common.

Let $S =< E, R >$, $m_1 =< E_{m1}, R_{m1} >$, and $m_2 =< E_{m2}, R_{m2} >$ such that the following conditions hold: $m_1 \subseteq S$, $m_2 \subseteq S$, $E = E_{m1} \bigcup E_{m2}$ and $E_{m1} \bigcap E_{m2} = 0$.

When partitioning the class diagram S into two disjoint class diagrams $m1$ and $m2$ the total number of server pages in S stays the same since both $m1$ and $m2$ do not have any server pages in common. $Size(S)$ is equal to all server pages in S, $Size(m_1)$ is equal to all server pages in $m1$, and $Size(m_2)$ is equal to all server pages in $m2$. It is clear that the total number of server pages in S is equal to the sum of server pages in $m1$ and $m2$. This means that $Size(S) = Size(m_1) + Size(m_2)$.

Figure 5.1 shows how the module additivity property is applied to a sample class diagram S. S is partitioned in two disjoint modules $m1$, and $m2$. Both modules do not have any elements in common as shown in the diagram. $Size(S)$ = 2 which is equal to calendar.jsp + edit_task_popup.jsp. $Size(m_1)$ = 1 which is equal to edit_task_popup.jsp. $Size(m_2)$ = 1 which is equal to calendar.jsp. We can see that the $Size(S) = Size(m_1) + Size(m_2)$. The NServerP for S is unchanged after the partitioning.

5.2.2 Complexity Metrics

For the complexity metrics five properties are defined: Nonnegativity, Null Value, Symmetry, Module Monotonicity, and Disjoint Module Additivity as shown in the following definitions:

- **Property Complexity 1:(Nonnegativity)**. The complexity of a system $S =< E, R >$ is nonnegative: $Complexity(S) \geq 0$.

- **Property Complexity 2:(Null Value)**. The complexity of a system $S =< E, R >$ is null if E is empty: $E = 0 \Rightarrow Complexity(S) = 0$

- **Property Complexity 3:(Symmetry)**. The complexity of a system $S =< E, R >$ is not related to the convention used to represent the relationships between elements:

  $(S =< E, R >$ and $S^{-1} =< E, R^{-1} >) \Rightarrow Complexity(S) = Complexity(S^{-1})$. S is a system where the relationships R are in the incoming direction, and S^{-1} is a system where the relationships R^{-1} are in the outgoing direction. The complexity should not be sensitive to the direction of arcs representing system relationships.

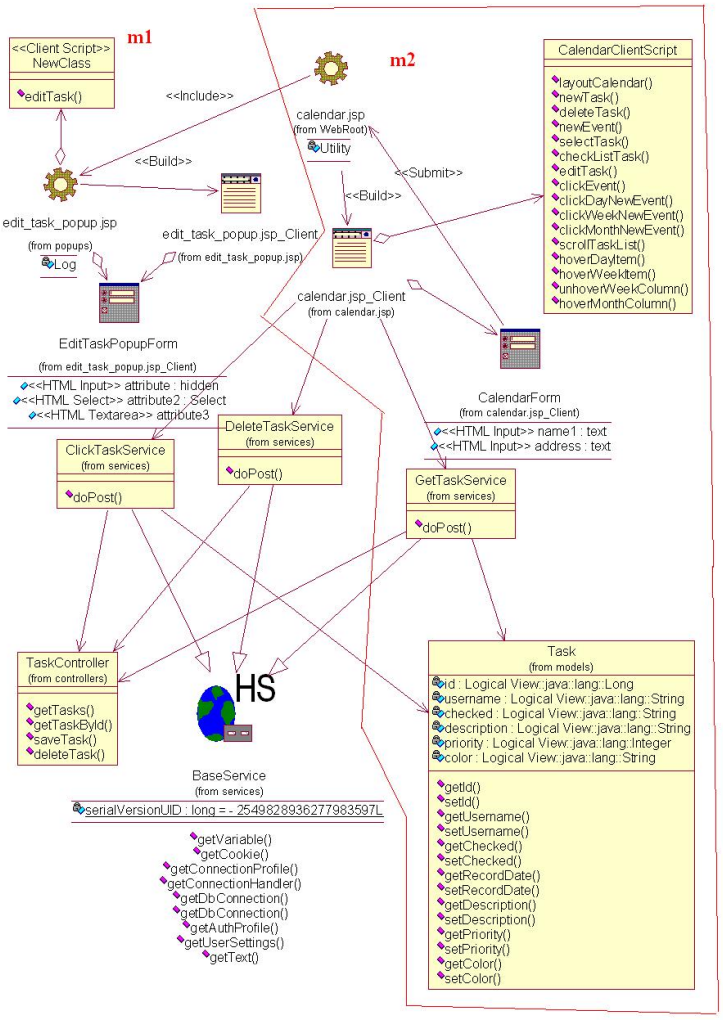


Figure 5.1: Sample Class diagram S

- **Property Complexity 4:(Module Monotonicity)**. The complexity of a system $S =< E, R >$ is equal to or greater than the sum of the complexities of two of its modules with no relationships in common:

  $(S =< E, R >, m_1 =< E_{m1}, R_{m1} >, m_2 =< E_{m2}, R_{m2} >, m_1 \bigcup m_2 \subseteq S$ and $R_{m1} \bigcap R_{m2} = 0) \Rightarrow Complexity(S) \geq Complexity(m_1) + Complexity(m_2)$

- **Property Complexity 5:(Disjoint Module Additivity)**. The complexity of a system

  $S =< E, R >$ composed of two disjoint modules m_1 or m_2 is equal to the

sum of the complexity of the two modules: $S =< E, R >$, $S = m_1 \bigcup m_2$, and $m_1 \bigcap m_2 = 0) \Rightarrow Complexity(S) = Complexity(m_1) + Complexity(m_2)$.

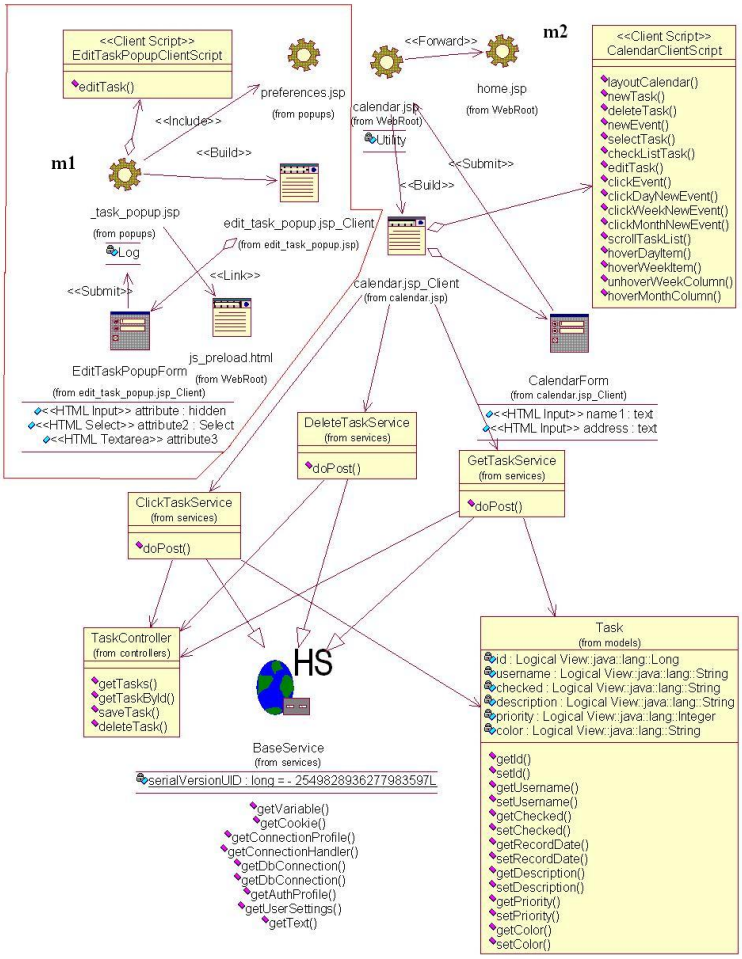


Figure 5.2: Class diagram S with $m1$, $m2$ disjoint modules

The following complexity metrics **NAssoc**, **NAgg**, **NLinkR**, **NSubmitR**, **NbuildsR**, **NForwardR**, and **NIncludeR** satisfy properties 1-5. We will take one of the complexity metrics and prove that it satisfies properties 1-5. The validation of the other complexity metrics is the same.

Figure 5.2 is a class diagram S that is partitioned in two disjoint class diagrams $m1$, and $m2$. Both class diagrams do not have any relationships and elements in common as

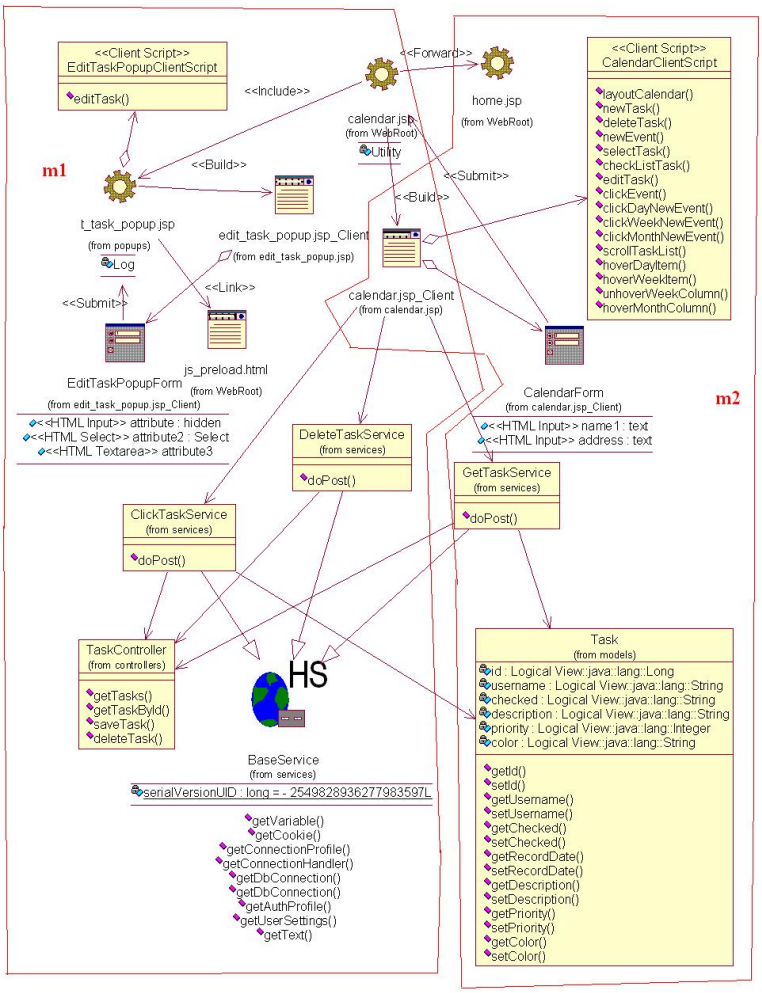


Figure 5.3: Class diagram S showing Module Monotonocity

shown in Figure 5.2. The class diagram is used show the Disjoint Module Additivity of the complexity metrics. Figure 5.3 is a class diagram S that is partitioned in two class diagrams $m1$, and $m2$. Neither class diagrams have any relationships in common. They share the calendar.jsp_client element as shown in Figure 5.3.

NAssoc

Definitions: A system $S =< E, R >$ is defined as a class diagram where E are all the classes in the class diagram and R are all the relationships in the class diagram. The complexity S of the class diagram is a function $Complexity(S)$ that is defined as the sum of all associations in the class diagram. $Complexity(S)$ is a function that is charac-

terized by five properties Nonnegativity, Null Value, Symmetry, Module Monotonicity and Disjoint Module Additivity.

Nonnegativity: The number of associations is obtained as a sum of all associations in the class diagram which is a sum of nonnegative numbers so the nonnegativity property holds.

Null Value: When there are no associations in the class diagram the sum of associations is equal to null.

Symmetry: The complexity is independent of the convention chosen to represent the association relationships. Let ($S =< E, R >$ and $S^{-1} =< E, R^{-1} >$) where S and S^{-1} are two class diagrams that have the same number of elements and association relationships. Let E be the set of elements in S and S^{-1}. R is chosen to be all outgoing associations and R^{-1} is chosen to be all incoming associations. The $Complexity(S)$ is equal to NAssoc which is equal to the number of all outgoing associations R. The $Complexity(S^{-1})$ is equal to NAssoc which is equal to the number of all incoming associations R^{-1}. The number of outgoing associations R is equal to the number of incoming associations R^{-1} so $Complexity(S) = Complexity(S^{-1})$.

An example of applying the symmetry property to NAssoc is shown in Figure 5.2 where $Complexity(S) = Complexity(S^{-1})$ = 11.

Module Monotonicity: Assume S is a class diagram, E are the set of elements in S, and R are the set of association relationships between the elements. Class diagram S is partitioned in two class diagrams $m1$, and $m2$. E_{m1} are the elements in $m1$, and R_{m1} are the set of association relationships between elements in $m1$. E_{m2} are the set of elements in $m2$, and R_{m2} are the set of association relationships between elements in $m2$. We have $m_1 \bigcup m_2 \subseteq S$ and $R_{m1} \bigcap R_{m2} = 0$. All the association relationship in R also exist in R_{m1} and R_{m2}. There is a possibility that some association relationships exist in R but do not exist in R_{m1} or R_{m2} since we have $m_1 \bigcup m_2 \subseteq S$. This means that $R \geq R_{m1} + R_{m2}$. The $Complexity(S)$ is equal to NAssoc which is equal to the number of all associations R in S. The $Complexity(m_1)$ is equal to NAssoc which is equal to the number of all associations R_{m1} in $m1$. The $Complexity(m_2)$ is equal to NAssoc which is equal to the number of all associations R_{m2} in $m2$. Since we have $R \geq R_{m1} + R_{m2}$ this means that

$Complexity(S) \geq Complexity(m_1) + Complexity(m_2)$.

An example of applying the module monotonicity property to NAssoc is shown in Figure 5.3 where $Complexity(S)$ = 11, while $Complexity(m_1)$ = 6, and $Complexity(m_2)$ = 2. One can see that $Complexity(S) \geq Complexity(m_1) + Complexity(m_2)$.

Disjoint Module Additivity: Assume S is a class diagram, E are the set of elements in S, and R are the set of association relationships between the elements. Class diagram S is partitioned in two disjoint class diagrams $m1$, and $m2$. E_{m1} are the elements in $m1$, and R_{m1} are the set of association relationships between elements in $m1$. E_{m2} are the set of elements in $m2$, and R_{m2} are the set of association relationships between elements in $m2$. We have $S = m_1 \bigcup m_2$ and $m_1 \bigcap m_2 = 0$. $m1$ and $m2$ do not have any element in common which means when creating S by combining $m1$ and $m2$ not extra relationships will be added to class diagram S. This means that $R = R_{m1} + R_{m2}$. The $Complexity(S)$ is equal to NAssoc which is equal to the number of all associations R in S. The $Complexity(m_1)$ is equal to NAssoc which is equal to the number of all associations R_{m1} in $m1$. The $Complexity(m_2)$ is equal to NAssoc which is equal to the number of all associations R_{m2} in $m2$. Since we have $R = R_{m1} + R_{m2}$ this means that $Complexity(S) = Complexity(m_1) + Complexity(m_2)$.

An example of applying the disjoint module additivity property to NAssoc is shown in Figure 5.2 where $Complexity(S)$ = 11 which is equal to the sum of associations in m_1 and m_2. $Complexity(m_1)$ = 0 which is equal to the sum of all associations in m_1. $Complexity(m_2)$ = 11 which is equal to the sum of all associations in m_2. We can see that the $Complexity(S) = Complexity(m_1) + Complexity(m_2)$. The NAssoc for S is unchanged after the partitioning.

5.2.3 Coupling Metrics

For the coupling metrics five properties are defined: Nonnegativity, Null Value, Monotonicity, Merging of Modules and Disjoint Module Additivity as shown in the following definitions:

- **Property Coupling 1:(Nonnegativity)**. The coupling of a system $S =< E, R >$

is nonnegative: $Coupling(S) \geq 0$.

- **Property Coupling 2:(Null Value)**. The coupling of a system $S =< E, R >$ is null if R is empty: $R = 0 \Rightarrow Coupling(S) = 0$

- **Property Coupling 3:(Monotonicity)**. We have $S_1 =< E, R_1 >$ and $S_2 =< E, R_2 >)$ where S_1 and S_2 are two systems that have the same number of elements. Let E be the set of elements in S_1 and S_2. R_1 are the set of relationships between the elements in S_1 and R_2 are the set of relationships between the elements in S_2. Let R_2 = $R_1 + 1$. This property shows that adding a relationship does not decrease the coupling which means we will have $Coupling(S_1) \leq Coupling(S_2)$.

- **Property Coupling 4:(Merging of Modules)**. If two modules m_1, m_2 are merged to form a new module m_3 which is the union of the two modules m_1, m_2 then the coupling of module m_3 is not greater than the sum of the coupling of m_1, m_2. Then $Coupling(m_1) + Coupling(m_2) \geq Coupling(m_3)$

- **Property Coupling 5:(Disjoint Module Additivity)**. The coupling of a system $S =< E, R >$ composed of two disjoint modules m_1 or m_2 is equal to the sum of the coupling of the two modules where no relationships exists between the elements of the two modules: $Coupling(S) = Coupling(m_1) + Coupling(m_2)$.

EntropyCoupling

Definitions: A system $S =< E, R >$ is defined as a class diagram where E are all the classes in the class diagram and R are all the relationships in the class diagram. The coupling S of the class diagram is a function $Coupling(S)$ (EntropyCoupling) that is defined as $1/n \times (-\log 1/(1 + R))$ where n is total number of elements and R is total number of relationships. $Coupling(S)$ is a function that is characterized by five properties Nonnegativity, Null Value, Monotonicity, Merging of Modules and Disjoint Module Additivity. EntropyCoupling satisfies properties 1-4 but did not satisfy property 5 (Disjoint Module Additivity).

Nonnegativity: EntropyCoupling is obtained by using the following equation: $1/n \times (-\log 1/(1 + R))$ where n is total number of elements and R is total number of re-

lationships. Both are a sum of nonnegative numbers. If $R = 0$ then $EntropyCoupling = 0$. If $R > 0$ then EntropyCoupling will always be a nonnegative number since $(-\log 1/(1+R))$ will always be positive for $R > 0$.

Null Value: When there are no relationships in the class diagram which means $R = 0$ EntropyCoupling is equal to 0. We have $(-\log 1/(1+0))$ = $(-\log 1/1)$ = 0.

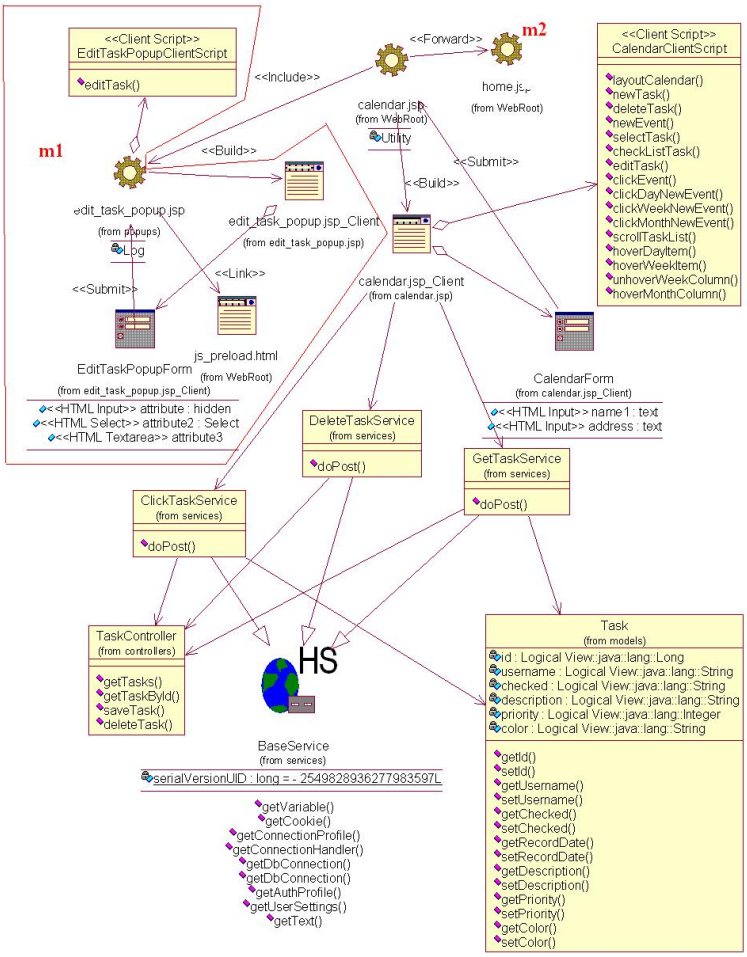


Figure 5.4: Class diagram S showing Module Monotonocity for Coupling

Monotonicity: Assume $S_1 =< E, R_1 >$ and $S_2 =< E, R_2 >$) where S_1 and S_2 are two class diagrams that have the same number of elements. Let E be the set of elements in S_1 and S_2. R_1 are the set of relationships between the elements in S_1 and R_2 are the set of relationships between the elements in S_2. Let R_2 = $R_1 + 1$. The number of rela-

tionships in R_2 are more than the number of relationships in R_1. We have $Coupling(S_1)$ = $1/n \times (-\log 1/(1+R_1))$ and $Coupling(S_2)$ = $1/n \times (-\log 1/(1+R_2))$. We have R_2 = $R_1 + 1$ so $Coupling(S_2)$ = $1/n \times (-\log 1/(1+R+1))$ = $1/n \times (-\log 1/(R+2))$.

We can see that $1/n \times (-\log 1/(1+R_1)) \leq 1/n \times (-\log 1/(R+2))$ which means that $Coupling(S) \leq Coupling(S_2)$.

An example of applying the monotonicity property to EntropyCoupling is shown in Figure 5.4 where adding the include relationship to module m_1 between edit_task_popup.jsp and calendar.jsp does not decrease its coupling.

Merging of Modules: let $m_1 =< E_1, R_1 >$, $m_2 =< E_2, R_2 >$, $m_3 =< E_3, R_3 >$. Let E_1, E_2, E_3 be the set of elements in m_1, m_2, m_3 respectively. Let R_1, R_2, R_3 be the set of elements in m_1, m_2, m_3 respectively. Let m_3 be the union of the two class diagrams m_1, m_2.

We have $Coupling(m_1)$ = $1/n \times (-\log 1/(1+R_1))$,

$Coupling(m_2)$ = $1/n \times (-\log 1/(1+R_2))$, $Coupling(m_3)$ = $1/n \times (-\log 1/(1+R_3))$.

We need to show that

$Coupling(m_1) + Coupling(m_2) \geq Coupling(m_3)$

which means that $1/n \times (-\log 1/(1+R_1))$ + $1/n \times (-\log 1/(1+R_2)) \geq 1/n \times (-\log 1/(1+R_3))$. We can show that $1/(1+R_1) + 1/(1+R_2) \geq 1/(1+R_3) \Rightarrow$

$(1+R_3)/(1+R_1) + (1+R_3)/(1+R_2) \geq (1+R_3)/(1+R_3) \Rightarrow$

$(1+R_3)/(1+R_1) + (1+R_3)/(1+R_2) \geq 1$. We know that $R_3 \geq R_1$, and $R_3 \geq R_2$ since m_3 is the union of the two class diagrams m_1, m_2, the number of relationship in m_3 is greater or equal than the number of relationships in m_1 and m_2 . This means that $(1+R_3)/(1+R_1) \geq 1$, and $(1+R_3)/(1+R_2) \geq 1 \Rightarrow Coupling(m_1)+Coupling(m_2) \geq Coupling(m_3)$.

Figure 5.6 shows the result of combining three classes ClickTaskService, DeleteTaskService, and GetTaskService into CombineTaskService. It is clear that $Coupling(m_1)+Coupling(m_2) \geq Coupling(m_3)$

Disjoint Module Additivity: For the disjoint module additivity property we need to show that $Coupling(m3) = Coupling(m_1) + Coupling(m_2)$. We will use the same definitions from the previous property *Merging of Modules* except no relationships exist

between the elements of the class diagrams $m_1, m_2 \Rightarrow R_3 = R_1 + R_2$. We need to show that

$1/(1 + R_1) + 1/(1 + R_2) = 1/(1 + R_3) \Rightarrow$

$(1 + R_3)/(1 + R_1) + (1 + R_3)/(1 + R_2) = (1 + R_3)/(1 + R_3) \Rightarrow$

$(1 + R_3)/(1 + R_1) + (1 + R_3)/(1 + R_2) = 1$. When looking at left hand side of the equation we can use $R_3 = R_1 + R_2 \Rightarrow (1 + R_2 + R_1)/(1 + R_2)$ is greater than 1 which is greater than right hand side of equation. This means that the disjoint module additivity is not satisfied.

The following coupling metrics **WebControlCoupling**, **WebDataCoupling** satisfy properties 1-5. We will take **WebControlCoupling** and prove that it satisfies properties 1-5. The validation of the **WebDataCoupling** metric is the same.

WebControlCoupling

Definitions: A system $S =< E, R >$ is defined as a class diagram where E are all the classes in the class diagram and R are all the relationships in the class diagram. The coupling S of the class diagram is a function $Coupling(S)$ (WebControlCoupling) that is defined as R/E where R is total number of relationships and E is total number of web pages in the class diagram. $Coupling(S)$ is a function that is characterized by five properties Nonnegativity, Null Value, Monotonicity, Merging of Modules and Disjoint Module Additivity.

Nonnegativity: WebControlCoupling is obtained by using the following equation: R/E where R is total number of relationships and E is total number of web pages in the class diagram. Both are nonnegative numbers, WebControlCoupling will always be a nonnegative.

Null Value: When there are no relationships in the class diagram which means $R = 0$ WebControlCoupling is equal to null.

Monotonicity: Assume $S1 =< E, R_1 >$ and $S2 =< E, R_2 >$ where $S1$ and $S2$ are two class diagrams that have the same number of elements. Let E be the set of elements in $S1$ and $S2$. R_1 are the set of relationships between the elements in $S1$ and $R2$ are the set of relationships between the elements in $S2$. Let R_2 = $R_1 + 1$. The number of relationships in R_2 are more than the number of relationships in R_1. We

have $Coupling(S1)$ = R_1/E and $Coupling(S2)$ = R_2/E. We have $R2$ = $R + 1$ so $Coupling(S2)$ = $(R_1 + 1)/E$.

We can see that $R_1/E \leq (R_1+1)/E$ which means that $Coupling(S1) \leq Coupling(S2)$.

Merging of Modules: let $m_1 =< E, R_1 >$, $m_2 =< E, R_2 >$, $m_3 =< E, R_3 >$. Let E be the set of elements in m_1, m_2, m_3. Let R_1, R_2, R_3 be the set of elements in m_1, m_2, m_3 respectively. Let m_3 be the union of the two class diagrams m_1, m_2. We have $Coupling(m_1)$ = R_1/E, $Coupling(m_2)$ = R_2/E, $Coupling(m_3)$ = R_3/E. We need to show that

$Coupling(m_1) + Coupling(m_2) \geq Coupling(m_3)$

which means that R_1/E + $R_2/E \geq R_3/E$. We know that m_3 is the union of the two class diagrams m_1, m_2, so the number of relationships in m_3 has to be less or equal to the number of relationships in m_1+m_2. This means that $Coupling(m_1)+Coupling(m_2) \geq Coupling(m_3)$.

Disjoint Module Additivity: For the disjoint module additivity property we need to show that $Coupling(m3) = Coupling(m_1) + Coupling(m_2)$. We will use the same definitions from the previous property *Merging of Modules* except no relationships exist between the elements of the class diagrams $m_1, m_2 \Rightarrow R_3 = R_1 + R_2$. We need to show that

R_1/E + $R_2/E \geq R_3/E$. We have $R_3 = R_1 + R_2$ which we can substitute in the right hand side of the equation to become $(R_1 + R_2)/E$. We can see that both sides of the equation are equal which means $Coupling(m_3) = Coupling(m_1) + Coupling(m_2)$.

5.2.4 Cohesion Metrics

For the cohesion metrics four properties are defined: Nonnegativity, Null Value, Monotonicity, and Merging of Modules as shown in the following definitions:

- **Property Cohesion 1:(Nonnegativity and Normalization)**. The cohesion of a module belongs to a specific interval [0,Max].

- **Property Cohesion 2:(Null Value)**. The cohesion of a module $m =< E, R >$ is null if R is empty: $R = 0 \Rightarrow Cohesion(m) = 0$ This means if the set of

intramodule relationships is empty the cohesion of the module is 0.

- **Property Cohesion 3:(Monotonicity)**. We have two modules $m_1 =< E_{m1}, R_{m1} >$, and $m_2 =< E_{m2}, R_{m2} >$ such that E is the set of elements in m_1 and m_2. R_{m1} are the set of relationships between the elements in m_1 and R_{m2} are the set of relationships between the elements in m_2. Let $R_2 = R_1 + 1$. then $Cohesion(m_1) \leq Cohesion(m_2)$.

- **Property Cohesion 4:(Merging of Modules)**. If two unrelated modules m_1, m_2 are merged to form a new module m_3 which is the union of the two modules m_1, m_2 then the cohesion m_3 is not greater than the maximum cohesion of m_1, m_2.

  $max(Cohesion(m_1), Cohesion(m_2)) \geq Cohesion(m_3)$

The Cohesion metrics are defined based on properties 1-4:

EntropyCohesion

Definitions: A system $S =< E, R >$ is defined as a class diagram where E are all the classes in the class diagram and R are all the relationships in the class diagram. The Cohesion S of the class diagram is a function $Cohesion(S)$ (EntropyCohesion) that is defined as $\frac{TotalEntropyCoupling}{(1/n \times (-\log 1/(1+R)))}$ where $TotalEntropyCoupling$ is equal to the sum of the EntropyCoupling (defined in Section 5.2.3) metric for all class diagrams in the system, n is total number of elements and R is total number of relationships. $Cohesion(S)$ is a function that is characterized by four properties Nonnegativity, Null Value, Monotonicity, and Merging of Modules.

Nonnegativity and Normalization: The *EntropyCohesion* of a class diagram is equal to total *EntropyCoupling* of all class diagrams over the *EntropyCoupling* of one class diagram. The *EntropyCohesion* is computed as shown in the following equation: $\frac{\sum_{i=1}^{k}(1/n \times (-\log 1/(1+R)))}{(1/n \times (-\log 1/(1+R)))}$ where n is the total number of elements in the class diagram, R is the total number of relationships in the class diagram, and k is the total number of class diagrams in the application. If $R = 0$ then $EntropyCohesion$ is defined to be 0. If $R > 0$ then EntropyCohesion will always be a nonnegative number since $\sum_{i=1}^{k}(1/n \times (-\log 1/(1+R)))$ will always be positive for $R > 0$.

Null Value: When there are no relationships in the class diagram which means $m = 0$ EntropyCohesion is equal to 0.

Monotonicity: When adding a cohesive interactions to the class diagram this cannot decrease its cohesion. When looking at the EntropyCohesion metric we can see that adding a relationship to the denominator does not decrease the cohesion of the class diagram. we have the following: $\frac{TotalEntropyCoupling}{(1/n\times(-\log 1/(1+R)))} \geq \frac{TotalEntropyCoupling}{(1/n\times(-\log 1/(1+R+1)))}$. We can see that $Cohesion(m_1) \leq Cohesion(m_2)$.

Merging of Modules: When merging the two disjoint class diagrams m_1, and m_2 to form a new class diagram m_3 the number of relationships in m_3 will be equal or more than the number of relationships in m_1, or m_2 alone. Assume $r3$ are the number of relationships in m_3, and $r1$ are the number of relationships in m_1, $r2$ are the number of relationships in m_2.

We have $r3 \geq r1 \Rightarrow \frac{TotalEntropyCoupling}{(1/n\times(-\log 1/(1+r1)))} \geq \frac{TotalEntropyCoupling}{(1/n\times(-\log 1/(1+r3)))}$.

We have $r3 \geq r2$ $\frac{TotalEntropyCoupling}{(1/n\times(-\log 1/(1+r2)))} \geq \frac{TotalEntropyCoupling}{(1/n\times(-\log 1/(1+r3)))}$.

Then $max(Cohesion(m_1), Cohesion(m_2)) \geq Cohesion(m_3)$.

Summary

In this chapter, theoretical validation of our UML design metrics has been accomplished by defining our metrics using two validation frameworks, one proposed by Kitchenham [63] and one proposed by Briand [16]. We have defined all the size, complexity, coupling and cohesion metrics based on the structural model proposed by Kitchenham [63]. We showed that most of the size, complexity, coupling, and cohesion metrics satisfy the properties proposed in Briand's framework [16]. The EntropyCoupling which is a coupling metric did not satisfy the Disjoint Module Additivity property defined in Section 5.2.3.

Table 5.6: Metrics Definition based on Kitchenham's model

Name	Type	Entity	Attribute	Unit	Scale
NServerP	direct	class diagram	number of server pages	server page	interval
NClientP	direct	class diagram	number of client pages	client page	interval
NWebP	direct	class diagram	number of web pages	web page	interval
NFormP	direct	class diagram	number of form pages	form page	interval
NFormE	direct	class diagram	number of form elements	form element	interval
NClientScriptsComp	direct	class diagram	number of client scripts components	client scripts components	interval
NServerScriptsComp	direct	class diagram	number of server scripts components	server scripts components	interval
NC	direct	class diagram	number of classes	class element	interval
NA	direct	class diagram	number of attributes	attribute element	interval
NM	direct	class diagram	number of methods	method element	interval
NAssoc	direct	class diagram	number of association relationships	association relationship	interval
NAgg	direct	class diagram	number of aggregation relationships	aggregation relationship	interval
NLinkR	direct	class diagram	number of link relationships	link relationship	interval
NSubmitR	direct	class diagram	number of submit relationships	submit relationship	interval
NbuildsR	direct	class diagram	number of build relationships	build relationship	interval
NForwardR	direct	class diagram	number of forward relationships	forward relationship	interval
NIncludeR	direct	class diagram	number of include relationships	include relationships	interval
WebControlCoupling	indirect	class diagram	total number of relationships divided by the total number of web pages	control coupling	ratio
WebDataCoupling	indirect	class diagram	total number of form elements divided by the total number of server pages	data coupling	ratio
EntropyCoupling	indirect	class diagram	$1/n \times (-\log 1/(1+m))$ where n is total number of elements and m is total number of relationships in the class diagram	coupling	ratio
EntropyCohesion	indirect	class diagram	total entropy coupling of the application / entropy coupling of one class diagram	cohesion	ratio
WebReusability	indirect	class diagram	total number of include relationships divided by the total number of web pages	reusability	ratio

Table 5.7: Metrics Validation based on Kitchenham's model

Name	Attribute validity	Unit Validity	Instrument Validity	Protocol validity
NServerP	has attribute # of server pages	measured by counting # of server pages	parsing, counting # of server pages in class diagram	counting # of server pages is consistent and unambiguous
NClientP	has attribute # of client pages	measured by counting # of client pages	parsing, counting # of client pages in class diagram	counting # of client pages is consistent and unambiguous
NWebP	has attribute # of web pages	measured by counting # of web pages	parsing, counting # of web pages in class diagram	counting # of web pages is consistent and unambiguous
NFormP	has attribute # of form pages	measured by counting # of form pages	parsing, counting # of form pages in class diagram	counting # of form pages is consistent and unambiguous
NFormE	has attribute # of form elements	measured by counting # of form elements	parsing, counting # of form elements in class diagram	counting # of form elements is consistent and unambiguous
NClient-ScriptsComp	has attribute # of client scripts component	measured by counting # of client scripts components	parsing, counting # of client scripts component in class diagram	counting # of client scripts component is consistent and unambiguous
NServer-ScriptsComp	has attribute # of server scripts	measured by counting # of server scripts components	parsing, counting # of server scripts components in class diagram	counting # of server scripts components is consistent and unambiguous
NC	has attribute # of class elements	measured by counting # of class elements	parsing, counting # of class elements in class diagram	counting # of class elements is consistent and unambiguous
NA	has attribute # of attributes in classes	measured by counting # of attributes in class diagram	parsing, counting # of attributes in class diagram	counting # of attributes is consistent and unambiguous
NM	has attribute # of method elements	measured by counting # of methods in class diagram	parsing, counting # of methods in class diagram	counting # of methods is consistent and unambiguous
NAssoc	has attribute # of association relationships	measured by counting # of association relationship	parsing, counting # of association relationships in class diagram	counting # of association relationships is consistent and unambiguous
NAgg	has attribute # of aggregation relationships	measured by counting # of aggregation relationships	parsing, counting # of aggregation relationships in class diagram	counting # of aggregation relationship is consistent and unambiguous
NLinkR	has attribute # of link relationships	measured by counting # of link relationships	parsing, counting # of link relationships in class diagram	counting # of link relationships is consistent and unambiguous

Table 5.8: Metrics Validation based on Kitchenham's model

Name	Attribute validity	Unit Validity	Instrument Validity	Protocol validity
NSubmitR	has attribute # of submit relationships	measured by counting # of submit relationships	parsing, counting # of submit relationships in class diagram	counting # of submit relationships is consistent and unambiguous
NbuildsR	has attribute # of build relationships	measured by counting # of build relationships	parsing, counting # of build relationships in class diagram	counting # of build relationships is consistent and unambiguous
NForwardR	has attribute # of forward relationships	measured by counting # of forward relationships	parsing, counting # of forward relationships in class diagram	counting # of forward relationships is consistent and unambiguous
NIncludeR	has attribute # of include relationships	measured by counting # of include relationships	parsing, counting # of include relationships in class diagram	counting # of include relationships is consistent and unambiguous
WebControl-Coupling	has attribute # of form elements and # of server pages. # of server pages can not be zero	measured by dividing the total # of relationships by the total # of web pages	parsing, counting total of # of relationships and web pages in class diagram	counting # of relationships and web page is consistent and unambiguous
WebData-Coupling	has attribute # of form elements and number of server pages. # of web pages can not be zero	measured by dividing the total # of form elements by the total # of server pages	parsing, counting # of form elements and server pages in class diagram	counting # of form elements and server pages is consistent and unambiguous
Entropy-Coupling	has attribute # of elements and # of relationships. # of elements can not be zero	$1/n \times (-\log 1/(1+m))$ where n is total number of elements and m is total number of relationships in the class diagram	parsing, counting total of # of relationships and elements in class diagram	counting # of relationships and elements is consistent and unambiguous
Entropy-Cohesion	has attribute # of elements and # of relationships. # of elements can not be zero	total entropy coupling of the application / entropy coupling of the class diagram	parsing, counting total of # of relationships and elements in class diagram	counting # of relationships and elements is consistent and unambiguous
Web-Reusability	has attribute # of include relationships and # of web pages. The # of web pages can not be zero	measured by dividing the total # of include relationships by the total # of web pages	parsing, counting # of include relationships and web pages in class diagram	counting # of include relationships and web pages is consistent and unambiguous

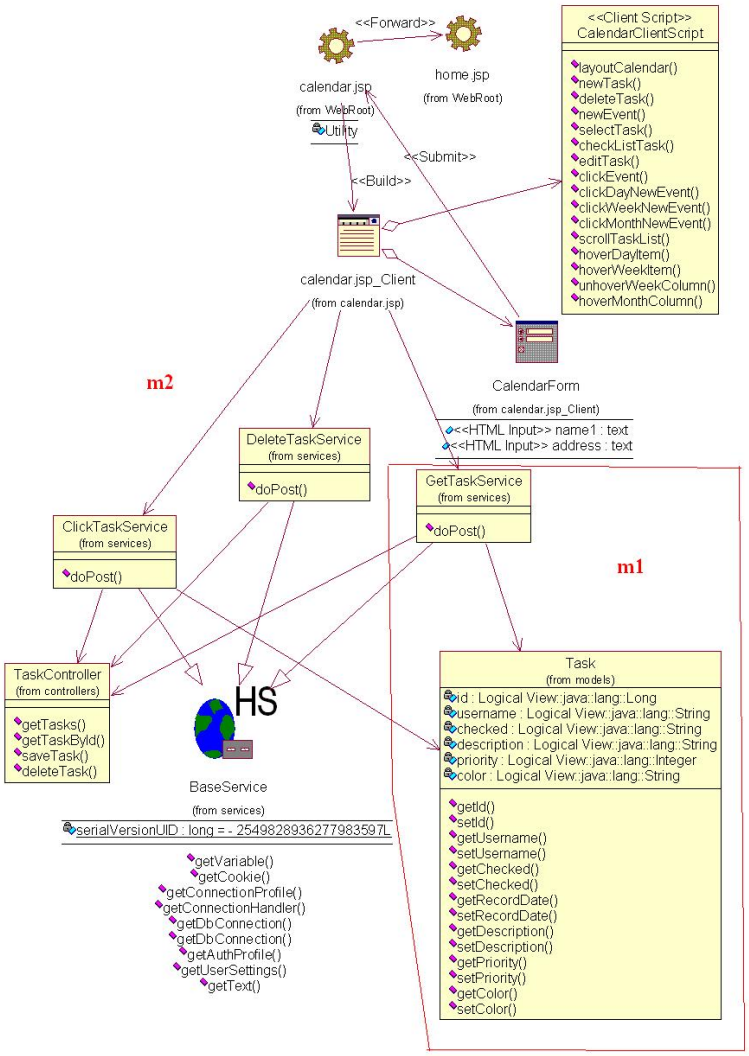


Figure 5.5: Class diagram S showing Modules m1, m2

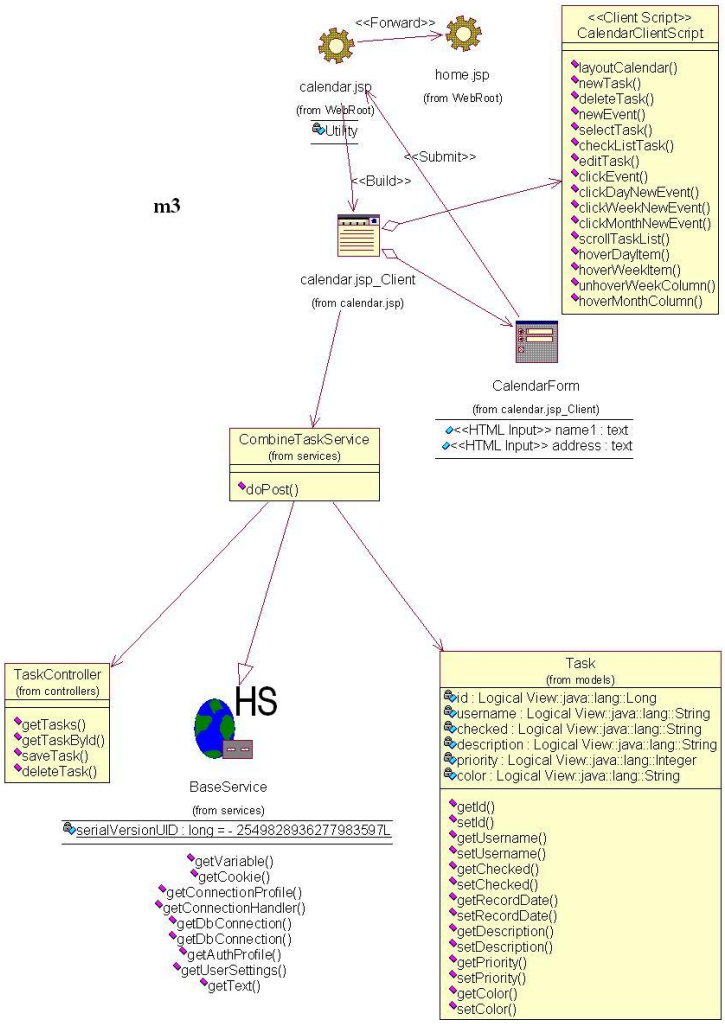


Figure 5.6: Class diagram S showing Merging of Modules m1, m2

6

Empirical Validation

This chapter describes the case studies and experiments that have been used in this research. Initially some preliminary studies were conducted to get an idea on the usefulness of the metrics. Then several studies were conducted using an industrial application and an open source web application to validate the metrics empirically.

We started with an explorative experiment using a PetStore web application. The subjects were students with some experience with web development. One of our goals was to study the relationship between the metrics and maintenance time and to see if there is a possibility for that relationship to exist. Another goal was to see how easy it is to understand and use our new metrics. Our metrics have not been used before but are based on UML. We had to assure the assumptions that the metrics will be easy to use and understand. We got positive results from the PetStore experiment and decided to try our metrics in industrial web applications to see if we still get similar results. We conducted a case study using two industrial web applications to explore the metrics further. We were able to apply our metrics successfully to both industrial web applications and got an indication on the usefulness of using the metrics for improving maintainability in industrial web applications. We were ready to do more analysis on our metrics and study the metric individually or collectively using more statistical analysis. In the last case study we used another industrial web application and used a 3 year data for studying the relationship between our metrics and two variables nRev (Total number of revisions for components in a class diagram), and LOC (Total number of Lines of Code added

and deleted for components in a class diagram).

The following sections provide a detailed description of the empirical studies.

6.1 PetStore Experiment

This study is an explorative experiment conducted at Illinois State university in the US. The subjects are graduate and junior undergraduate students taking a summer course in the Information Technology department

Experiment Context

In this study the pet store web application version 1.3.1 is used for the experiment. The pet store web application is available at the sun web site. The pet store application is a sample application that provides customers with online shopping. A customer can browse the pet store site, look at the catalog and add shopping items to the shopping cart. The customer will need to sign in and create a user account before making a purchase with a credit card. The pet store application is an example of a typical e-commerce web application. The subjects were taking a summer course at the Information Technology department at the university of Illinois. The experiment was given as a bonus quiz in order to make sure that the students have motivation for the experiment. The dependent variable for this study is maintenance time. Maintainability can be defined as
the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment [10, 82].
Maintainability can be measured by measuring some of the sub-characteristics of maintainability such as understandability, analyzability, modifiability and testability. Some studies have measured maintainability by measuring both modifiability and understandability [13, 70, 59]. Understandability is an important sub-characteristics of maintainability, since professionals spend at least half of their time analyzing software to understand it [27]. This study measures two sub-characteristics of maintainability, namely understandability time and modifiability time. Understandability time represents the time spent on understanding the class diagram in order to complete the questionnaire.

Modifiability time represents the time spent on identifying places for modification and making those modifications. The dependent variables in this study are understandability time and modifiability time. The independent variables are the metrics defined in Table 6.9.

Table 6.9: Web Application Class Diagram Metrics

Metric Name	**Description**
NServerP	Total number of server pages
NClientP	Total number of client pages
NWebP=(NServerP + NClientP)	Total number of web pages
NFormP	Total number of form pages
NFormE	Total number of form elements
NLinkR	Total number of link relationships
NSubmitR	Total number of Submit relationships
NbuildsR	Total number of builds relationships
NForwardR	Total number of forward relationships
NIncludeR	Total number of include relationships
NClientScriptsComp	Total number of client scripts components
NServerScriptsComp	Total number of server scripts components
WebControlCoupling = (NLinkR + NSubmitR + NbuildsR + NForwardR + NIncludeR)/ NWebP)	Number of relationships over number of web pages
WebDataCoupling = (NFormE/NServerP)	Number of data exchanged over number of server pages
WebReusability = (NIncludeR/NWebP)	Number of include relationships over number of web pages
NC	Total number of classes
NA	Total number of attributes
NM	Total number of methods
NAssoc	Total number of associations
NAgg	Total number of aggregation relationships

Hypothesis

This study is trying to answer the following question: Is there a relationship between the metrics identified in Table 6.9(NServerP, NClientP, NWebP, NFormP, NFormE, NLinkR, NSubmitR, NBuildsR, NForwardR, NIncludeR, WebControlCoupling, WebDataCoupling, WebReusability) and two sub-characteristics of maintainability, understandability time and modifiability time ? We will show both the null (H_O) and the alternate

hypotheses (H_A).

The following hypotheses are investigated:

- $H1_A$: The higher the size metrics of the class diagram, the higher the understandability time.
- $H1_O$: There is no difference in the understandability time required to understand the class diagram, irrespective of whether the class diagram has high or low size metrics.
- $H2_A$: The higher the size metrics of the class diagram, the higher the modifiability time.
- $H2_O$: There is no difference in the modifiability time required to make changes to systems, irrespective of whether the class diagram has high or low size metrics.
- $H3_A$: The higher the structural complexity metrics of the class diagram, the higher the understandability time.
- $H3_O$: There is no difference in the understandability time required to understand the class diagram, irrespective of whether the class diagram has high or low complexity metrics.
- $H4_A$: The higher the structural complexity metrics of the class diagram, the higher the modifiability time.
- $H4_O$: There is no difference in the modifiability time required to make changes to systems, irrespective of whether the class diagram has high or low complexity metrics.
- $H5_A$: The higher the coupling metrics of the class diagram, the higher the understandability time.
- $H5_O$: There is no difference in the understandability time required to understand the class diagram, irrespective of whether the class diagram has high or low coupling metrics.

- $H6_A$: The higher the coupling metrics of the class diagram, the higher the modifiability time.
- $H6_O$: There is no difference in the modifiability time required to make changes to systems, irrespective of whether the class diagram has high or low coupling metrics.
- $H7_A$: The higher the reusability of the class diagram, the lower the understandability time.
- $H7_O$: There is no difference in the understandability time required to understand the class diagram, irrespective of whether the class diagram has high or low reusability metrics.
- $H8_A$: The higher the reusability of the class diagram, the lower the modifiability time.
- $H8_O$: There is no difference in the modifiability time required to make changes to systems, irrespective of whether the class diagram has high or low reusability metrics.

Data Collection

In this experiment an IBM tool called Rational XDE is used to reverse engineer the pet store web application. Rational XDE combines the visual modeling with a Java forward and reverse engineering tool. The pet store code was imported to the Rational tool and the design for the welcome screen, cartScreen and createCustomer was generated. Three class diagrams were given to students: The welcomeScreen class diagram Figure A.1, the cartScreen class diagram Figure A.2, and the createCustomer class diagram Figure A.3 are all shown in the appendix. The subjects were given instructions, the class diagrams and a questionnaire that was used to collect results and personal information on the subject's experience. The subjects were asked to answer certain questions from the three diagrams to measure the understandability time and the modifiability time. The time was recorded in seconds and minutes. Each subject was asked to record the time

after answering a question. The subjects were asked to start with the welcomeScreen diagram shown in Figure A.1. The welcomeScreen Diagram is not used in calculating the results in order to minimize the learning effects.

Results and Analysis

The main objective in this study was to explore the relationship between the metrics identified in Table 6.9 and understandability and modifiability time. The empirical study was carried out using an experiment in which students at the university of Illinois participated as subjects. Subjects came from a total of three sections with number of students ranging from 10 to 22. A total of fifty three students participated in the experiment. The students participating in this research were graduate and undergraduate who had some experience in web development.

Each student received three class diagrams: welcome screen, cartScreen and createCustomer class diagram. For each class diagram, the student was asked to answer the questions and record the time in minutes and seconds. The welcome screen diagram was given as a tutorial and was not included in the analysis and results. The results are recorded in Table 6.10. The metrics were computed from the class diagrams and the understandability time and modifiability time were computed by recording the time students spent on understanding and modifying the class diagrams. The unit of measurement used is seconds. Table 6.10 shows the mean understandability and modifiability times for the cart and create customer class diagrams. Figure 6.1 shows vertical box-plots for the create customer and cart diagram understandability time, side-by-side for comparison. And Figure 6.2 shows the vertical box-plots for create customer and cart diagram modifiability time. The box-plot is a way to examine the data graphically. It divides the data into five groups: the minimum, the 1st quartile, the median, the 3rd quartile, and the maximum. The box starts from the 1st quartile, and the notch shows the median confidence interval. The line in the middle of the box is the median. The box ends at the 3rd quartile. The diamond shows the mean and the requested confidence interval around the mean.

The create customer class diagram has higher size metrics, structural complexity

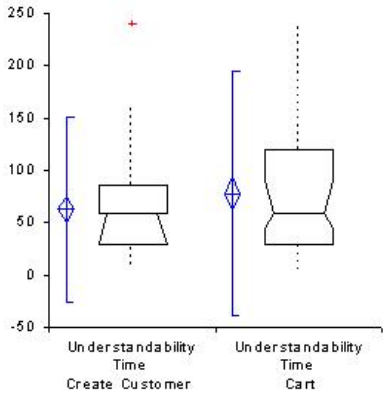


Figure 6.1: Comparative Analysis between Cart and Create Customer Understandability Time

metrics, coupling metrics and lower reusability metrics than the cart class diagram. It is expected to have greater understandability time and modifiability time. The create customer diagram had a lower mean understandability time about 63 seconds compared to the cart diagram about 78 seconds. ANOVA was performed to test the statistical significance of this difference, and the results are as shown in Table 6.11. From the analysis, one can notice that the differences are not statistically significant (P value of 0.1385). Thus we accept the null hypotheses $H1_O$, $H3_O$, $H5_O$, and $H7_O$, and reject the alternative hypotheses $H1_A$, $H3_A$, $H5_A$, and $H7_A$. When looking at the ANOVA results for the modifiability time, one can notice that the mean modifiability time for the create customer diagram is 158 seconds which is higher than the mean modifiability time for the cart diagram around 91 seconds. One can notice that the differences are statistically significant (P value of 0.0004) as shown in Table 6.11. Thus, one can say that a class diagram with higher size metrics, higher structural complexity metrics, higher coupling metrics and lower reusability metrics requires more modifiability time. Thus we can accept the alternative hypotheses $H2_A$, $H4_A$, $H6_A$, and $H8_A$ and reject the null hypotheses $H2_O$, $H4_O$, $H6_O$, and $H8_O$.

As a conclusion, we found that the new UML metrics were easy to use and understand by students. The study gave an indication that there is a possibility for a relationship to exist between our metrics and modifiability time. This study can serve as a basis

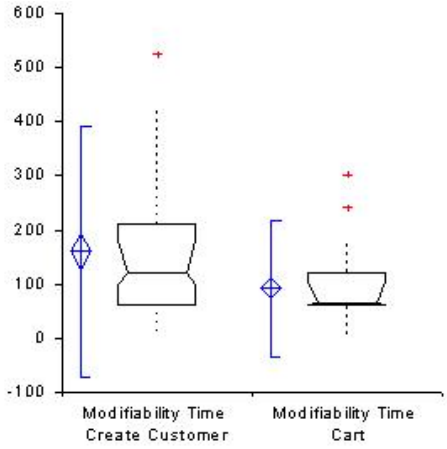


Figure 6.2: Comparative Analysis between Cart and Create Customer Modifiability Time

for future studies, and can provide a first indication of the use of the newly introduced metrics.

Threats to Validity

This study was based on students and this may limit how much the results can be generalized to professional developers. However, many researchers use students as subjects and their results are accepted as valid empirical studies. The students were asked to do the first diagram as a learning diagram in order to minimize the learning effects in this study. Another limitation is the web application used in this experiment. The pet store is a toy application compared to complex web applications. However in this study it was important to give students reasonable tasks which they can understand and solve in a reasonable amount of time. Also if this study can prove that there is a relationship between the metrics for the pet store application and maintainability time then it is likely that a relationship will exist for more complex systems. Another threat to validity is that student's time was self recorded. However the grades were not based on the time taken to accomplish the tasks. This was mentioned clearly in the instructions given to students. Therefore there would be no reason for students to report their time inaccurately.

The results give a first indication of the usefulness of the UML design metrics. The

Table 6.10: Results & Analysis

Metric Type	Metric Name	Create Customer	Cart
Size	NServerP	9	8
Size	NClientP	7	6
Size	NWebP	16	14
Size	NFormP	2	2
Size	NFormE	22	4
Structural Complexity	NLinkR	3	3
Structural Complexity	NSubmitR	2	2
Structural Complexity	NBuildsR	7	6
Structural Complexity	NForwardR	0	0
Structural Complexity	NIncludeR	5	4
Control Coupling	WebControlCoupling	1.06	1.07
Data Coupling	WebDataCoupling	2.44	0.5
Reusability	WebReusability)	0.031	0.29
Understandability Time	Mean Understandability Time	63 sec	78 sec
Modifiability Time	Mean Modifiability Time	158 sec	91 sec

exploratory experiment shows that higher values of size metrics, structural complexity metrics, coupling metrics and lower values of reusability metrics results in higher values of modifiability time. The results did not show statistical significance on the effect of the metrics on understandability time.

6.2 Telecom Web Applications

The proposal of new metrics is not helpful if their practical use is not proved through case studies. We demonstrate our approach using two web applications case studies. The case studies are exploratory in nature and will provide a basis for future research. Both web applications are from the telecommunication operational support system (OSS) domain. Table 6.12 shows the characteristics of both web applications.

Table 6.11: ANOVA Analysis

Name	F	P
Create Customer vs Cart Modifiability Time	13.24	.0004
Create Customer vs Cart Understandability Time	2.22	.1385

Characteristic	**WebApp1**	**WebApp2**
Application Domain	Telecom	Telecom
Age	4	5
Language	Java	Java
Web Server	WebLogic 7.0	Tomcat 4.1
Application Server	WebLogic 7.0	None
Framework	Struts 1.1	None
Database	Oracle	MySql
Configuration Management Tool	CVS	None
Design Tool	Rational Rose	None

Table 6.12: Characteristics of Web Applications

Case Study Context

The first web application is a provisioning application which is used to provision and activate the wireless service in the network. This study refers to the first web application as WebApp1. WebApp1 has around 10,000 users of which 2,500 are concurrent. It is a critical application that is used by customer care advocates to resolve provisioning issues for wireless subscribers. WebApp1 is built using the latest web technologies and frameworks such Struts, and EJBs and uses Oracle for the database.

In order to further evaluate the metrics this methodology is applied to a second web application. This study refers to the second web application as WebApp2. WebApp2 is a fault management application that is used to automate the configuration management for fault management applications. It is used to automate password resets, creation of new users, and creation of configuration management tickets. WebApp2 uses basic web

application technologies such as Javascript, servlets, and mysql database.

Both web applications are considered to be of medium size. Due to space limitations the case studies cannot be presented in detail. This study will analyze the web applications based on system metrics.

Metric Type	**Description**
Size	Total number of server pages (NServerP)
	Total number of client pages (NClientP)
	Total number of web pages (NWebP)=(NServerP + NClientP)
	Total number of form pages (NFormP)
	Total number of form elements (NFormE)
	Total number of client components (style sheet and JavaScript components)(NClientC)
Structural Complexity	Total number of link relationships (NLinkR)
	Total number of Submit relationships (NSubmitR)
	Total number of builds relationships (NbuildsR)
	Total number of forward relationships(NForwardR)
	Total number of include relationships(NIncludeR)
	Total number of use tag relationships(NUseTagR)
Control Coupling	Number of relationships over number of web pages: WebControlCoupling = (NLinkR + NSubmitR + NbuildsR + NForwardR + NIncludeR + NUseTagR)/ NWebP)
Data Coupling	Number of data exchanged over number of server pages: WebDataCoupling = (NFormE / NServerP)
Reusability	Number of include relationships over number of web pages: WebReusability = (NIncludeR / NWebP)

Table 6.13: Web Application Design Metrics

Hypothesis

This study aims to answer the following question: Is there a relationship between the metrics identified in Table 6.13 and maintainability? Since the study is explorative in nature, it measures maintainability in a subjective manner. The maintainability is measured by getting input from the developers on the modifiability maintainability subcharacteristic. The modifiability is based on how easy it is to make changes to the web application. In this context a high modifiability means that it is easier to make changes to the web application, while a low modifiability means that it is difficult to make changes

to the web application. The desire of developers is to have high modifiability since this makes it easier for them to modify and change the web application during maintenance phase.

We will show both the null (H_O) and the alternate hypotheses (H_A). The following hypotheses are investigated: $(H1_A)$: the lower the size metrics of the class diagram, the higher the modifiability. $(H1_O)$: There is no difference in the modifiability, irrespective of whether the class diagram has high or low size metrics. $(H2_A)$: the lower the structural complexity metrics of the class diagram, the higher the modifiability. $(H2_O)$: There is no difference in the modifiability, irrespective of whether the class diagram has high or low complexity metrics. $(H3_A)$: the lower the coupling metrics of the class diagram, the higher the modifiability. $(H3_O)$: There is no difference in the modifiability, irrespective of whether the class diagram has high or low coupling metrics. $(H4_A)$: the lower the reusability metrics of the class diagram, the lower the modifiability. $(H4_O)$: There is no difference in the modifiability, irrespective of whether the class diagram has high or low reusability metrics.

Data Collection

In this case study an IBM tool: Rational XDE is used to reverse engineer these web applications. Rational XDE combines the visual modeling with a Java forward and reverse engineering tool. The metrics are measured directly from generated class diagrams. This can become cumbersome if the application is large since a tool was not available to calculate these metrics.

In this study several attributes of web applications that are expected to affect maintainability are considered. These attributes include size, complexity, coupling, and reusability. Table 6.13 gives a description of the metrics that were used in the case studies. This study provided two levels of modifiability measures: low and high. The team lead of each web application was asked to provide a measure for the modifiability based on these two levels.

Results and Analysis

Metric Type	Metric Name	WebApp1	WebApp2
Size	(NServerP)	35	9
Size	(NClientP)	37	74
Size	(NWebP)	72	83
Size	(NFormP)	11	10
Size	(NFormE)	20	117
Size	(NClientC)	22	158
Structural Complexity	(NLinkR)	44	468
Structural Complexity	(NSubmitR)	8	9
Structural Complexity	(NBuildsR)	35	9
Structural Complexity	(NForwardR)	0	0
Structural Complexity	(NIncludeR)	39	0
Structural Complexity	(NUseTagR)	71	0
Control Coupling	(WebControlCoupling)	2.73	5.85
Data Coupling	(WebDataCoupling)	0.57	13
Reusability	(WebReusability)	0.54	0
Maintainability Measurement	Modifiability	high	low

Table 6.14: Results

The results are recorded in Table 6.14. The analysis is as follows:

Looking at the size attribute for WebApp1 one may notice that this application has more server side pages than WebApp2. As a conclusion WebApp2 needs developers with server side development experience, while WebApp1 needs developers with client side development experience. When comparing the structural complexity of WebApp1 to WebApp2, one can notice that WebApp2 has ten times more link relationships than WebApp1 even though the number of web pages for both application is almost the same. WebApp1 and WebApp2 have similar number of submit relationships, but WebApp1 has more form elements per form page, which means it will pass more data for each submit relationship. WebApp1 uses quite a few tags and has many include relationships. On the other hand WebApp2 does not use tags and has no include relationships. While looking at the control coupling for both web applications, one can see that WebApp1 and WebApp2 both have high control coupling due to the number of control relationships in both applications. The data coupling for WebApp2 is very high due to the high number

of data elements that are passed between components in the web application. If one looks at reusability one can notice that WebApp1 has good reusability due to the number of include relationships. On the other hand WebApp2 does not demonstrate good reusability that makes maintenance more difficult and makes WebApp2 more prone to error propagation.

WebApp2 has higher size metrics, structural complexity metrics, coupling metrics and lower reusability metrics than WebApp1. WebApp2 is expected to have lower modifiability. According to the results in Table 6.14 the modifiability for WebApp2 is low, while the modifiability for WebApp1 is high. Thus we can accept the alternative hypotheses $H1_A$, $H2_A$, $H3_A$, and $H4_A$, and reject the null hypotheses $H1_O$, $H2_O$, $H3_O$, and $H4_O$.

As a conclusion, this study showed that we were able to apply our metrics successfully to both industrial web applications and got an indication on the usefulness of using the metrics for improving maintainability in industrial web applications. This study can serve as a basis for future studies, and can provide a first indication of the use of the newly introduced metrics.

Threats to Validity

It is important to look at the internal and external validity of this study. In terms of internal validity, firstly, there was no automated tool for collecting the metrics from the design artifacts. There can be some human error in the process of computing the metrics from the class diagrams. Secondly, there was no configuration management tool for WebApp2. As a result some of the components might be outdated and not representative of the actual application. Also the maintainability is measured in a subjective manner which is less accurate than objective measures. With regard to external validity, one can see that the results can be generalized to other settings. The web applications used are from the telecommunication domain, but they are still using technologies that are similar to other applications in the market. The design of the application was collected using Rational XDE, which requires a license and might not be available for everyone to conduct a similar study, but still the outcome design is based on UML and there are

many freeware tools in the market that can be used to generate the design.

6.3 Industrial Provisioning Web Application

Case Study Context

The web application used is from the telecommunication Operational Support System (OSS) domain. It is a provisioning application which is used to provision and activate the wireless service in the network. We refer to the web application as ProvisionWebApp. ProvisionWebApp has around 10,000 users of which 2,000 are concurrent. It is a critical application that is used by customer care advocates to resolve provisioning issues for wireless subscribers. The ProvisionWebApp is divided into the following functional modules: *Login Module, Search Module, Current Transactions Module, Service Transaction Module, Device Transaction Module , UserName Module, Retrigger IOTA Module, Error Queue Module, Password Module, Network Provisioning Status Module, and Help Module*. ProvisionWebApp is built using the latest web technologies and frameworks such as Struts, and EJBs, and uses Oracle for the database. The web application uses Java as its main language. It has a Concurrent Versions System (CVS) repository for storing code changes. The data used in this case study is from year 2002 to year 2005.

This study is trying to explore the relationship between the following metric set (NServerP, NClientP, NWebP, NFormP, NFormE, NLinkR, NSubmitR, NbuildsR, NForwardR, NIncludeR, NClientScriptsComp, NServerScriptsComp, WebControlCoupling, NC, NA, NM, NAssoc, NAgg, CoupEntropy, CohesionEntropy) and maintenance effort measured by the number of lines of code changed and the number of revisions for components in a class diagram. In addition to that we would like to get an idea of how accurately our UML design metrics predict maintenance effort. CoupEntropy, and CohesionEntropy are described in the independent variables section while the rest of the metrics are described in Table 6.13.

Dependent Variables

The main goal of this study is to empirically explore the relationship between UML design metrics and maintenance effort. In this research two dependent variables are used to measure maintenance effort namely:

- LOC: The Absolute net value of the total number of lines added and deleted for components in a class diagram.
- nRev: Total number of revisions for components in a class diagram

The dependent variables are collected from a Concurrent Version System (CVS) repository.

Independent Variables

In this research the metrics based on the web application reference model shown in Figure 2.6 are used as independent variables. The following metrics (NServerP, NClientP, NWebP, NFormP, NFormE, NLinkR, NSubmitR, NbuildsR, NForwardR, NIncludeR, NClientScriptsComp, NServerScriptsComp, WebControlCoupling, WebDataCoupling, WebReusability) were defined in the authors previous study [48]. The following (NC, NA, NM, NAssoc, NAgg) metrics were defined in the study carried by Genero [44] on class diagram metrics for object oriented applications. The metrics use the different components of the web application reference model as units of measurement. In addition to the above mentioned metrics this study also uses the following two metrics: *CoupEntropy* and *CohesionEntropy*. They were first presented in [3], but we have modified them a little bit to fit in the context of UML class diagram metrics. A description of each of the metrics investigated is given as follows:

- CoupEntropy: The *CoupEntropy* is computed as shown in the following equation: $1/n \times (-\log 1/(1+m))$ where n is the total number of elements in the class diagram and m is the total number of relationships in the class diagram. The total number of elements in the class diagram is the sum of all server pages, client

pages, form pages, and interface classes. The total number of relationships is the sum of all *builds*, *links*, *submit*, *includes*, *forwards*, *NAssoc*, and *NAgg* relationships.

- CohesionEntropy: The *CohesionEntropy* of a class diagram is equal to total *CoupEntropy* of all class diagrams over the *CoupEntropy* of one class diagram. The *CohesionEntropy* is computed as shown in the following equation: $\frac{\sum_{i=1}^{k}(1/n\times(-\log 1/(1+m)))}{(1/n\times(-\log 1/(1+m)))}$ where n is the total number of elements in the class diagram, m is the total number of relationships in the class diagram, and k is the total number of class diagrams in the application.

Hypothesis

We will show both the null (H_O) and the alternate hypotheses (H_A). The following hypotheses are investigated:

- $H1_A$: There is a nonzero regression relationship between the size metrics of the class diagram and LOC.
- $H1_O$: There is no regression relationship between the size metrics of the class diagram and LOC.
- $H2_A$: There is a nonzero regression relationship between the size metrics of the class diagram and nRev.
- $H2_O$: There is no regression relationship between the size metrics of the class diagram and nRev.
- $H3_A$: There is a nonzero regression relationship between the complexity metrics of the class diagram and LOC.
- $H3_O$: There is no regression relationship between the complexity metrics of the class diagram and LOC.
- $H4_A$: There is a nonzero regression relationship between the complexity metrics of the class diagram and nRev.

- $H4_O$: There is no regression relationship between the complexity metrics of the class diagram and nRev.
- $H5_A$: There is a nonzero regression relationship between the coupling metrics of the class diagram and LOC.
- $H5_O$: There is no regression relationship between the coupling metrics of the class diagram and LOC.
- $H6_A$: There is a nonzero regression relationship between the coupling metrics of the class diagram and nRev.
- $H6_O$: There is no regression relationship between the coupling metrics of the class diagram and nRev.
- $H7_A$: There is a nonzero regression relationship between the cohesion metrics of the class diagram and LOC.
- $H7_O$: There is no regression relationship between the cohesion metrics of the class diagram and LOC.
- $H8_A$: There is a nonzero regression relationship between the cohesion metrics of the class diagram and nRev.
- $H8_O$: There is no regression relationship between the cohesion metrics of the class diagram and nRev.

Data Collection

In this study an automated tool named WapMetrics is used for the data collection. WapMetrics is described in Appendix C. It is a web tool that takes UML diagrams in XMI [83] format as input and produces the results in different output formats. WapMetrics is used to compute the UML metrics from the class diagrams and provide the results in excel format in order to be used in the statistical analysis phase.

Unfortunately, the class diagrams were out of sync for the ProvisionWebApp application. IBM Rational Rose Enterprise Edition [53] was used for reverse engineering

the ProvisionWebApp application. Rational Rose has a visual modeling component. It can create the design artifacts of a software system. The Web Modeler component in Rational Rose supports Conallen's extension for web applications. The Web Modeler component was used to generate the class diagrams for the various components of the ProvisonWebApp application.

For some of the class diagrams, we had to hide the visibility of attributes and methods in classes in order to fit them in one page. This did not affect the computation of the metrics since all attribute and methods are exported in the XMI file. Some of the relationships were not in the generated class diagrams. For example the *include* and *forward* relationships shown in our reference model in Figure 2.6 were not generated by Rational tool. We had to add these relationships manually to the class diagrams. After the class diagrams were validated, Unisys Rose XML [53] was used to export the UML class diagrams into XML Metadata Interchange (XMI) [83]. The WapMetrics tool was used to compute the independent variables from the XMI input file. The dependent variables are collected from the CVS system. LOC is computed by adding the absolute value of Lines of Code for all classes in a class diagram. nRev is computed by adding all revisions for all classes in a class diagram.

Analysis Results

Table 6.15: Descriptive Statistics

Variable	N	Min	Max	Mean	Std Deviation
LOC	30	13	13179	2780.9	3725.44
nRev	30	2	229	74.1	78.03
NFormP	30	0	2	.47	.681
NClientScriptsComp	30	0	4	.63	1.098
NServerScriptsComp	30	0	10	4	3.029
NC	30	0	12	3.27	3.704
NA	30	0	218	29.63	54.097
NM	30	0	472	62.47	113.99
NAssoc	30	0	14	3.2	3.745
NAgg	30	0	2	.70	.837
NBuildsR	30	0	14	4.73	3.581
CoupEntropy	30	0	.00644	.0041	.00145
CohesionEntropy	30	1	82.43	31.94	16.41

Descriptive Statistics

Table 6.15 shows the descriptive statistics of the independent and dependent variables used in this study. The measures for LOC dependent variable are much higher than the measures for nRev. The maximum for LOC is 13179 lines of code added or deleted while the maximum for nRev is 229 revisions. This result is expected since each revision will have more than one line of code associated with it. For the independent variables we can see that NM has the highest value with 472 methods. For the minimum value we can see that several variables have 0 measures. This is understandable since some class diagrams in our application did only have client pages or server pages, which resulted in having 0 measures for several independent variables.

Univariate Negative Binomial Regression

Table 6.16: Univariate Analysis Results

Metric Type	Metric Name	LOC Coef/StdErr/Sig	nRev Coef/StdErr/Sig
Size	NFormP	.639/.286/.025	.580/.283/.040
Size	NClientScriptsComp	.623/.183/.001	.610/.197/.002
Size	NServerScriptsComp	.183/.081/.023	.187/.080/.019
Size	NC	.468/.058/.000	.376/.059/.000
Size	NA	.017/.007/.017	.016/.006/.012
Size	NM	.008/.003/.017	.008/.003/.012
Structural Complexity	NAssoc	.487/.062/.000	.387/.063/.000
Structural Complexity	NAgg	.650/.205/.002	.672/.213/.002
Structural Complexity	NBuildsR	.201/.066/.003	.210/.072/.004
Coupling	CoupEntropy	1013.9/112.8/.000	970.9/141.8/.000
Cohesion	CohesionEntropy	-.058/.009/.000	-.078/.013/.000

The main goal of this case study is to investigate the feasibility of using the metrics described in Table 6.13 as predictors of maintenance effort. We build 2 models based on the LOC and nRev dependent variables separately. Both variables are discrete count variables that are highly skewed and always positive. Modeling using ordinary least squares regression (OLS)leads to highly non-normal error distributions leading to invalid final models. In order to cope with variables of this type Generalized Linear Models have been devised. These models include Poisson and Negative Binomial Re-

gression. Negative Binomial Regression for log link function was chosen for this data as it copes with the overdispersion ($variance > mean$) found in a Poisson model [36].

We started by looking at the individual relationships between all the metrics defined in Table 6.13 and the dependent variables: LOC and nRev.

Table 6.16 shows the results from applying univariate negative binomial regression to the data set. "Coeff" indicates the coefficient in the regression equation, "StdErr" its standard error and "Sig" its significance or p-value, that is the probability the coefficient is greater than zero by chance.

The following size metrics (NFormP, NClientScriptsComp, NServerScriptsComp, NC, NA, NM) showed significance ($p < 0.05$) with LOC and nRev. For the structural complexity metrics only (NAssoc, NAgg, NbuildR) showed significance with LOC and nRev. Both CoupEntropy, and CohesionEntropy showed significance with LOC and nRev.

Multivariate Negative Binomial Regression

Having examined the relationship of individual metrics (the predictors) and the dependent variables, LOC and nRev, we can now examine the combined effect of metrics on the dependent variables by performing a multivariate analysis.

The selection of the predictors can be made using two different stepwise regression techniques: the forward selection method, and the backward elimination method. The forward method starts with a model that only includes a constant and then adds single predictors based on a specific statistical criteria. Forward selection regression is used when there is no previous research telling us what to expect from the results. The backward method starts with a model that includes all predictors, which are deleted one at a time from the model based on a specific statistical criteria until an optimal model is found. In this study, we use $\alpha > 0.05$ for excluding the predictors from the model and, the backward elimination method with Negative Binomial distribution with a log link function for building the model. The likelihood-ratio chi-square test is used to compare the current model versus the intercept model. A significance value of less than 0.05 indicates that the current models outperforms the intercept model. All models have a

value of less than 0.05 which means they outperformed the intercept only model. The following is the discussion of the analysis results:

- **Size Metric Model:** The Size Metric Model predicting LOC from NClientScriptsComp, NServerScriptsComp, and NC is statistically significant with $\chi^2(3) = 67.1$, $p < 0.05$ The predictors NClientScriptsComp, NServerScriptsComp, and NC were each statically significant. The Size Metric Model predicting nRev from NC is statistically significant with $\chi^2(1) = 38.7, p < 0.05$. We can accept the alternative hypothesis $H1_A$ and $H2_A$, and reject the null hypothesis $H1_O$ and $H2_O$.

  The predictor NC was statically significant. Table 6.17 shows the coefficients, standard error and significance for all the independent variables in the size metric models. The negative coefficient for NServerScriptsComp is counterintuitive, since we expect the Lines of Code to increase as we have more server script components. The reason for the negative number can be explained by the suppressor relationship between NServerScriptsComp and NClientScriptsComp which is common between correlated variables [9]. This is not of a concern as long as no strong multicollinearity [41] exists which was determined to be negligible since the condition number was equal to 3.87. A condition number of more than 30 indicates that strong multicollinearity exists between variables [41].

Table 6.17: Size Metrics Model

Parameter	**Coeff**	**Std. Error**	**Sig**
LOC Model			
Intercept	5.712	.304	.000
NClientScriptsComp	.556	.214	.009
NServerScriptsComp	-.212	.069	.002
NC	.488	.071	.000
nRev Model			
Intercept	2.415	.275	.000
NC	.376	.059	.000

- **Complexity Metric Model:** The Complexity Metric Model predicting LOC from NAssoc, is statistically significant with $\chi^2(1) = 53.4, p < 0.05$

Table 6.18: Complexity Metrics Model

Parameter	Coeff	Std. Error	Sig
LOC Model			
Intercept	5.482	.270	.000
NAssoc	.487	.062	.000
nRev Model			
Intercept	2.426	.279	.000
NAssoc	.387	.063	.000

The predictor NAssoc was statically significant. The Complexity Metric Model predicting nRev from NAssoc is statistically significant with $\chi^1(3) = 37.6, p < 0.05$. We can accept the alternative hypothesis $H3_A$ and $H4_A$, and reject the null hypothesis $H3_O$ and $H4_O$.

The predictor NAssoc was statically significant. Table 6.18 shows the coefficients, standard error and significance for all the independent variables in the complexity metric models.

- **Coupling Metric Model:** The Coupling Metric Model predicting LOC from CoupEntropy is statistically significant with $\chi^2(1) = 31.2, p < 0.05$. The predictor CoupEntropy was statically significant. The Coupling Metric Model predicting nRev is statistically significant with likelihood ratio $\chi^2(1) = 30.1, p < 0.05$. We can accept the alternative hypothesis $H5_A$ and $H6_A$, and reject the null hypothesis $H5_O$ and $H6_O$. The predictor CoupEntropy was statically significant. Table 6.19 shows the coefficients, standard error and significance for all the independent variables in the coupling metric models.

- **Cohesion Metric Model:** The Cohesion Metric Model predicting CohesionEntropy is statistically significant with $\chi^2(1) = 7.2, p < 0.05$. The predictor CohesionEntropy was statically significant. The Cohesion Metric Model predicting nRev from CohesionEntropy is statistically significant with $\chi^2(1) = 10.3, p < 0.05$. We can accept the alternative hypothesis $H7_A$ and $H8_A$, and reject the null hypothesis $H7_O$ and $H8_O$.

  The predictor CohesionEntropy was statically significant. Table 6.19 shows the

Table 6.19: Coupling and Cohesion Metrics Model

Parameter	**Coeff**	**Std. Error**	**Sig**
LOC Coupling Model			
Intercept	4.048	.501	.000
CoupEntropy	691.8	95.9	.000
nRev Coupling Model			
CoupEntropy	699.3	111.0	.000
LOC Cohesion Model			
Intercept	8.933	.354	.000
CohesionEntropy	-.038	.010	.000
nRev Cohesion Model			
Intercept	5.743	.431	.000
CohesionEntropy	-.055	.014	.000

coefficients, standard error and significance for all the independent variables in the cohesion metric models.

- **All Metric Model:** The All Metric Model predicting LOC from NAssoc, NClientScriptsComp, NServerScriptsComp and CoupEntropy is statistically significant with $\chi^2(4) = 71.1, p < 0.05$. The predictors NAssoc, NClientScriptsComp, NServerScriptsComp and CoupEntropy were each statically significant. The reason for the negative number for NServerScriptsComp can be explained by the suppressor relationship between NServerScriptsComp and CoupEntropy which is common between correlated variables. This is not of a concern as long as no strong multicollinearity exists which was determined to be negligible since the condition number was equal to 11.95.

  The All Metric Model predicting nRev from NC and CoupEntropy is statistically significant with $\chi^2(2) = 43.3, p < 0.05$. The predictors NC and CoupEntropy were each statically significant. Table 6.20 shows the coefficients, standard error and significance for the independent variables in the all metric models.

Table 6.20: All Metrics Model

Parameter	Coeff	Std. Error	Sig
LOC Model			
Intercept	4.506	.639	.000
NAssoc	.439	.079	.000
NClientScriptsComp	.568	.212	.007
NServerScriptsComp	-.338	.087	.000
CoupEntropy	375.7	187.8	.045
nRev Model			
NC	.284	.075	.000
CoupEntropy	335.2	148.6	.024

Model Validation

In this study we use the Magnitude of Relative Error (MRE) [36] for evaluating the prediction models. The MRE is shown in Equation 6.1:

$$MRE = |\frac{x - y}{y}| \tag{6.1}$$

where x is the predicted value and y is the actual value. The result can be multiplied by 100 to get the percentage of deviation from the actual value. The $MMRE$ is the mean of the MRE, it is one of the most widely used criterion for assessing the performance of software prediction models [80, 61].

Table 6.21 shows the values of MRE values in the data set. It shows the mean, standard deviation, 25th percentile(P25), median, and 75th percentile (P75). When checking the mean MRE for LOC models we can see that the mean MRE for size, structural complexity, coupling and all metrics models ranges between .17 to .51. The best mean MRE value was for the All metric model (.1795) while the worst mean MRE value was for the cohesion model (5178).

When looking at the results for the nRev models, we can see that the mean MRE values ranges between .32 to .88. The best mean MRE value was for the All metric model (.3222) while the worst mean MRE value was for the cohesion model (.8848).

It is important to have confidence in our results. Bootstrapping is one technique that is used to obtain confidence intervals for small data sets [18]. In this study we would like to find 95 percent confidence intervals for our prediction models. We follow the

following bootstrapping procedure [57]:

Table 6.21: Goodness of Fit: Values of MREs for All Models

	Size	**Complexity**	**Coupling**	**Cohesion**	**All Metrics**
LOC Model					
Mean	.1998	.2646	.3972	.5178	.1795
StdDev	.22127	.26368	.48582	.57416	.20045
P25	.0695	.0785	.0587	.0717	.0567
Median	.1259	.1665	.1385	.1560	.1167
P75	.2624	.4693	.5395	.9762	.2580
Rev Model					
Mean	.4086	.4160	.4430	.8848	.3222
StdDev	.73204	.74078	.55033	1.58322	.25940
P25	.0615	.0523	.1488	.1208	.1688
Median	.1580	.1597	.2391	.1795	.2642
P75	.2770	.3043	.6166	.9346	.3951

1. Sample 1000 times and replace randomly our 30 MRE values to obtain 1000 samples of 30 observations.

2. For each sample compute the mean MRE values for each of the models.

3. Compute the 2.5 percent and the 97.5 percent percentiles which is considered an estimate of the 95 percent confidence interval of the mean MRE values.

Table 6.22: Goodness of Fit: Values of MEAN MREs for All Models using Boostrapping

	Size	**Complexity**	**Coupling**	**Cohesion**	**All Metrics**
LOC Model					
P2.5	.24	.17	.23	.31	.11
P97.5	.59	.37	.58	.71	.25
nRev Model					
P2.5	.27	.18	.27	.40	.23
P97.5	.65	.70	.65	1.44	.41

Table 6.22 shows the results of the bootstrapping procedure described above. One can see that the best results are for the LOC complexity and LOC All metric models.

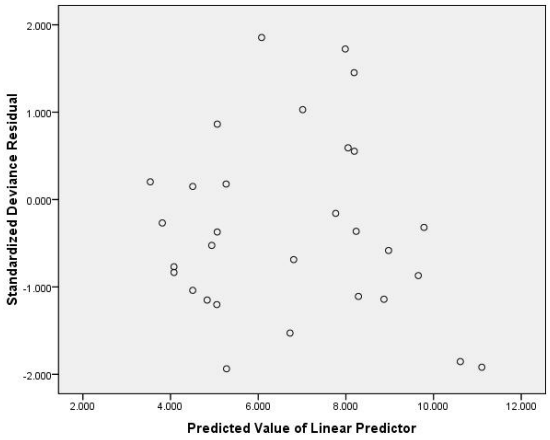


Figure 6.3: ScatterPlot LOC Model

The mean MRE for the LOC complexity model is between .17 and .37. This means that we can have 96 percent confidence that the mean MRE for the complexity model will be between .17 and .37. The mean MRE for the LOC ALL metric model has even better results (.11 to.25). For the nRev model the best results are for the nRev ALL metric model (.23 to .41).

We used the likelihood-ratio chi-square test to compare current model versus the intercept only model. All models showed a significance value of less than 0.05 which indicates that the current models outperforms the intercept only model. We also have searched for the influential points and outliers in the models. We draw charts of standardized deviance residual versus and predicted values of the linear predictor variable. Figure 6.3 shows the this scatterplot for the LOC ALL Metrics Model. The resulting scatterplot appears to not have any outlying points. Similarly, we drew the scatterplots for the Size, Structural Complexity, Coupling, and Cohesion Metric Models, and we got similar results with no outlying points.

Threats to Validity

It is important to look into threats to validity in order to make sure the results are valid. We will look into three types of threats that can limit us to draw conclusions from the

results: Construct Validity, Internal Validity, and External Validity.

Construct Validity

Construct Validity is the degree to which the independent variables and the dependent variables are accurately measured in the study. The dependent variables in this study are LOC and nRev. Both of these measures were measured from a CVS repository. The CVS repository has an accurate value for both of these measures. However, human error can happen in computing and recording both dependent variables, therefore we have repeated the measure for both variables a second time and made sure that the results from the first and second time match.

Another issue is that the measurement of the independent variables was performed from source code since no complete design was available from which the measures could be obtained. In practice the measures for the independent variables should be taken from early UML design diagrams. Measures from source code are more accurate but an investigation on how these measures compare to measures taken from design diagrams, and how this can affect the accuracy of the prediction model must be carried out.

Internal Validity

Internal Validity is the degree to which conclusions can be drawn about the effect of the independent variables on the dependent variables. In this study we have demonstrated that some of the metrics have a statistically and significant relationship with LOC and nRev. This relationship does not prove a causal relationship, it only provides evidence that such a relationship might exist. The only way to prove causality is to run controlled experiments where the independent variables are varied in a controlled manner while preserving the functionality and size of the application. In practice this is difficult to accomplish.

External Validity

External Validity is the degree to which results can be generalized to other research settings. We have used a real industrial application with two years of data stored in a CVS repository. In addition we have used bootstrapping to have confidence in the results for the mean MRE values. However, other factors such as developer experience, size of application and technologies used can limit the generalization of the results to other web applications.

Conclusions

Early measures of software maintainability can help in allocating project resources efficiently, predicting the effort of maintenance tasks and controlling the maintenance process. In this study we explore the relationship between UML class design metrics and maintenance effort which is measured by the number of lines of code changed and by the number of revisions. The results showed that there is a reasonable chance that useful prediction models can be built from early UML design metrics. We have obtained good results using bootstrapping, for the LOC ALL metric model the mean MRE lies between 11 to 25 percent for 95 percent of the cases. For the nRev ALL metric model we also got good results, the mean MRE lies between 23 to 41 percent for 95 percent of the cases.

We studied the following metric models and got the following results:

- LOC Size metric model: This model shows the relationship between our size metrics and LOC(Lines of Code), we found that the following size metrics NClientScriptsComp, NServerScriptsComp, and NC explained the effort measured by LOC(Lines of Code).

- nRev Size metric model: This model shows the relationship between our size metrics and nRev(Number of Revisions), we found that only NC metric explained the effort measured by nRev(Number of Revisions).

- LOC Complexity metric model: This model shows the relationship between our

complexity metrics and LOC(Lines of Code), we found that the following complexity metric NAssoc explained the effort measured by LOC(Lines of Code).

- nRev Complexity metric model: This model shows the relationship between our size metrics and nRev(Number of Revisions), we found that NAssoc metric explained the effort measured by nRev(Number of Revisions).

- LOC Coupling metric model: This model shows the relationship between our coupling metrics and LOC(Lines of Code), we found that the following coupling metric CoupEntropy explained the effort measured by LOC(Lines of Code).

- nRev Coupling metric model: This model shows the relationship between our coupling metrics and nRev(Number of Revisions), we found that CoupEntropy metric explained the effort measured by nRev(Number of Revisions).

- LOC Cohesion metric model: This model shows the relationship between our cohesion metrics and LOC(Lines of Code), we found that CohesionEntropy explained the effort measured by LOC(Lines of Code).

- nRev Cohesion metric model: This model shows the relationship between our cohesion metrics and nRev(Number of Revisions), we found that CohesionEntropy metric explained the effort measured by nRev(Number of Revisions).

- LOC All metric model: This model shows the relationship between all our metrics and LOC(Lines of Code), we found that the following metrics NAssoc, NClientScriptsComp, NServerScriptsComp, and CoupEntropy explained the effort measured by LOC(Lines of Code).

- nRev ALL metric model: This model shows the relationship between all our metrics and nRev(Number of Revisions), we found that NC, and CoupEntropy metrics explained the effort measured by nRev(Number of Revisions).

7

Conclusion and Future Work

7.1 Conclusion

There are several design quality attributes that have an effect on the maintainability of software artifacts such as size, complexity, coupling, cohesion, and reusability. In this Ph.D thesis, we have defined new UML design metrics based on the Web Application Extension (WAE) [26] for UML. The metrics use the different components of the web application reference model as units of measurement. We have defined new metrics based on the following relations between different components in the web application reference model: *builds*, *links*, *submit*, *includes*, and *forwards*. In addition to that, we have defined our new UML metrics using the following components in the web application reference model: client Page, Server Page, Forms, Components, Scriplets and Interface Objects. The new UML metrics use the relationships and components of the web application reference model to measure attributes of class diagrams such as size, complexity, coupling, cohesion, and reusability. Our motivation in this research was to use these metrics to support the maintenance of web applications, and to show that the UML metrics can be useful in controlling the maintenance of web applications and can provide predictions to different measures of maintainability.

In this thesis, theoretical validation of our UML design metrics has been accomplished by defining our metrics using two validation frameworks, one proposed by Kitchenham [63] and one proposed by Briand [16].

For Kitchenham's framework we have defined our metrics using the structure model proposed by Kitchenham [63]. We have provided a classification of the metrics based on the measurement structure proposed by Kitchenham [63] and the metrics were validated based on the following conditions: attribute Validity, unit Validity, instrument Validity, and protocol Validity.

For Briand's framework we have defined our metrics using the general framework proposed by Briand *et al* [16]. We have defined our metrics using the properties defined by the framework for size, complexity, cohesion, and coupling. The metrics were defined using precise mathematical concepts where the class diagram was represented as a system which consists of a set of elements and a set of relationships between them.

The empirical validation of our UML metrics has been accomplished by conducting several empirical studies using an open source web application and industrial web applications from the telecommunication domain. We have used the pet store web application which is an open source web application available at the sun web site. The pet store application is an example of a typical e-commerce web application that provides customers with online shopping. We have used an industrial application from the telecommunication domain for our experiments. The industrial application is a provisioning application which is used to provision and activate the wireless service in the network. It has around 10,000 users of which 2,500 are concurrent. It is a critical application that is used by customer care advocates to resolve provisioning issues for wireless subscribers.

In this research we have built an open source tool called WapMetrics for measuring UML design metrics for web applications. WapMetrics provides an automated way to measure UML metrics and has the ability to show the results in different output formats. We have shown how WapMetrics can take UML diagrams in XMI [83] format as input and produce results in HTML format. The WapMetrics tool takes a standard input and can be used for any UML diagram in XMI [83] format. WapMetrics can measure and calculate web application metrics from UML diagrams based on the Conallen model. Most other tools concentrate on UML metrics for object-oriented applications. In addition to that, it is independent from the CASE tool used to build the models, and takes

an XMI file as input. The XMI file describes the UML model in a standard way. The XMI input allows the exchange of model information in a standard way regardless of the CASE tool used to create the XMI file. WapMetrics is a web application that can be deployed on a central server and used by many users without installing it on the client machines. This makes it easy to maintain and deploy enhancements to the WapMetrics tool. WapMetrics provides interoperability and, the outcome of WapMetrics is user friendly and easy usable by other tools. WapMetrics allows the output to be exported in several formats: HTML, XML, pdf, excel, rtf and csv. This allows the output to be used for statistical reporting and the results to be presented in graphs and other formats.

We studied the relationship between our UML metrics and the following maintainability measures: Understandability Time (the time spent on understanding the software artifact in order to complete the questionnaire), and Modifiability Time(the time spent on identifying places for modification and making those modifications on the software artifact). The study gave an indication that there is a possibility for a relationship to exist between our metrics and modifiability time. However, we did not find a relationship between our metrics and understandability time. The results did not show statistical significance on the effect of the metrics on understandability time. However the study was based on students and this may limit how much the results can be generalized to professional developers. Another limitation is the web application used in this experiment. The pet store is a toy application compared to complex web applications. We also studied the relationship between our metrics and LOC (total number of Lines of Code added and deleted for components in a class diagram), and nRev (total number of revisions for components in a class diagram). The results showed that there is a reasonable chance that useful prediction models can be built from early UML design metrics. We have obtained good results using bootstrapping, for the LOC ALL metric model the mean MRE lies between 11 to 25 percent for 95 percent of the cases. For the nRev ALL metric model we also got good results, the mean MRE lies between 23 to 41 percent for 95 percent of the cases. The study had several strengths: it used an industrial web application, used real data from year 2002 to year 2005, and we used regression analysis and bootstrapping for validating the metrics. Our results give a first indication of the

usefulness of the UML design metrics, they show that there is a reasonable chance that useful prediction models can be built from early UML design metrics.

We have provided a set of experiments for web applications that can be generalized to other settings. These experiments build on previous research and provide a starting point for further research on web applications maintainability.

Finally, we have defined an environment for the maintainability prediction model that includes tools and procedures so that it can be used in an industrial environment. We have described the complete process of computing our UML metrics from class diagrams. We have discussed web applications that do not have preexisting class diagrams and described how to reverse engineer these web applications to come up with the class diagrams.

Benefits of Research to Project Managers

This research can be used by project managers and team leads to identify the design components that need more time and resources. This will help them allocate resources efficiently. For example in a project that has just started, the designers will create the UML class diagrams for the web application based on the web application reference model defined in chapter 4. After creating the class diagrams, the developers will create an XMI [83] file representing those class diagrams. Most design tools can export class diagrams to XMI [83] which is a common language for representing design documents in XML. The WapMetrics tool (described in Appendix C) takes the XMI file as input and produces an HTML output showing the results of computing the metrics from the different class diagrams. These results can be exported to several formats such as pdf and excel. Based on the metrics of each class diagram the project manager can decide which class diagrams have a possibility of taking more time and can allocate resources accordingly.

Another example would be in a project that has already started but no design documents exist. In this case we have to reverse engineer the code to create the UML class diagrams. After creating the UML class diagrams and validating them by experienced developers we can use the WapMetrics tool to compute our metrics.

Issues and Problems

One of the main issues and problems we faced in this research was finding subjects for our experiments. It was hard to convince instructors to allow us to conduct the experiments in their classes. We succeeded though to convince one instructor at London South Bank University to conduct one experiment for one of his classes. The class had six students and they were all senior level. We chose an open source web application that was implementing a PetStore e-commerce site. The application was using some new server side technologies. We created all the design diagrams for the application after reverse engineering the code. Then, we created the questionnaire which was based on the design diagrams. Unfortunately we did not get a good participation in that experiment since only one student answered the questions. It was obvious that the questions were difficult for the students also the data was not useful for statistical analysis. We decided to include a brief questionnaire about the subjects knowledge on web applications design and technologies before conducting any future experiments. We tried to contact several other universities in the UK but with no luck. We were not able to get positive response from instructors to conduct our experiments. We decided to try some universities in the US and we got a positive response from an instructor at Illinois State University. Finally we were able to conduct some of our experiments there.

There is some debate on the usefulness of conducting experiments with students. Therefore, we decided to convince experienced programmers to participate in our experiments. We had an opportunity to talk to programmers during the Software Craftsmanship 2009 conference which was held in London. We started preparing for the experiment one month before the conference. The idea was to find an open source web application and to create the design diagrams by reverse engineering the code of that application. We were already working on an application named Claros and had some of the design diagrams created. We created the rest of the diagrams and created a survey questionnaire based on the design diagrams. During the conference we talked to several programmers and handed out flyers about the experiment. Unfortunately we got only one participant in this experiment, which was not enough to conduct statistical analysis.

As a conclusion it is hard to convince experienced programmers to participate in these experiments without some incentives, maybe there should be some money incentive or some type of recognition for the programmers that participate. This has to be studied carefully in order not to bias the results of the experiments.

7.1.1 Original Contribution to Knowledge

The contribution of this research can be summarized by the following points. It provides:

1. a study on the relationship between the UML design metrics and Understandability Time and Modifiability Time (the time spent on understanding and modifying a software artifact).

2. a study on the relationship between the UML design metrics and LOC (Lines of Code Changed). The research tells if by using the UML design metrics is it possible to predict the LOC of the class diagrams.

3. a study on the relationship between the UML design metrics and nRev (Number of Revisions). The research tells if by using the UML design metrics is it possible to predict the nRev for classes in a class diagram.

4. a validation of UML design metrics for web applications. This validation was accomplished using both theoretical and empirical validation.

5. a set of empirical experiments for web applications that can be generalized to other settings. These experiments build on previous research and provide a starting point for further research on web applications maintainability.

6. an environment for the maintainability prediction model that includes tools and procedures so that it can be used in an industrial environment.

7. an extension of Conallen's model to present web applications in more detail.

7.1.2 Future Work

For our UML metrics to be used as a standard for measuring the maintainability of web applications more experiments are to be conducted. It is important that more experiments are conducted using different web applications from different domains so that our results can be generalized to other research settings.

In our future work, we hope to explore the relationship between our UML design metrics and fault-proneness. Many studies [50, 21, 17, 7] have researched the relationship between fault-proneness and different quality metrics. There are some studies that investigated the relationship between metrics and fault-proneness of classes [50, 7]. We have not seen a study that investigated the relationship between quality metrics and fault-proneness in class diagrams. Fault-proneness in class diagrams is defined as the probability of detecting a fault in a UML class diagram. The faults can be collected from a bug tracking system and WapMetrics can be used to compute the UML metrics.

In the future, we hope to enhance our open source tool WapMetrics with new functionalities. Currently, WapMetrics does not allow the user to specify new metrics without re-programming the tool. We hope to add functions for defining UML metrics using XML files so that it can be easily enhanced with user defined metrics. In addition to that, WapMetrics can only measure metrics from class diagrams. We hope to add the support of measuring metrics from different design diagrams such as activity diagrams and use cases. One nice feature to add, is the measurement of differences between two versions of UML models.

Bibliography

[1] S. Abraho, N. Condori-Fernndez, L. Olsina, and O. Pastor. Defining and validating metrics for navigational models. In *Proceedings of the 9th International Software Metrics Symposium*, pages 200–210, Sydney, Australia, 2003. IEEE Computer Society Press.

[2] T. Al-Rousan, S. Sulaiman, Rosalina, and A. Salam. WPRiMA tool: Managing risks in web projects. *International Journal of Human and Social Science*, 5(11):686–693, 2010.

[3] E. Allen and T. Khoshgoftaar. Measuring coupling and cohesion: An information-theory approach. In *Proceeding of the 6th International Software Metrics Symposium*, pages 119–127, Boca Raton, FL , USA, 1999. IEEE Computer Society Press.

[4] M. Alshayeb and W. Li. An empirical validation of object-oriented metrics in two different iterative software processes. *IEEE Transactions on Software Engineering*, 29(11):1043–1049, 2003.

[5] L. Baresi, S. Morasca, and P. Paolini. Estimating the design effort of web applications. In *Proceedings of the 9th International Software Metrics Symposium*, pages 62–72, Sydney, Australia, 2003. IEEE Computer Society Press.

[6] R. Barga and D. Lomet. Recovery guarantees for Internet applications. *ACM Transactions on Internet Technology*, 4(3):289–328, 2004.

[7] V. Basili, L. Briand, and W. Melo. A validation of object-oriented design metrics

as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, 1996.

[8] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 2 edition, 2003.

[9] D. Belsley, E. Kuh, and R. Welsch. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. WileyBlackwell, 2004.

[10] P. Bhatt, G. Shroff, and A. Misra. Dynamics of software maintenance. *ACM SIGSOFT Software Engineering Notes*, 29(4):1–5, 2004.

[11] J. Bieman and B.-K. Kang. Cohesion and reuse in an object-oriented system. In *Proceedings of the ACM SIGSOFT Symposium on Software Reusability*, pages 259–262, Seattle, WA, USA, 1995. IEEE Computer Society Press.

[12] A. Bongio, S. Ceri, and P. Fraternali. Estimating the design effort of web applications. In *Proceedings of the 9th International Conference on the WWW*, pages 137–157, Amsterdam, Netherlands, 2000. Elsevier.

[13] L. Briand, C. Bunse, and J. Daly. A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. *IEEE Transactions on Software Engineering*, 27(06):513–530, 2001.

[14] L. Briand, Daly, J. Daly, and J. Wst. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering*, 25(1):91–121, 1999.

[15] L. Briand, J. Daly, and J. Wst. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering*, 3(1):65–117, 1998.

[16] L. Briand, S. Morasca, and V. Basili. Prperty-based software engineering measurement. *IEEE Transactions on Software Engineering*, 22(1):68–86, 1996.

[17] L. Briand, S. Morasca, and V. Basili. Defining and validating measures for object-based high-level design. *IEEE Transactions on Software Engineering*, 25(5):722–743, 1999.

[18] L. Briand and J. Wurst. Modeling development effort in object-oriented systems using design properties. *IEEE Transactions on Software Engineering*, 27(11):963–986, 2001.

[19] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley and Sons, 1 edition, 1996.

[20] D. Card, K. El-Emam, and B. Scalzo. Measurement of object-oriented software development projects. *Software Productivity Consortium NFP*, 2001.

[21] M. Cartwright and M. Shepperd. An empirical investigation of an object-oriented software system. *IEEE Transactions on Software Engineering*, 26(8):786–796, 2000.

[22] S. Chidamber and C. Kemerer. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[23] Claros. Claros in touch application. Website. `http://www.claros.org/web/home.do`.

[24] D. Coleman, D. Ash, B. Lowther, and P. Oman. Using metrics to evaluate software system maintainability. *IEEE Computer*, 27(8):44–49, 1994.

[25] S. Comai and P. Fraternali. A semantic model for specifying data-intensive web applications using WebML. In *Proceedings of the 1st International Semantic Web Working Symposium*, pages 566–585, Stanford, California, 2001. Stanford University.

[26] J. Conallen. *Building Web Applications with UML.* Addison-Wesley, 2 edition, 2003.

[27] T. Corbi. Program understanding: challenge for the 1990s. *IBM Systems Journal*, 28(2):294–306, 1989.

[28] A. DeLucia, R. Francese, G. Scanniello, and G. Tortora. Reengineering web applications based on cloned pattern analysis. In *Proceedings of the 12th IEEE International Workshop on Program Comprehension*, pages 132–141, Bari, Italy, 2004.

[29] S. Demeyer, S. Ducasse, and O. Nierstrasz. *Object-Oriented Reengineering Patterns*. Morgan Kaufmann, 1 edition, 2002.

[30] G. Di-Lucca, A. Fasolino, U. De-Carlini, and P. Tramontana. Ware: a tool for the reverse engineering of web applications. In *Proceeding of the 6th European Conference on Software Maintenance and Reengineering*, pages 241–250, Budapest, Hungary, 2002.

[31] G. Di-Lucca, A. Fasolino, and P. Tramontana. Reverse engineering web applications: the WARE approach. *Journal of Software Maintenance*, 16(2):71–101, 2004.

[32] G. DiLucca, A. Fasolino, P. Tramontana, and C. Visaggio. Towards the definition of a maintainability model for web applications. In *Proceeding of the 8th European Conference on Software Maintenance and Reengineering*, pages 279–287, Tampere, Finland, 2004. IEEE Computer Society Press.

[33] E. DiSciascio, F. Donini, M. Mongiello, and G. Piscitelli. Web applications design and maintenance using symbolic model checking. In *Proceedings of the 7th European Conference on Software Maintenance and Reengineering*, pages 63–72, Benevento, Italy, 2003. IEEE Computer Society Press.

[34] S. Ducasse, M. Rieger, and S. Demeyer. A language independent approach for detecting duplicated code. In *Proceedings of the 15th IEEE International Conference on Software Maintenance*, pages 109–118, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[35] C. Ebert and T. Liedtke. An integrated approach for criticality prediction. In *Proceeding of the 6th International Symposium on Software Reliability Engineering*, pages 14–23, Toulouse, France, 1995. IEEE Computer Society Press.

[36] K. EL-Emam. A methodology for validating software product metrics. Technical Report NRC 44142, National Research Council Canada, 2000.

[37] S. Elbaum, G. Rothermel, S. Karre, and M. Fisher. Leveraging user-session data to support web application testing. *IEEE Transactions on Software Engineering*, 31(3):187–202, 2005.

[38] N. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering*, 26(8):797–814, 2000.

[39] N. Fenton and S. Pfleeger. *Software Metrics A Rigourous and Practical Approach*. PWS, 2 edition, 1998.

[40] N. Fenton and S. Pfleeger. *Metrics and Models in Software Quality Engineering*. Addsion Wesley, 1 edition, 2003.

[41] A. Field. *Discovering Statistics Using SPSS*. Sage Publications, 2 edition, 2005.

[42] R. Fielding and R. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, 2002.

[43] F. Fioravanti and P. Nesi. Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems. *IEEE Transactions on Software Engineering*, 27(12):1062–1084, 2001.

[44] M. Genero, M. Piattini, and C. Calero. Empirical validation of class diagram metrics. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering*, pages 195–203, Nara, Japan, 2002. IEEE Computer Society Press.

[45] M. Genero, M. Piattini, and E. Manso. Finding early indicators of uml class diagrams understandability and modifiability. In *Proceedings of the 2004 International Symposium on Empirical Software Engineering*, pages 257–268, Redondo Beach, CA, 2004. IEEE Computer Society Press.

[46] M. Genero, M. Piattini, E. Manso, and G. Cantone. Building UML class diagram maintainability prediction models based on early metrics. In *Proceedings of the Ninth International Software Metrics Symposium*, pages 263–275, Sydney, Australia, 2003. IEEE Computer Society Press.

[47] E. Ghosheh and S. Black. Wapmetrics: a tool for computing UML design metrics for web applications. In *Proceedings of the 7th IEEE/ACS International Conference on Computer Systems and Applications*, pages 682–689, Rabat, Moroco, 2009. IEEE Computer Society Press.

[48] E. Ghosheh, S. Black, and J. Qaddour. An introduction of new UML design metrics for web applications. *International Journal of Computer & Information Science*, 8(4):600–609, 2007.

[49] M. P. Giuliano Antoniol and M. Zazzara. Understanding web applications through dynamic analysis. In *Proceeding of the 12th IEEE International Workshop on Program Comprehension*, pages 120–129, Bari, Italy, 2004. IEEE Computer Society Press.

[50] T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source softwaer for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910, 2005.

[51] T. Gymothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910, 2005.

[52] M. Hall. *Core Servlets and Java Server Pages*. Upper Saddle River, 1 edition, 2000.

[53] IBM. Rational Rose Enterprise Edition. Website. `http://www-306.ibm.com/software/awdtools/developer/rose/index.html`.

[54] IEEE. IEEE-14764 STD software engineering, software life cycle processes, maintenance, 2006.

[55] T. Isakowitz, A. Kamis, and M. Koufaris. Extending the capabilities of RMM: Russian Dolls and Hypert. In *Proceedings of 30th Hawaii International Conference on System Science*, pages 177–186, Maui, Hawaii, 1997. University of Hawai at Manoa.

[56] T. Isakowitz, E. Stohr, and P. Balasubramanian. RMM: a methodology for structured hypermedia design. *Communications of the ACM*, 38(8):34–44, 1995.

[57] R. Johnson. An introduction to the bootstrap. *Teaching Statistics*, 23(2):49–54, 2001.

[58] T. Khoshgoftaar, J. Munson, B. Bhattacharya, and G. Richardson. Predictive modeling techniques of sofware quality from software measures. *IEEE Transactions on Software Engineering*, 18(11):979–987, 1992.

[59] M. Kiewkanya, N. Jindasawat, and P. Muenchaisri. A methodology for constructing maintainability model of object-oriented design. In *Proceedings of the 4th International Conference on Quality Software*, pages 206–213, Braunschweig, Germany, 2004. IEEE Computer Society Press.

[60] E. Kirda, M. Jazayeri, C. Kerer, and M. Schranz. Experiences in engineering flexible web services. *IEEE MultiMedia*, 8(1):58–65, 2001.

[61] B. Kitchenham and I. Myrtveit. A simulation study of the model evlauation criterion mmre. *IEEE Transactions on Software Engineering*, 29(11):985–995, 2003.

[62] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El-Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(08):721–734, 2002.

[63] B. Kitchenhaum, S. Pleeger, and N. Fenton. Towards a framework for software measurement. *IEEE Transactions on Software Engineering*, 21(12):929–944, 1995.

[64] S. Kojarski and D. Lorenz. Domain driven web development with WebJinn. In *the 18th International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 53–65. ACM Press, 2003.

[65] P. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, 1995.

[66] W. Kurt and P. Oman. Software maintainability metrics models in practice. *Crosstalk, Journal of Defense Software Engineering*, 8(11):19–23, 1995.

[67] M. Lehman, J. Ramil, P. Wernick, D. Perry, and W. Turski. Metrics and laws of software evolution the nineties view. In *Proceedings of the 4th International Software Metrics Symposium*, pages 20–32, Albuquerque, NM, USA, 1997. IEEE Computer Society Press.

[68] W. Li and S. Henry. Object-oriented metrics that predict maintainability. *Journal of Software Systems*, 23(2):111–122, 1993.

[69] W. Li, S. Henry, D. Kafura, and R. Schulman. Measuring object-oriented design. *Journal of Object-Oriented Programming*, 8(4):48–55, 1995.

[70] M. Mario, E. Manso, and G. Cantone. Building UML class diagram maintainability prediction models based on early metrics. In *Proceedings of the 9th International Software Metrics Symposium*, pages 263–278, Sydney, Australia, 2003. IEEE Computer Society Press.

[71] E. Mendes, N. Mosley, and S. Counsell. Web metrics - estimating design and authoring effort. *IEEE Multimedia*, 08(01):50–57, 2001.

[72] E. Mendes, N. Mosley, and S. Counsell. A comparison of development effort estimation techniques for web hypermedia applications. In *Proceedings of the*

8th International Software Metrics Symposium, pages 131–140, Ottawa, Canada, 2002. IEEE Computer Society Press.

[73] E. Mendes, N. Mosley, and S. Counsell. Early web size measures and effort prediction for web costimation. In *Proceedings of the 9th International Software Metrics Symposium*, pages 18–39, Sydney, Australia, 2003. IEEE Computer Society Press.

[74] E. Mendes, N. Mosley, and S. Counsell. The need for web engineering: An introduction. In *Web Engineering*, chapter 1, pages 1–26. Springer-Verlag, 2006.

[75] K.-H. Moller and D. Paulish. An empirical investigation of software fault distribution. In *Proceedings of the 4th International Software Metrics Symposium*, pages 82–90, Baltimore, MD, USA, 1993. IEEE Computer Society Press.

[76] N. Moreno, P. Fraternali, and A. Vallecillo. A UML 2.0 profile for WebML modeling. In *Proceedings of the 6th International Conference on Web Engineering*, page 4, Palo Alto, California, USA, 2006.

[77] J. Munson and T. Khoshgoftaat. The detection of fault-prone programs. *IEEE Transactions on Software Engineering*, 18(5):423–433, 1992.

[78] G. Mustafa, A. Shah, K. Asif, and A. Ali. A strategy for testing web based software. *Information Technology Journal*, 6(1):74–81, 2007.

[79] I. Myrtveit and E. Stensrud. A controlled experiment to assess the benefits of estimating with analogy and regression models. *IEEE Transactions on Software Engineering*, 25(04):510–525, 1999.

[80] I. Myrtveit, E. Stensrud, and M. Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineering*, 31(05):380–391, 2005.

[81] N. Ohlsson and H. Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*, 22(12):886–894, 1996.

[82] P. Oman and J. Hagemeister. Construction of testing polynomials predicting software maintainability. *Journal of Software Systems*, 27(3):251–266, 1994.

[83] OMG. Xml Metadata Interchange. Website. `http://www.omg.org/`.

[84] R. S. Pressman. *Software Engineering: A Practioner's Approch*. McGraw-Hill, 6 edition, 2004.

[85] D. Reifer. Web development: estimating quick-time-to-market. *IEEE Software*, 17(8):57–64, 2000.

[86] G. Rhode and R. Shapiro. Falling through the net: toward digital inclusion. Technical report, United Stated Department of Commerce, 2000.

[87] F. Ricca. Analysis, testing and re-structuring of web applications. In *Proceedings of the 20th IEEE International Conference on Software Maintenance*, pages 474–478, Chicago Illinois, USA, 2004. IEEE Computer Society Press.

[88] F. Ricca, M. Penta, M. Torchiano, P. Tonella, and M. Ceccato. An empirical study on the usefulness of conallens stereotypes in web application comprehension. In *Proceedings of the Eighth IEEE International Symposium on Web Site Evolution*, pages 58–68, Philadelphia, PA, USA, 2006. IEEE Computer Society Press.

[89] F. Ricca and P. Tonella. Understanding and restructuring web sites with ReWeb. *IEEE Multimedia*, 8(2):40–51, 2001.

[90] M. Ruhe, R. Jeffery, and S. Wieczorek. Using web objects for estimating software development effort for web applications. In *Proceedings of the 9th International Software Metrics Symposium*, pages 30–37, Sydney, Australia, 2003. IEEE Computer Society Press.

[91] D. Schwabe, L. Esmeraldo, G. Rossi, and F. Lyardet. Engineering web applications for reuse. *IEEE Multimedia*, 8(1):20–31, 2001.

[92] J. Singer and N. Vinson. Ethical issues in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 28(12):1171–1180, 2002.

[93] H. Sneed and A. Merey. Automated software quality assurance. *IEEE Transactions on Software Engineering*, 11(9):909–916, 1985.

[94] O. Source. Display tag. Website. `http://displaytag.sourceforge.net/11/`.

[95] R. Subramanyam and M. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering*, 29(04):297–310, 2003.

[96] Sun. Pet store application. Website. `http://www.java.sun.com`.

[97] N. Wilde and R. Huitt. Maintenance support for object-oriented programs. *IEEE Transactions on Software Engineering*, 18(12):1038–1044, 1992.

[98] C. Wohlin, M. Host, and K. Henningsson. Empirical research methods in web and software engineering. In *Web Engineering*, chapter 13, pages 409–429. Springer-Verlag, 2006.

[99] Y. Wu and J. Offutt. Modeling and testing web-based applications. Technical Report ISE-TR-02-08, George Mason University/Department of Information and Software Engineering, 2002.

[100] F. Zhuo, B. Lowther, P. Oman, and J. Hagemeister. Constructing and testing software maintainability assessment models. In *Proceedings of the 1st International Software Metrics Symposium*, pages 61–70, Baltimore, MD, USA, 1993. IEEE Computer Society Press.

A

Empirical Surveys

General Questionnaire Please answer each question provided in this questionnaire.

Personal Identification and Experience:

Name:

Email:

Rate your perceived web application modeling experience (refer to Glossary):

None (1) O Little (2) O Average (3) O Good (4) O Very good (5)

Please answer the questions based on your experience scale:

1. What is your experience with software engineering principles?(circle number)1 2 3 4 5

2. What is your experience with design documents in general?(circle number) 1 2 3 4 5

3. What is your experience with web application modeling?(circle number) 1 2 3 4 5

4. What is your experience performing impact analysis?(circle number) 1 2 3 4 5

Motivation and Performance

1. Estimate how motivated you were to perform well in the study?(circle number) 1 2 3 4 5

2. Estimate how well you understood what was required of you?(circle number) 1 2 3 4 5

3. What approach did you adopt to the exercise?(circle number)

a) Read the documents fully and then attempt the tasks.

b) Read the documents while thinking about the tasks.

c) Straight into the tasks, reading the documents as required.

d) Other- please specify.

4. Estimate the correctness(in percent) of your answers to the questionnaire. 0-20 21-40 41-60 61-80 81-100

5. If you could not complete all the tasks, please indicate why?(circle number)

a) Ran out of time.

b) Did not fully understand the task.

c) Did not fully understand the design documents. d) Other- please specify.

6. In your opinion, what caused you the most difficulty to understand the design documents?(circle number)

a) Nothing in particular.

b) The notation used.

c) Number of relationships in the diagram

d) Size of design document

e) Other- please specify

7. In your opinion, what caused you the most difficulty to perform impact analysis on the design documents?(circle number)

a) Nothing in particular.

b) The notation used.

c) Number of relationships in the diagram.

d) Size of design document

e) Other- please specify Miscellaneous

Miscellaneous

1. How do you judge the size of the design documents you had?(circle number)

Too Small Small About Right Large Too Large 1 2 3 4 5

2. On a scale of 1 to 10 estimate, in terms of understandability, the quality of design documents you had?

1- barely understandable; 10 easily understandable

3. On a scale of 1 to 10 estimate, estimate your overall understanding of the design documents?

1- very little; 10 - complete

4. What did you understand least about the design documents and why? Please specify?

5. On a scale of 1 to 10 estimate in terms of modifiability the quality of design documents?

1-barely modifiable 10- easily modifiable

6. On a scale of 1-10 estimate the overall difficulty of the tasks you have been asked to perform?

1- very easy; 10 very difficult

7. Having performed the tasks would you do anything different next time?

8. Have you learned anything from participating in this study?

9. any additional comments

Thank you

Design Diagram Questionnaire:

Please answer each question provided in this questionnaire. Please enter the time in seconds and minutes. There might be more than one correct answer for some questions.

Personal Identification and Experience:

Name:

Email:

Questions Diagram: welcomeScreen-web-design.Gif:

1. How many server pages are in the diagram?

2. How many client pages are in the diagram?

3. What is the key server page (page with most relationships) in the diagram?

Please enter time(seconds and minutes) for answering questions 1-3:

4. When removing footer.jsp page what are the components and relationships in the diagram that will be affected?

5. If you need to include preference.jsp in the template.jsp page what are the relationships and components that will be added to the diagram?

Please enter time(seconds and minutes) for answering questions 4-5:

Diagram: createCustomer-web-design.GIF

1. How many form pages are there in the diagram?

2. How many attributes are in create-customer.jsp form page (named form1)?

Please enter time(seconds and minutes) for answering questions 1-2:

3. If you need to add account.jsp as an html link to the banner.jsp page what are the relationships and components that will be added to the diagram?

4. If you need to add an HTML select attribute named color to the banner.jsp form page what are the components that will be added to the diagram?

Please enter time(seconds and minutes) for answering questions 3-4:

Diagram: cartScreen-web-design.GIF

1. How many use tags relationships are there in the diagram?

2. How many aggregation relationships are there in the diagram?

Please enter time(seconds and minutes) for answering questions 1-2:

3. If you need to add a JSP tag class named struts to the cart.jsp form page what are the components and relationships that will be added to the diagram?

Please enter time(seconds and minutes) for answering question 3:

Diagram: createUserSequence.gif

1. How many components in the diagram are in the web tier?

2. How many session beans are in the diagram?

3. How many methods does SignOnEJB component have?

Please enter time(seconds and minutes) for answering questions 1-3:

Data For PetStore Experiment

Table A.1 shows the data for the PetStore Experiment. The column UndTime stands for Understandability Time while the column ModTime stands for Modifiability time.

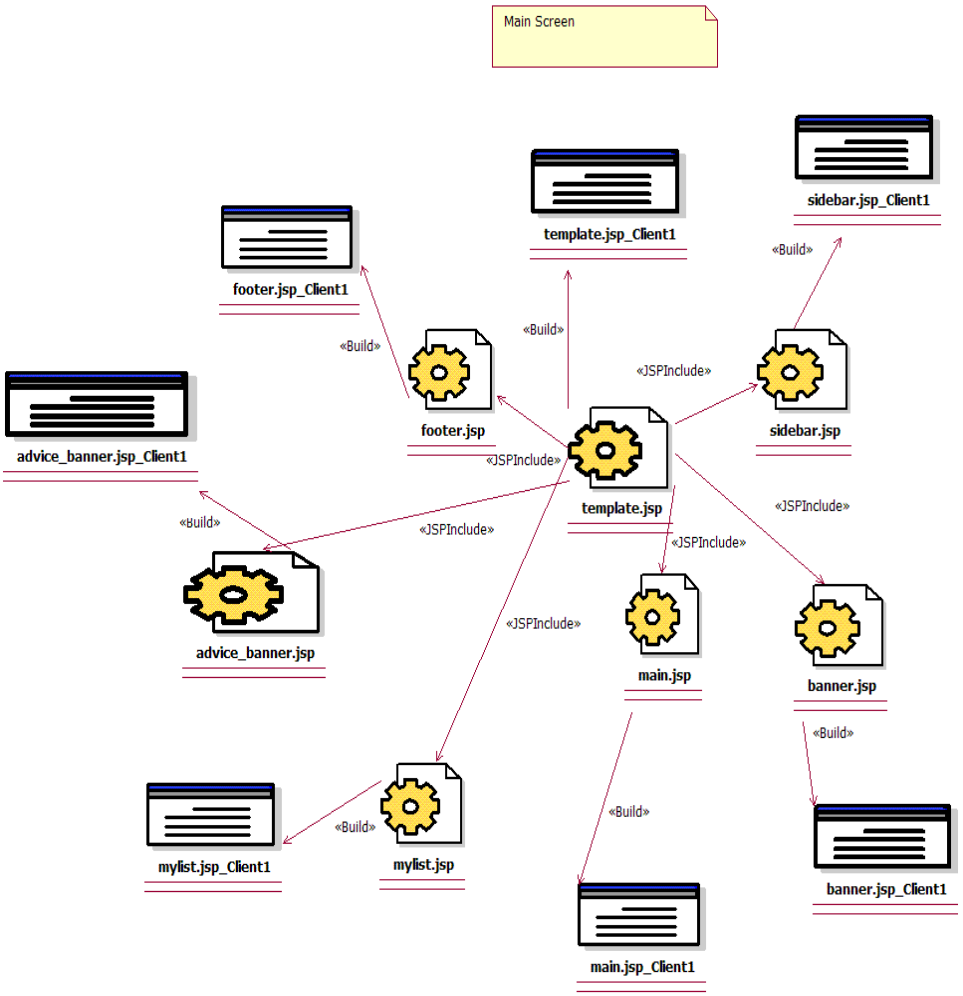


Figure A.1: welcomeScreen Design

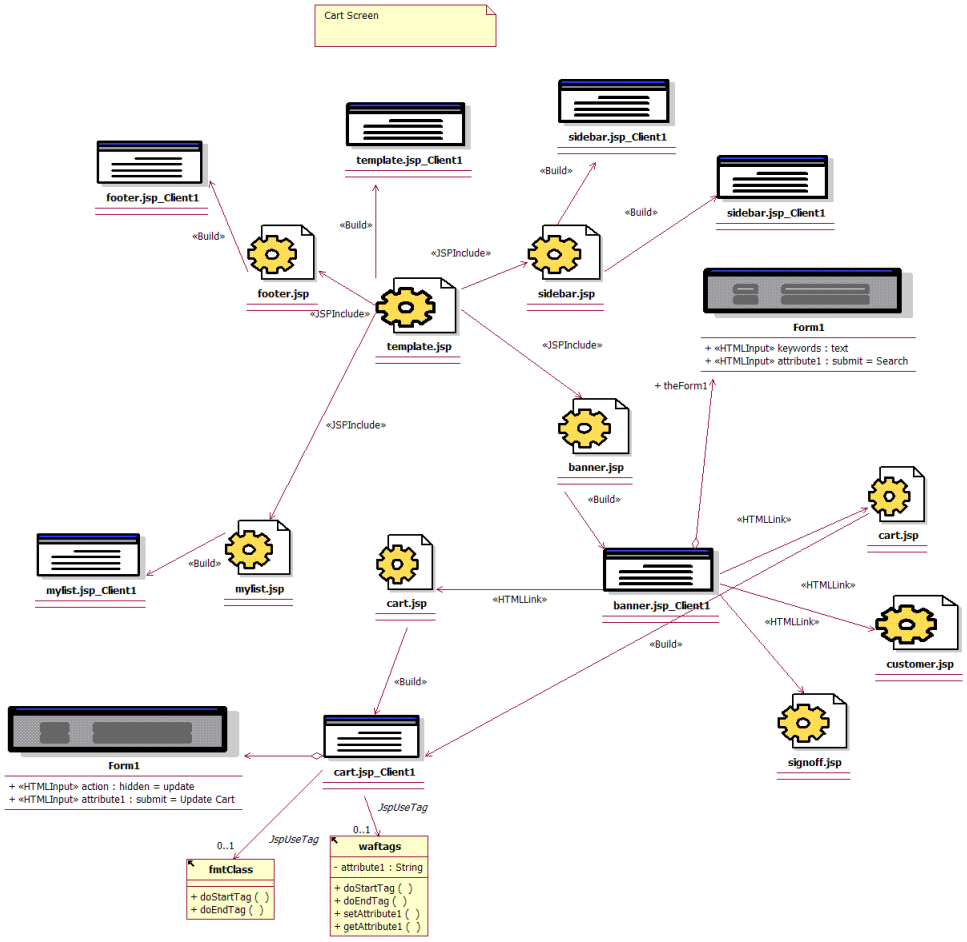


Figure A.2: Cart Design

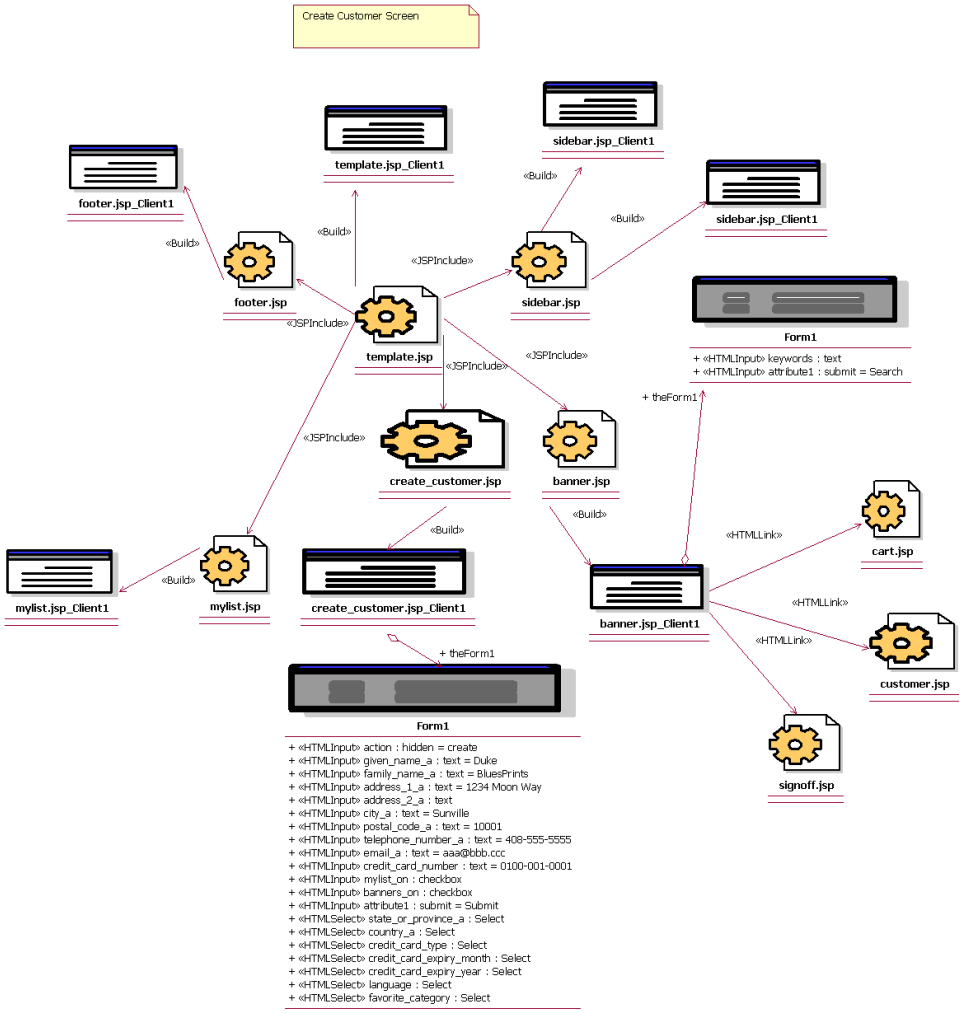


Figure A.3: Create Customer Design

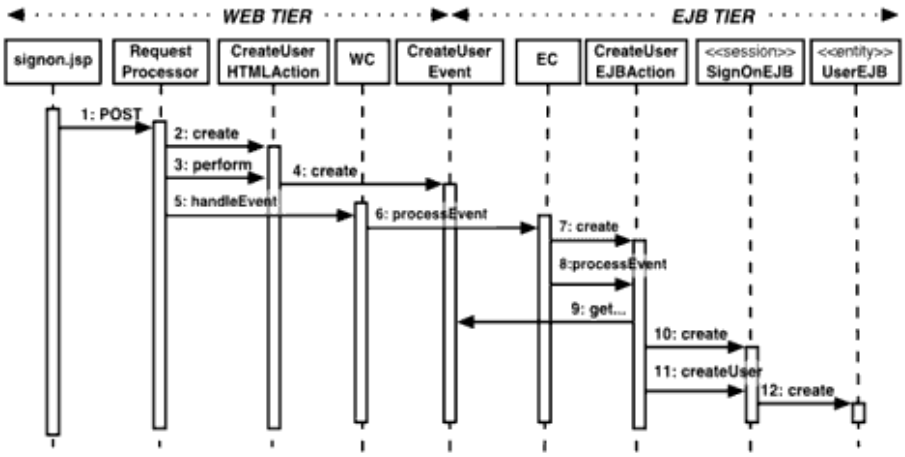


Figure A.4: Create User Design

UndTime welcome-Screen	**ModTime welcome-Screen**	**UndTime createCus-tomer**	**ModTime createCus-tomer**	**UndTime cartCus-tomer**	**ModTime cartCus-tomer**
60	120	60	60	30	120
45	120	20	60	20	15
60	270	45	315	15	60
70	310	125	290	120	90
60	360	120	180	120	120
63	420	90	180	120	120
30	60	15	15	15	30
30	60	15	15	15	30
40	120	40	140	140	40
25	180	30	420	20	180
20	180	60	420	60	240
32	174	131	210	207	91
70	200	54	132	86	42
25	70	20	120	60	80
60	120	60	240	120	60
20	120	30	300	30	120
25	15	25	25	15	10
157	180	60	180	120	180
45	90	10	45	45	60
30	60	15	180	30	180
45	90	120	120	90	120
35	163	113	120	83	72
45	120	20	60	20	15
60	270	45	315	15	60
10	7	10	15	5	5
120	180	60	180	60	180
180	120	60	120	60	60
180	105	86	323	132	104
88	127	30	120	27	31
20	150	80	60	120	60

Table A.1: Data For PetStore Experiment

UndTime welcome-Screen	**ModTime welcome-Screen**	**UndTime createCus-tomer**	**ModTime createCus-tomer**	**UndTime cartCus-tomer**	**ModTime cartCus-tomer**
51	198	97	224	52	85
30	60	30	60	45	60
320	208	42	525	30	-
30	30	30	50	45	30
60	135	30	180	120	90
90	120	90	180	240	120
180	120	60	120	60	60
180	240	120	180	60	180
90	300	160	60	190	120
40	180	240	120	180	240
10	50	20	180	20	60
15	60	30	60	20	20
180	180	120	300	120	300
300	10	10	15	240	180
180	300	120	360	60	120
90	120	60	120	120	120
150	90	60	60	120	60
75	185	75	240	50	60
5	10	60	10	60	60
45	160	55	175	135	80
50	110	75	100	70	60
61	62	52	60	60	63
60	90	30	60	30	30

Table A.1: Data For PetStore Experiment

B

Empirical Case Studies

The web application used is from the telecommunication Operational Support System (OSS) domain. It is a provisioning application which is used to provision and activate the wireless service in the network. We refer to the web application as ProvisionApp. ProvisionApp has around 10,000 users of which 2,500 are concurrent. It is a critical application that is used by customer care advocates to resolve provisioning issues for wireless subscribers. ProvisionApp is built using the latest web technologies and frameworks such Struts, and EJBs and uses Oracle for the database. Table B.3 shows the characteristics of ProvisionApp.

Characteristic	**WebApp1**
Application Domain	Telecom
Age	4
Language	Java
Web Server	WebLogic 7.0
Application Server	WebLogic 7.0
Framework	Struts 1.1
Database	Oracle
Configuration Management Tool	CVS
Design Tool	Rational Rose

Table B.1: Characteristics of ProvisionApp

B.1 ProvisionApp Interfaces

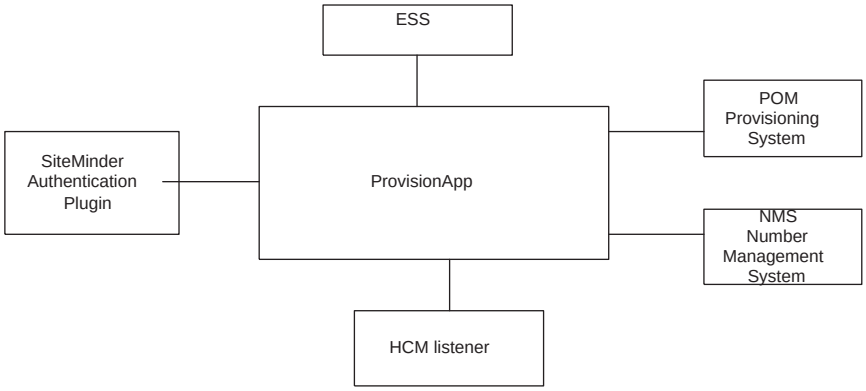


Figure B.1: ProvisionApp Interfaces

The ProvisionApp has the following interfaces:

SiteMinder

The SiteMinder is a plugin to apache server to provide authentication and authorization for the users. The authentication and authorization is done against an LDAP server.

POM

The Provisioning Order Manager serves as the work order manager for provisioning data received from the billing applications. It is used to deliver service orders from Renaissance and P2K billing applications to Actiview or to individual network service applications.

NMS

The Number Management System is the central repository for all wireless phone numbers (MDN/MSID), ESN, and NAI NAI data. The equipment attributes and subscription relationships currently held in NMS will migrate to a centralized database, leaving NMS to focus specifically on number management.

ESS

The Enterprise Security System interface provides the ability for users to initiate Vision Password Reset.

HCM Listener

The Handset Configuration Manager (HCM) application is responsible for over-the-air programming of 3G devices. The HCM delivers 3G parameters, but it will eventually download PRLs, software, and NAM information to devices capable of processing InternetProtocol Over The Air (IOTA) messages.

B.2 ProvisionApp Functional Modules

The ProvisionApp is divided into the following functional modules:

Login Module

Figure B.2: Login Module

The Login Module provides the ability to access the ProvisionApp by entering a

users Network Active Directory domain UserID and Password. Each user can have one of the following three roles:

- Admin: This user profile has unlimited functionality.
- Troubleshooter: This profile has the following capabilities: ”View List of Customer Transactions”, ”View Customer Transaction details” View Network/Device Transactions”,”View Network/Device Transaction details ”, View Erred Transactions (by interface category) ”Retry an erred Network/Device Transaction”,”View Active NAIs ” Initiate a Vision Password Reset”,”Force Close erred Network/Device transactions for all System/Network Elements except AAA.”, ”View Vision Password History ”, ”View Current Network Provisioning Status ”, ”Retrigger IOTA”
- General User: This profile has the following capabilities: ”View list Customer Transactions ”, ”View Customer Transaction details”,”View Network/Device Transactions”,View Network/Device Transaction details ”,”View Active NAIs”,”Initiate a Vision Password Reset”, ”View Vision Password History”,”View Current Network Provisioning Status”

Search Module

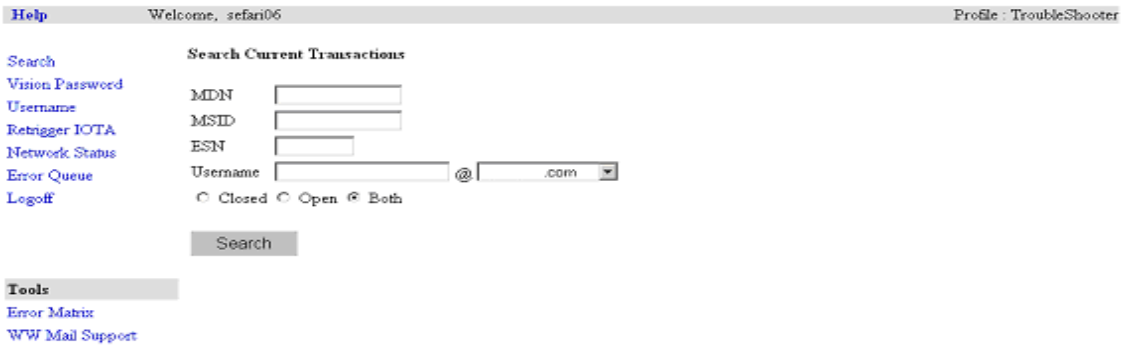


Figure B.3: Search Module

The Search Module is the first module presented to the User upon successful authentication of the User ID and Password. It provides the ability to Search for transaction by one of the following user identifiers: MDN, MSID, ESN or NAI. It also Provides the ability to search Open, Closed or both transactions. Once the parameters are entered and search button is selected the user will be taken to the Current Transactions module.

Current Transactions Module

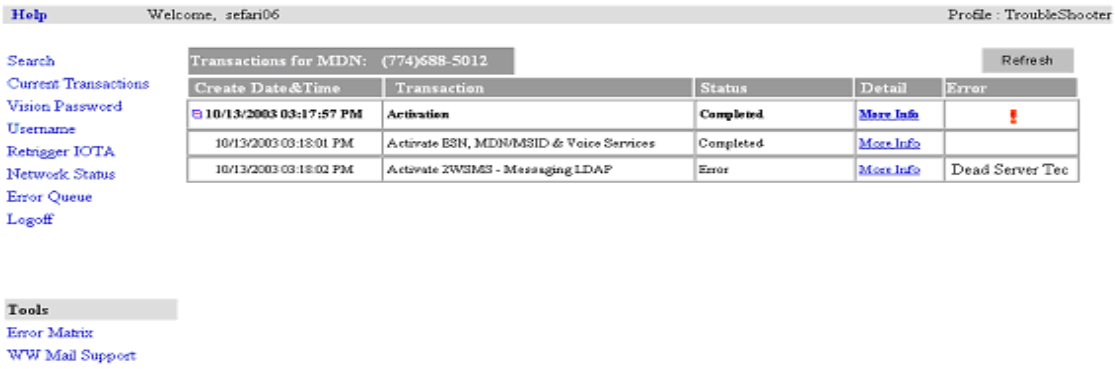


Figure B.4: Current Transactions Module

The Current Transactions Module has the following abilities: ”Displays the current and archived Service transactions with oldest transactions displayed at the top”, ”Provides ability to get More Information about the parent - Service Transaction ”, ”Provides the ability to retrieve Child - Network/Device transactions associated with the Parent - Service Transaction”, ”Provides the ability to get ”More Info” about the child - Network/Device Transaction”, ”Provides the ability to see Network/Device transactions for more than 1 service transaction at a time”, ”Provides ability to refresh transactions so that the user will not have to perform another search to view transactions updates”

Service Transaction Module

The Service Transaction Module provides detailed information about the Service Transaction as shown in Figure B.5

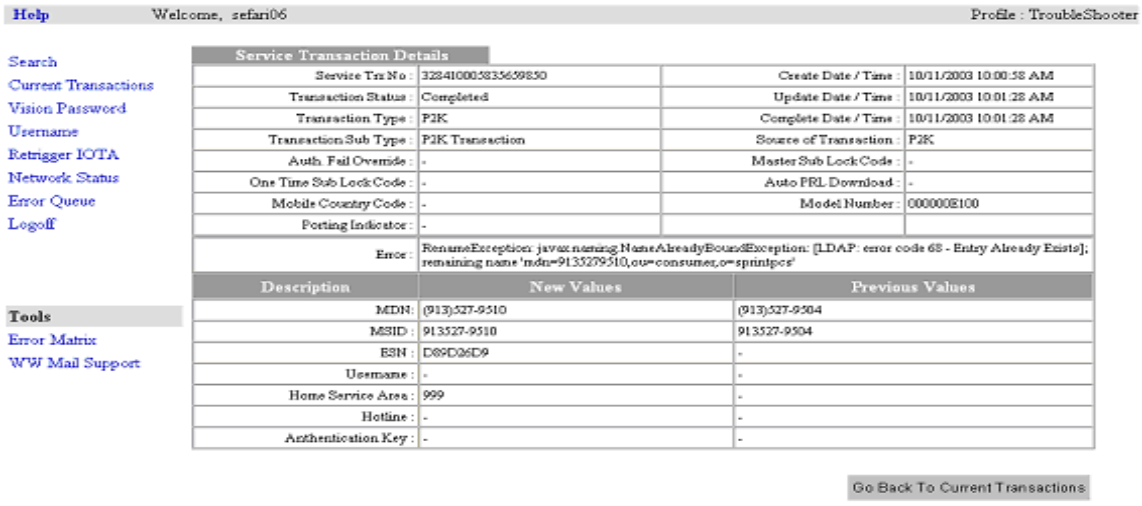


Figure B.5: Service Transaction Module

Device Transaction Module

The Device Transaction Module provides the following abilities: ” Provides detailed information about the Network/Device Transaction as shown in Figure B.6”, Provides the ability to Retry an Erred Transaction (if a transaction is any status other than Error the Retry button will be disabled), ” Provides the ability to Force Close an Erred Transaction” (if a transaction is any status other than Error the Force Close button will be disabled),” Provides transaction history (Error history and Transaction Activity) for the selected Network/Device Transaction with most recent displaying at top”, ”Provides the ability to refresh the Network/Device Transaction information ”, ”Error Log and Activity Log boxes provide the ability to scroll. This will provide visibility to all errors and activity for the Network/Device Transaction.”, ”The Error Message field will expand to display the entire error message so that the user doesn’t have to scroll to see one particular error message.

UserName Module

The UserName Module provides for the following abilities: ” Provides a list of Active and/or Reserved Usernames stored in the Number Management System database for a

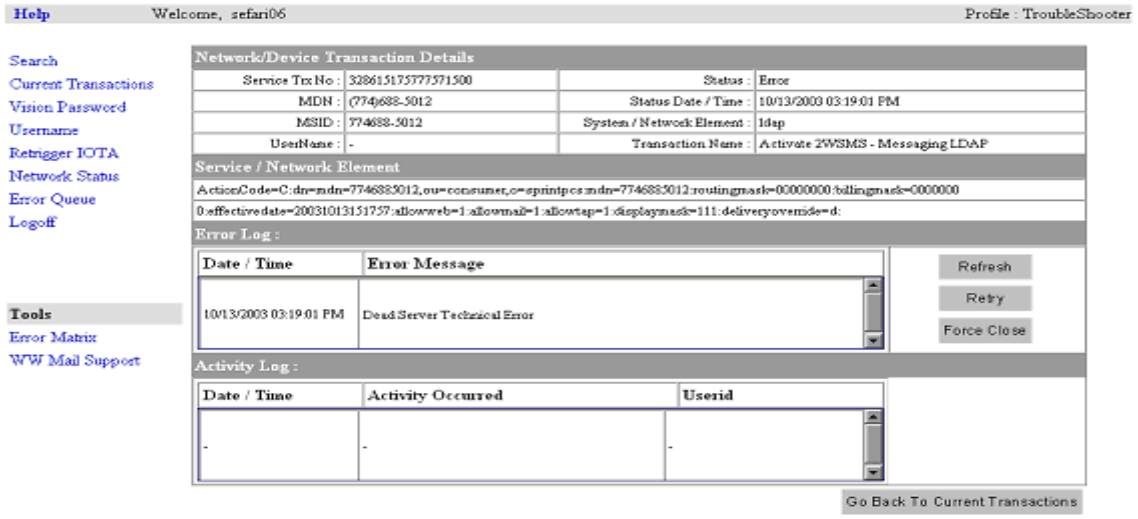


Figure B.6: Device Transaction Module

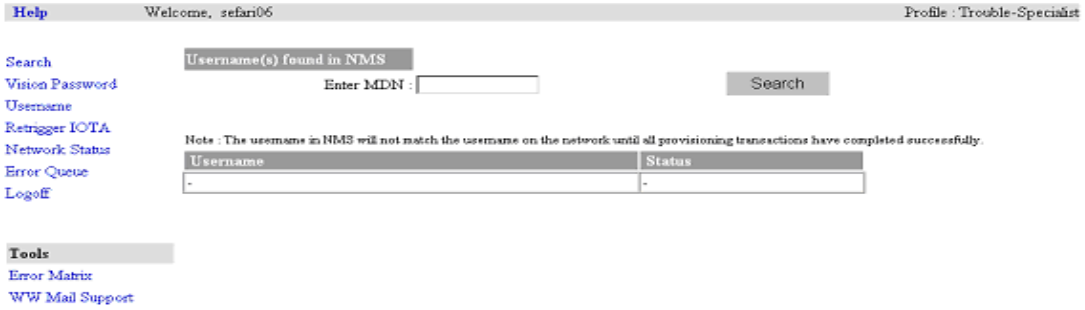


Figure B.7: UserName Module

user.", "Provides the individual status for each Username ", "If Usernames do not exist for a user then the display will show: "Username not found".

Retrigger IOTA Module

The Retrigger IOTA Module has the following abilities: "Provides the ability to retrigger handset programming", The user will not be able to Retrigger IOTA for a given MDN within 30 seconds of another Retrigger IOTA request. ", "Upon selecting Retrigger IOTA the application should check to see if another Retrigger IOTA request, for a given MDN, has been sent within the last 30 seconds. If a request has been made then an error

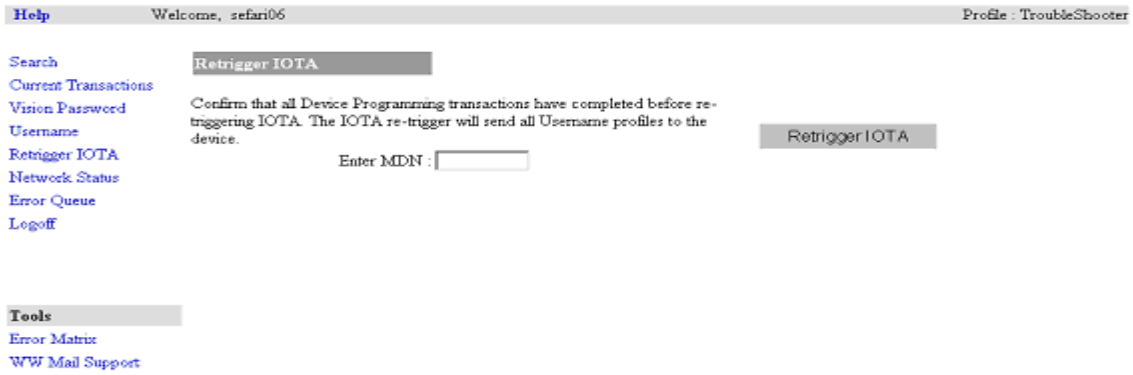


Figure B.8: Retrigger IOTA Module

shall display to the user stating: ”IOTA can only be retriggered every 30 seconds. Check Pending Transactions and try again later.”

Error Queue Module

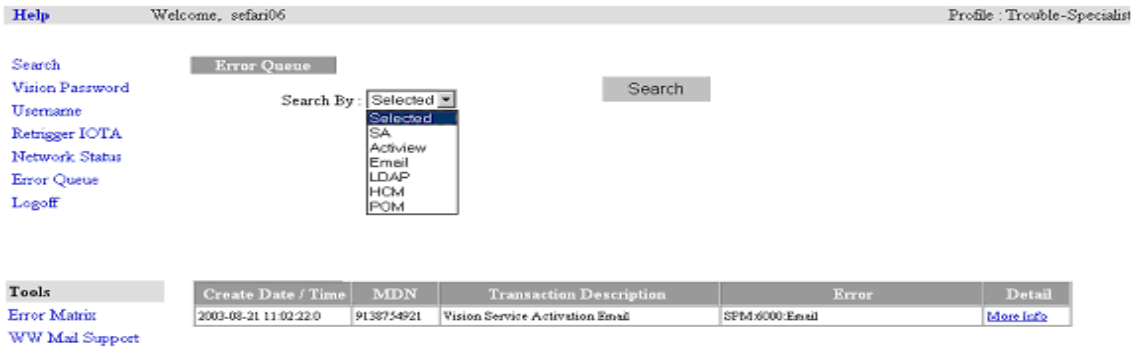


Figure B.9: Error Queue Module

The Error Queue Module has the following abilities: ”Provides a list of Error Transactions by System/Network Element ”, ” Provides the ability to sort Erred transactions by Error or Create Date/Time ”, ”Provides ability to get ’more info’ about the Transaction. This will launch the Network/Device Transaction Details page. ” ”Provides the ability refresh the display.”

Vision Password Module

Figure B.10: Vision Password Module

The Vision Password Module has the following abilities: ”Provides Vision Password History for the user with most recent history at top. ”, ”Provides the ability to reset the Vision Password for the Username entered. ”, ”Vision Password history will not be available until the entire service transaction for Vision Service Activation has completed successfully. ”

Network Provisioning Status Module

The Network Status Module has the following abilities: ”Provides the system/network elements where the Username is currently provisioned , ”Provides user profile information ”, ”Network Status information will not be available until the entire service transaction for Vision Service Activation has completed successfully”

Help Module

The Help Module has the following abilities: ” Provides Help desk about the Provsion-App GUI” , ”Based upon the link selected the user will be navigated to the specific section that discusses that topic ”, ”Provides the ability to return to the previous screen by selecting a back button”

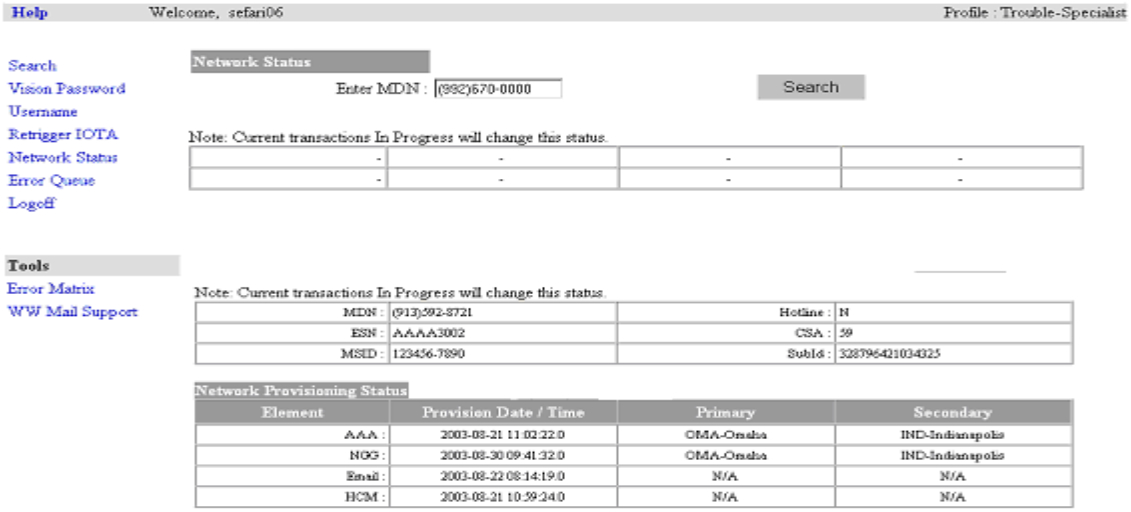
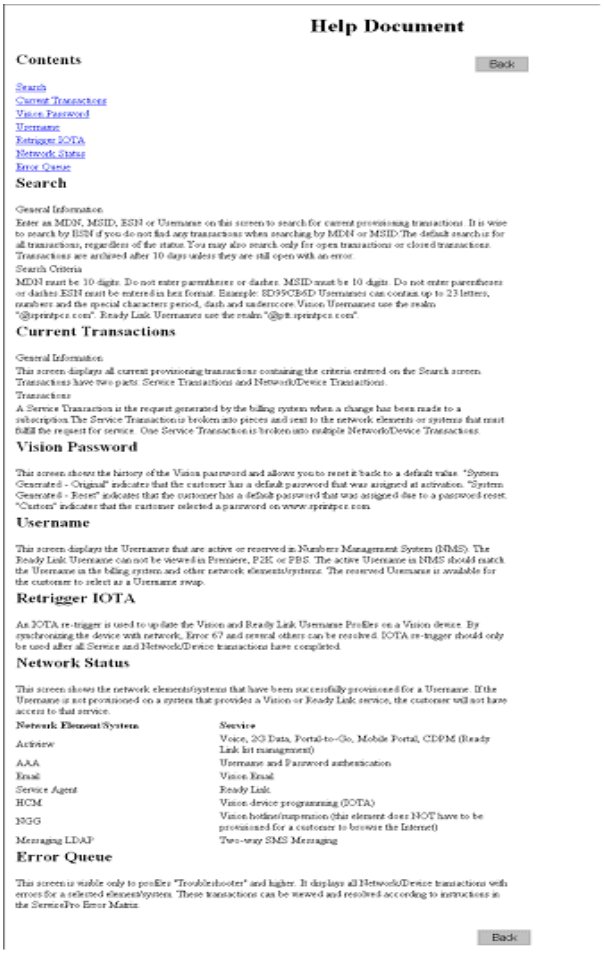


Figure B.11: Network Provisioning Module

B.3 Industrial Provisioning Web Application Data



Help Document

Contents

Back

Search
Current Transactions
Vision Password
Username
Retrigger IOTA
Network Status
Error Queue

Search

General Information

Enter an MDN, MSID, ESN or Username on this screen to search for current provisioning transactions. It is wise to search by ESN if you do not find any transactions when searching by MDN or MSID. The default search is for all transactions, regardless of the status. You may also search only for open transactions or closed transactions. Transactions are archived after 10 days unless they are still open with an error.

Search Criteria

MDN must be 10 digits. Do not enter parentheses or dashes. MSID must be 10 digits. Do not enter parentheses or dashes. ESN must be entered in hex format. Example: 8D99CB8D. Usernames can contain up to 23 letters, numbers and the special characters period, dash and underscore. Vision Usernames use the realm "@sprintpcs.com". Ready Link Usernames use the realm "@pit.sprintpcs.com".

Current Transactions

General Information

This screen displays all current provisioning transactions containing the criteria entered on the Search screen. Transactions have two parts: Service Transactions and Network/Device Transactions.

Transactions

A Service Transaction is the request generated by the billing system when a change has been made to a subscription. The Service Transaction is broken into pieces and sent to the network elements or systems that must fulfill the request for service. One Service Transaction is broken into multiple Network/Device Transactions.

Vision Password

This screen shows the history of the Vision password and allows you to reset it back to a default value. "System Generated - Original" indicates that the customer has a default password that was assigned at activation. "System Generated - Reset" indicates that the customer has a default password that was assigned due to a password reset. "Custom" indicates that the customer selected a password on www.sprintpcs.com.

Username

This screen displays the Usernames that are active or reserved in Numbers Management System (NMS). The Ready Link Username can not be viewed in Premiere, P2K or PBS. The active Username in NMS should match the Username in the billing system and other network elements/systems. The reserved Username is available for the customer to select as a Username swap.

Retrigger IOTA

An IOTA re-trigger is used to update the Vision and Ready Link Username Profiles on a Vision device. By synchronizing the device with network, Error 67 and several others can be resolved. IOTA re-trigger should only be used after all Service and Network/Device transactions have completed.

Network Status

This screen shows the network elements/systems that have been successfully provisioned for a Username. If the Username is not provisioned on a system that provides a Vision or Ready Link service, the customer will not have access to that service.

Network Element/System	Service
Actiview	Voice, 2G Data, Portal-to-Go, Mobile Portal, CDPM (Ready Link list management)
AAA	Username and Password authentication
Email	Vision Email
Service Agent	Ready Link
HCM	Vision device programming (IOTA)
NGG	Vision hotline/suspension (this element does NOT have to be provisioned for a customer to browse the Internet)
Messaging LDAP	Two-way SMS Messaging

Error Queue

This screen is visible only to profiles "Troubleshooter" and higher. It displays all Network/Device transactions with errors for a selected element/system. These transactions can be viewed and resolved according to instructions in the ServicePro Error Matrix.

Back

Figure B.12: Help Module

Class Diagram Name	Revisions	LOC
deviceTrx.jsp	8	80
deviceTrxDetail.jsp	229	10019
error.jsp	8	108
errorBucket.jsp	13	32
help.jsp	11	1649
failure.jsp	2	13
logon.jsp	56	1165
mainSearch.jsp	147	4889
networkStatus.jsp	9	35
popUp.html	3	34
retriggerIota	8	41
roleNotFound.jsp	2	102
servicenode.jsp	207	7894
servicetree.jsp	12	124
serviceTrxDetail.jsp	106	2550
sessionTimeout.jsp	8	229
smlogin.html	2	320
userName.jsp	8	26
visionPassword.jsp	6	24
header.inc	10	38
leftNavigation.inc	49	468
errorBucket.inc	91	1941
errorQueue.inc	96	2016
networkStatusResult.inc	146	10696
networkStatusSearch.inc	203	13179
retriggerIotaSearch.inc	179	5889
userNameResult.inc	101	2623
usernameSearch.inc	169	6017
visionPasswordResult.inc	172	5920
visionPasswordSearch.inc	162	5306

Table B.2: Data for Dependent Variables for the Industrial Provisioning Web Application

Class Diagram Name	NServerP	NClientP	NWebP	NFormP	NFormE
error.jsp	3	1	4	1	0
deviceTrx.jsp	3	1	4	0	0
deviceTrxDetail.jsp	3	1	4	2	2
errorBucket.jsp	4	1	5	0	0
help.jsp	1	1	2	0	0
failure.jsp	1	1	2	1	2
logon.jsp	1	1	2	2	10
mainSearch.jsp	3	1	4	2	13
networkStatus.jsp	4	1	5	0	0
popUp.html	0	1	1	0	0
roleNotFound.jsp	1	1	2	0	0
servicenode.jsp	5	1	6	0	0
servicetree.jsp	3	1	4	0	0
serviceTrxDetail.jsp	3	1	4	0	0
sessionTimeout.jsp	4	2	6	0	0
smlogin.html	0	1	1	1	6
userName.jsp	4	1	5	0	0
visionPassword.jsp	4	1	5	0	0
retriggerIota.jsp	4	1	5	0	0
header.inc	1	1	2	0	0
leftNavigation.inc	1	1	2	0	0
errorBucket.inc	1	1	2	0	0
errorQueue.inc	1	1	2	1	1
networkStatusResult.inc	1	1	2	0	0
networkStatusSearch.inc	1	1	2	1	1
retriggerIotaSearch.inc	1	1	2	1	1
userNameResult.inc	1	1	2	0	0
usernameSearch.inc	1	1	2	1	1
visionPasswordResult.inc	1	1	2	0	0
visionPasswordSearch.inc	1	1	2	1	1

Table B.3: Data for Independent Variables for the Industrial Provisioning Web Application

Class Diagram Name	NLinkR	NForwardR	NIncludeR	NClientScriptComp
error.jsp	0	0	2	0
deviceTrx.jsp	0	0	2	0
deviceTrxDetail.jsp	3	0	2	3
errorBucket.jsp	0	0	3	0
help.jsp	6	0	0	0
failure.jsp	0	0	0	0
logon.jsp	0	0	0	0
mainSearch.jsp	0	0	2	4
networkStatus.jsp	0	0	3	0
popUp.html	0	0	0	0
roleNotFound.jsp	0	0	0	0
servicenode.jsp	0	0	4	0
servicetree.jsp	2	0	2	0
serviceTrxDetail.jsp	0	0	2	0
sessionTimeout.jsp	0	0	2	0
smlogin.html	0	0	0	0
userName.jsp	0	0	3	0
visionPassword.jsp	0	0	3	0
retriggerIota.jsp	0	0	3	0
header.inc	1	0	0	0
leftNavigation.inc	9	0	0	1
errorBucket.inc	2	0	0	1
errorQueue.inc	0	0	0	1
networkStatusResult.inc	0	0	0	0
networkStatusSearch.inc	0	0	0	3
retriggerIotaSearch.inc	0	0	0	2
userNameResult.inc	0	0	0	0
usernameSearch.inc	0	0	0	2
visionPasswordResult.inc	0	0	0	1
visionPasswordSearch.inc	0	0	0	1

Table B.3: Data for Independent Variables for the Industrial Provisioning Web Application- Continued

Class Diagram Name	NServerScriptsComp	NC	NA	NM	NAssoc	NAgg
error.jsp	3	0	0	0	0	1
deviceTrx.jsp	3	0	0	0	0	0
deviceTrxDetail.jsp	10	6	80	176	6	2
errorBucket.jsp	4	0	0	0	0	0
help.jsp	0	0	0	0	0	0
failure.jsp	2	0	0	0	0	1
logon.jsp	4	7	36	70	6	1
mainSearch.jsp	10	8	41	70	6	2
networkStatus.jsp	8	0	0	0	0	0
popUp.html	0	0	0	0	0	0
roleNotFound.jsp	0	0	0	0	0	0
servicenode.jsp	8	12	218	472	14	0
servicetree.jsp	1	2	8	17	2	0
serviceTrxDetail.jsp	7	5	120	249	6	0
sessionTimeout.jsp	3	0	0	0	0	0
smlogin.html	0	0	0	0	0	1
userName.jsp	7	0	0	0	0	0
visionPassword.jsp	7	0	0	0	0	0
retriggerIota.jsp	9	0	0	0	0	0
header.inc.jsp	1	0	0	0	0	0
leftNavigation.inc	1	0	0	0	0	1
errorBucket.inc	1	5	19	47	5	1
errorQueue.inc	5	5	19	47	5	2
networkStatusResult.inc	3	7	22	46	6	0
networkStatusSearch.inc	4	7	22	46	6	2
retriggerIotaSearch.inc	4	10	186	378	10	2
userNameResult.inc	4	7	34	70	7	0
usernameSearch.inc	4	7	34	70	7	2
visionPasswordResult.inc	3	5	25	58	5	1
visionPasswordSearch.inc	4	5	25	58	5	2

Table B.3: Data for Independent Variables for the Industrial Provisioning Web Application- Continued

Class Diagram Name	CoupEntropy	CohesionEntropy	BuildWeight	SubmitWeight
error.jsp	.00385	31.8920	3	0
deviceTrx.jsp	.00298	41.2198	3	0
deviceTrxDetail.jsp	.00596	20.6099	13	2
errorBucket.jsp	.00346	35.5048	4	0
help.jsp	.00447	27.4799	0	0
failure.jsp	.00298	41.2198	2	2
logon.jsp	.00495	24.8168	4	10
mainSearch.jsp	.00551	22.2783	14	13
networkStatus.jsp	.00346	35.5048	8	0
popUp.html	.00000	1.0000	0	0
roleNotFound.jsp	.00149	82.4397	0	0
servicenode.jsp	.00644	19.0747	8	0
servicetree.jsp	.00447	27.4799	1	0
serviceTrxDetail.jsp	.00495	24.8168	7	0
sessionTimeout.jsp	.00346	35.5048	6	0
smlogin.html	.00149	82.4397	0	0
userName.jsp	.00346	35.5048	7	0
visionPassword.jsp	.00346	35.5048	7	0
retriggerIota.jsp	.00346	35.5048	9	0
header.inc	.00236	52.0136	1	0
leftNavigation.inc	.00534	22.9960	2	0
errorBucket.inc	.00495	24.8168	2	0
errorQueue.inc	.00495	24.8168	6	1
networkStatusResult.inc	.00447	27.4799	3	0
networkStatusSearch.inc	.00516	23.8304	7	1
retriggerIotaSearch.inc	.00582	21.1011	6	1
userNameResult.inc	.00472	26.0068	4	0
usernameSearch.inc	.00534	22.9960	6	1
visionPasswordResult.inc	.00447	27.4799	4	0
visionPasswordSearch.inc	.00495	24.8168	5	1

Table B.3: Data for Independent Variables for the Industrial Provisioning Web Application- Continued

Class Diagram Name	WebControlCoupling	WebDataCoupling	WebReusability
error.jsp	1.2500	.0000	.5000
deviceTrx.jsp	1.2500	.0000	.5000
deviceTrxDetail.jsp	5.0000	.6667	.5000
errorBucket.jsp	1.4000	.0000	.6000
help.jsp	3.0000	.0000	.0000
failure.jsp	2.0000	2.0000	.0000
logon.jsp	7.0000	10.0000	.0000
mainSearch.jsp	7.2500	4.3333	.5000
networkStatus.jsp	2.2000	.0000	.6000
popUp.html	.0000	.0000	.0000
roleNotFound.jsp	.0000	.0000	.0000
servicenode.jsp	2.0000	.0000	.6667
servicetree.jsp	1.2500	.0000	.5000
serviceTrxDetail.jsp	2.2500	.0000	.5000
sessionTimeout.jsp	1.3333	.0000	.3333
smlogin.html	.0000	.0000	.0000
userName.jsp	2.0000	.0000	.6000
visionPassword.jsp	2.0000	.0000	.6000
retriggerIota.jsp	2.4000	.0000	.6000
header.inc	1.0000	.0000	.0000
leftNavigation.inc	5.5000	.0000	.0000
errorBucket.inc	2.0000	.0000	.0000
errorQueue.inc	3.5000	1.0000	.0000
networkStatusResult.inc	1.5000	.0000	.0000
networkStatusSearch.inc	4.0000	1.0000	.0000
retriggerIotaSearch.inc	3.5000	1.0000	.0000
userNameResult.inc	2.0000	.0000	.0000
usernameSearch.inc	3.5000	1.0000	.0000
visionPasswordResult.inc	2.0000	.0000	.0000
visionPasswordSearch.inc	3.0000	1.0000	.0000

Table B.3: Data for Independent Variables for the Industrial Provisioning Web Application- Continued

C

WapMetrics Tool

WapMetrics is an open source tool that is used for measuring UML design metrics for web applications. The tool can be downloaded at `http://www.sueblack.co.uk/WapMetrics.war`, and instructions for installation and usage can be found in Appendix D. WapMetrics provides an automated way to measure UML metrics and has the ability to show the results in different output formats. This research uses UML design metrics rather than source code metrics for measuring maintainability as many studies have shown that early metrics are much more useful [18, 20, 47].

C.1 WapMetrics Tool

It is important to have an automated tool for computing UML metrics from design diagrams. WapMetrics is a web tool that takes UML diagrams in XMI [83] format as input and produces the results in HTML format. The WapMetrics tool has the following features:

- it can measure and calculate web application metrics from UML diagrams based on the Conallen model. Most other tools concentrate on UML metrics for object-oriented applications. In addition to that, it is independent from the CASE tool used to build the models, and takes an XMI file as input. The XMI file describes the UML model in a standard way. The XMI input allows the exchange of model information in a standard way regardless of the CASE tool used to create the XMI

file.

- it is a web application that can be deployed on a central server and used by many users without installing it on the client machines. This makes it easy to maintain and deploy enhancements to the WapMetrics tool.

- it provides interoperability and, the outcome of WapMetrics is user friendly and easy usable by other tools. WapMetrics allows the output to be exported in several formats: HTML, XML, pdf, excel, rtf and csv. This allows the output to be used for statistical reporting and the results to be presented in graphs and other formats.

The WapMetrics tool architecture is composed of three components:

1. *Presentation Component*: responsible for getting the input from the user and displaying the results in HTML.

2. *Controller Component*: mainly responsible for communicating back and forth between the Presentation Component and the Business Component.

3. *Business Component*: responsible for the parsing and computation of the metrics.

Figure C.1 shows the architecture components of the WapMetrics tool.

C.1.1 Presentation Component

The presentation component provides the user interface for starting the WapMetrics tool. The presentation component has been implemented with Java, JSP, JavaScript, HTML and stylesheets. Figure C.2 shows the main screen which has XMI [83] and email input. The XMI file contains the design of the model in XML [83] format. Many UML design tools are able to export their design to the XMI format which makes the tool interoperable with them. The email input allows the user to get an email with the results once processing is complete, the email input is validated on the client side using JavaScript to make sure the email has the correct format. Figure C.3 shows part of the results screen which is implemented using JSP [52] and the display tag [94]. The results

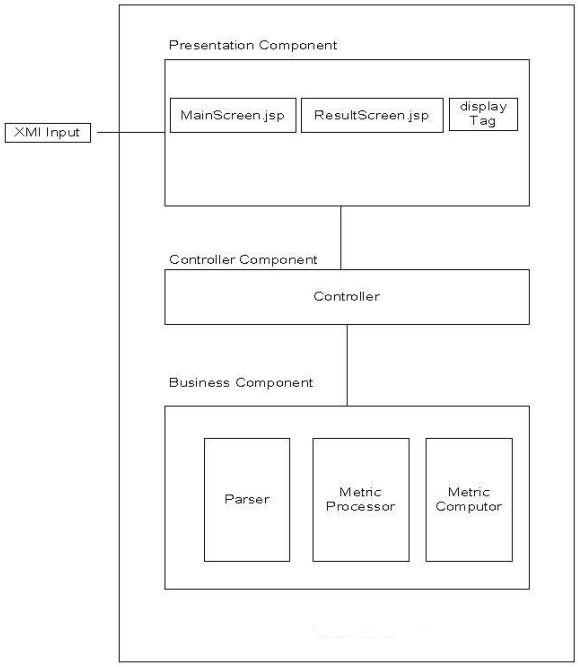


Figure C.1: WapMetrics Tool Architecture

screen allows the user to export the result in different output formats. The result can be exported in XML, pdf, excel, rtf and csv formats.

C.1.2 Controller Component

The controller component is the communication medium between the presentation component and business component. It is implemented totally in Java and provides some validation on the input data. The controller component carries out the validation on the XMI input file to make sure it is well formatted. If it finds errors in the format, it displays an error message to the presentation component, otherwise it passes the data to the business component for further processing.

C.1.3 Business Component

The business component is the main component of the application. It is responsible for the extraction, analysis and display of the results of the metrics computation. The

Figure C.2: WapMetrics MainScreen

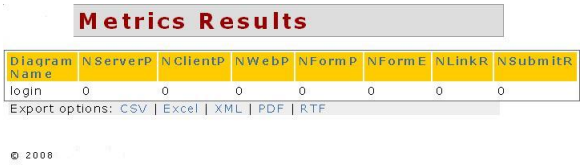


Figure C.3: WapMetrics Results Screen

business component is composed of three parts: the parser, metrics processor, and metric computor. The parser is a wrapper of the SAX parser version SAX 2.0.1. It extracts the data from the XMI input file and puts the data in object classes. It creates an Array which has all the diagrams as elements. The array is passed to the metric processor which extracts all the diagrams and calls the corresponding method on the metric computor. The metric computor implements the algorithms for computing all the metrics defined in Table 4.2.1.

C.2 Case Study

C.2.1 Introduction

We use Claros [23]in our case study. Claros is an open source project with the goal of providing an easy to use personal information suite for its users. This case study uses Claros inTouch version 2.1 [23]. Claros inTouch is an Ajax communication suite having the following components: webmail, address, book, post-it notes, calendar, webdisk,

Figure C.4: Claros Home Screen [23]

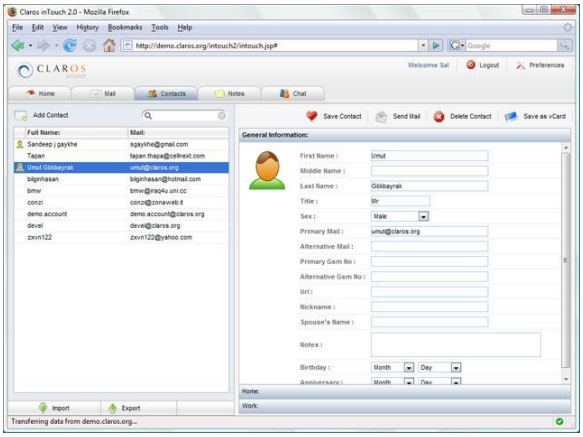


Figure C.5: Claros Contacts Screen [23]

built-in instant messenger and rss reader. It is an open source web application using web 2.0 technologies.

Figure C.4 shows the home screen for Claros which has tabs for the Mail, Contacts, Notes, and chat components. In our study we will use the Contacts component shown in Figure C.5. The Contacts component allows the user to add a new contact, save contact, send mail to contact, delete contact and save contact as vCard. The Contacts components stores general information about the user, home address and work address.

C.2.2 Data Collection

The WapMetrics tool takes as input class diagrams in XMI format. Unfortunately, there were no preexisting class diagrams for the Claros web application, so, we set up a running Claros web application to help us understand, and reverse engineer the Contacts component of Claros. To run Claros successfully we had to install the following components: Java 1.5 or higher which can be downloaded from the SUN website [96], Tomcat5.x webserver which can be donwloaded from the apache website [23], MySQL for the database which can be downloaded from the MySQL download center [23]. After setting up all the software components, Claros source code was downloaded and added to the web folder in Tomcat. Finally, we started the webserver and opened the home page for the Claros web application.

For generating the class diagram for the Claros web application IBM Rational Rose Enterprise Edition [53] was used. Rational Rose has a visual modeling component, which can create the design artifacts of a software system. The Web Modeler component in Rational Rose supports Conallen's extension for web applications. The Web Modeler component was used to generate the class diagram for the Contacts components in the Claros web application. Figure C.6 shows the class diagram for the Contacts component which was validated by comparing the running Claros web application and the source code with the class diagram. After the class diagram was validated, Unisys Rose XML [53] was used to export the UML class diagrams into XML Metadata Interchange (XMI) [83]. The WapMetrics tool was used to compute the metrics defined in Table 4.2.1 from the XMI input file.

C.2.3 Results

Table C.4 shows the results of applying the WapMetrics tool on the class diagram shown in Figure C.6. The results have been validated by computing the metrics manually from the class diagrams and comparing the output to results from the WapMetrics tool. As shown in Table C.4 the Contacts component has four server pages (NServerP). The number of form elements (NFormE) is thirty six which is quite high for a single form

Metric Name	Value
NServerP	4
NClientP	1
NWebP	5
NFormP	1
NFormE	36
NClientC	19
NLinkR	7
NSubmitR	1
NbuildsR	1
NForwardR	0
NIncludeR	3
NUseTagR	0
WebControlCoupling	2.4
WebDataCoupling	9
WebReusability	0.6
NC	14
NA	60
NM	130
NAssoc	26
NAgg	2

Table C.4: Claros Contacts Component Results

page. The number of client components (NClientC) is nineteen. This is expected since Claros is an Ajax application and uses a lot of Javascript. The number of methods and attributes in the controller, model and service classes is also high. The Claros Contacts component has many classes with sixty attributes and one hundred and thirty methods. This means that there is a considerable amount of development effort needed on the Java side of the Contacts component. The metrics results from WapMetrics were as expected since the Contacts component is one of the biggest components in Claros.

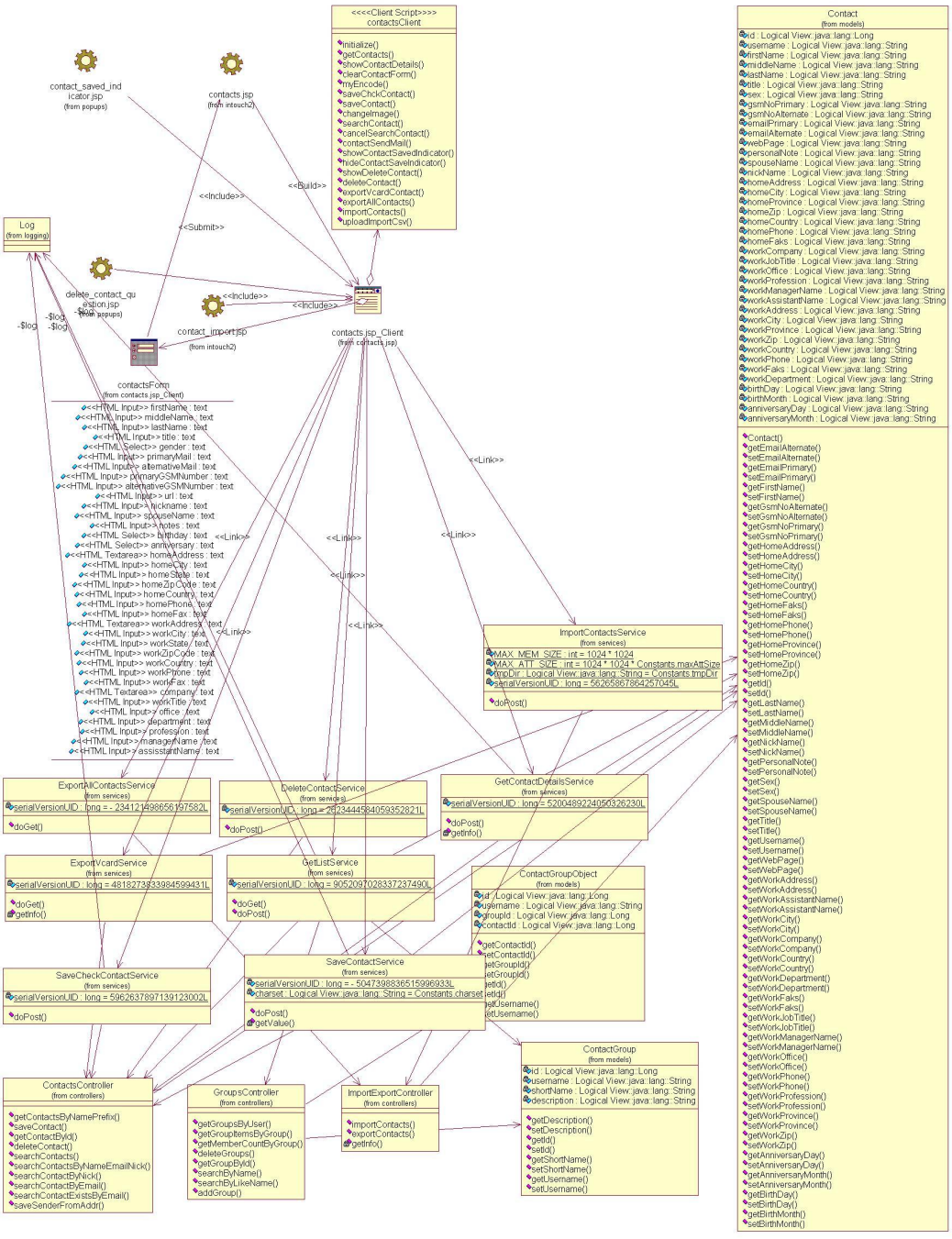


Figure C.6: Claros Contacts Class Diagram

D

WapMetrics Source Code

D.1 Installation Instructions

Please follow the following instruction for installing and using the WapMetrics Tool:

1. Download the WapMetrics tool from
   `http://www.sueblack.co.uk/WapMetrics.war`
2. Deploy the WapMetrics.war file to a web server
3. Put the XMI file in your C directory
4. Start the web server and go to the main screen index.jsp
5. Wait till your results are shown on the screen

D.2 XMLFileParser

```
package webmetrics.utils;

import java.io.PrintWriter; import java.util.ArrayList; import
java.util.HashMap;

import org.w3c.dom.Document; import org.w3c.dom.Element; import
org.w3c.dom.NodeList; import org.xml.sax.SAXException; import
```

```
org.xml.sax.SAXParseException;

import webmetrics.domain.ClassDiagram;

import dom.GetElementsByTagName; import dom.ParserWrapper;
public class XMLFileParser {

 /** Namespaces feature id (http://xml.org/sax/features/namespaces).
 */
 protected static final String NAMESPACES_FEATURE_ID =
        "http://xml.org/sax/features/namespaces";

 /** Validation feature id (http://xml.org/sax/features/validation).
 */
protected static final String VALIDATION_FEATURE_ID =
"http://xml.org/sax/features/validation";

 /**
 * Schema validation feature id
 * (http://apache.org/xml/features/validation/schema).
 */

 protected static final String SCHEMA_VALIDATION_FEATURE_ID =
 "http://apache.org/xml/features/validation/schema";

 /**
 * Schema full checking feature id
 * (http://apache.org/xml/features/validation/schema-full-checking).
 */


 protected static final String SCHEMA_FULL_CHECKING_FEATURE_ID =
 "http://apache.org/xml/features/validation/schema-full-checking";

    /**
     * Honour all schema locations feature id
```

```
     * (http://apache.org/xml/features/honour-all-schemaLocations).
     */

    protected static final String HONOUR_ALL_SCHEMA_LOCATIONS_ID =
    "http://apache.org/xml/features/honour-all-schemaLocations";

/**
 * Validate schema annotations feature id
 * (http://apache.org/xml/features/validate-annotations).
 */

protected static final String VALIDATE_ANNOTATIONS_ID =
 "http://apache.org/xml/features/validate-annotations";

 /**
 * Dynamic validation feature id
 * (http://apache.org/xml/features/validation/dynamic).
 */



 protected static final String DYNAMIC_VALIDATION_FEATURE_ID =
 "http://apache.org/xml/features/validation/dynamic";

 /** XInclude feature id (http://apache.org/xml/features/xinclude). */
 protected static final String XINCLUDE_FEATURE_ID =
 "http://apache.org/xml/features/xinclude";

/**
 * XInclude fixup base URIs feature id
* (http://apache.org/xml/features/xinclude/fixup-base-uris).
 */
 protected static final String XINCLUDE_FIXUP_BASE_URIS_FEATURE_ID =
"http://apache.org/xml/features/xinclude/fixup-base-uris";

/**
```

```
 * XInclude fixup language feature id
 * (http://apache.org/xml/features/xinclude/fixup-language).
 */

 protected static final String XINCLUDE_FIXUP_LANGUAGE_FEATURE_ID =
 "http://apache.org/xml/features/xinclude/fixup-language";

 // default settings

 /** Default parser name (dom.wrappers.Xerces). */
protected static final String DEFAULT_PARSER_NAME =
 "dom.wrappers.Xerces";

/** Default repetition (1). */
 protected static final int DEFAULT_REPETITION = 1;

 /** Default namespaces support (true). */
 protected static final boolean DEFAULT_NAMESPACES = true;

/** Default validation support (false).
*/ protected static final boolean DEFAULT_VALIDATION = false;

/** Default Schema validation support (false). */
protected static final boolean DEFAULT_SCHEMA_VALIDATION = false;

/** Default Schema full checking support (false).
 */
protected static final boolean DEFAULT_SCHEMA_FULL_CHECKING = false;

/** Default honour all schema locations (false).
*/
protected static final boolean
DEFAULT_HONOUR_ALL_SCHEMA_LOCATIONS = false;

/** Default validate schema annotations (false).
 */
```

```
protected static final boolean DEFAULT_VALIDATE_ANNOTATIONS = false;

/** Default dynamic validation support (false).
*/
protected static final boolean DEFAULT_DYNAMIC_VALIDATION = false;

/** Default XInclude processing support (false).
*/
 protected static final boolean DEFAULT_XINCLUDE = false;

 /** Default XInclude fixup base URIs support (true).
*/
protected static final boolean DEFAULT_XINCLUDE_FIXUP_BASE_URIS = true;

/** Default XInclude fixup language support (true). */
protected static final boolean DEFAULT_XINCLUDE_FIXUP_LANGUAGE = true;

public static ParserWrapper createParser() {
 ParserWrapper parser = null;
  // use default parser?
if (parser == null) {

 // create parser
 try {
 parser = (ParserWrapper) Class.forName(DEFAULT_PARSER_NAME)
 .newInstance();
 } catch (Exception e) {
 System.err.println("error: Unable to instantiate parser ("
 + DEFAULT_PARSER_NAME + ")");
 }
 }
 return parser;
 }


 public static void setParserFeatures(ParserWrapper parser) {
```

```
boolean namespaces = DEFAULT_NAMESPACES;
boolean validation = DEFAULT_VALIDATION;
boolean schemaValidation = DEFAULT_SCHEMA_VALIDATION;
 boolean schemaFullChecking = DEFAULT_SCHEMA_FULL_CHECKING;
boolean honourAllSchemaLocations =
DEFAULT_HONOUR_ALL_SCHEMA_LOCATIONS;
boolean validateAnnotations = DEFAULT_VALIDATE_ANNOTATIONS;
boolean dynamicValidation = DEFAULT_DYNAMIC_VALIDATION;
boolean xincludeProcessing = DEFAULT_XINCLUDE;
boolean xincludeFixupBaseURIs = DEFAULT_XINCLUDE_FIXUP_BASE_URIS;
boolean xincludeFixupLanguage = DEFAULT_XINCLUDE_FIXUP_LANGUAGE;

 try {
   parser.setFeature(NAMESPACES_FEATURE_ID, namespaces);
        } catch (SAXException e) {
  System.err.println("warning: Parser does not support feature ("
   + NAMESPACES_FEATURE_ID + ")");
    }



 try {
  parser.setFeature(VALIDATION_FEATURE_ID, validation);
 } catch (SAXException e) {
  System.err.println("warning: Parser does not support feature ("
   + VALIDATION_FEATURE_ID + ")");
  }

 try {
  parser.setFeature(SCHEMA_VALIDATION_FEATURE_ID, schemaValidation);
   } catch (SAXException e) {
System.err.println("warning: Parser does not support feature ("
+ SCHEMA_VALIDATION_FEATURE_ID + ")");
  }

 try {
parser.setFeature(SCHEMA_FULL_CHECKING_FEATURE_ID,
```

```
schemaFullChecking);
} catch (SAXException e) {
System.err.println("warning: Parser does not support feature ("
+ SCHEMA_FULL_CHECKING_FEATURE_ID + ")");
 }
try {
parser.setFeature(HONOUR_ALL_SCHEMA_LOCATIONS_ID,
honourAllSchemaLocations);
} catch (SAXException e) {
System.err.println("warning: Parser does not support feature ("
 + HONOUR_ALL_SCHEMA_LOCATIONS_ID + ")");
 }
try {
 parser.setFeature(VALIDATE_ANNOTATIONS_ID, validateAnnotations);
 } catch (SAXException e) {
System.err.println("warning: Parser does not support feature ("
+ VALIDATE_ANNOTATIONS_ID + ")");
 }

try {
parser.setFeature(DYNAMIC_VALIDATION_FEATURE_ID, dynamicValidation);
} catch (SAXException e) {
 System.err.println("warning: Parser does not support feature ("
+ DYNAMIC_VALIDATION_FEATURE_ID + ")");
      }
try {
  parser.setFeature(XINCLUDE_FEATURE_ID, xincludeProcessing);
} catch (SAXException e) {
System.err.println("warning: Parser does not support feature ("
 + XINCLUDE_FEATURE_ID + ")");
 }

try {
parser.setFeature(XINCLUDE_FIXUP_BASE_URIS_FEATURE_ID,
xincludeFixupBaseURIs);
 } catch (SAXException e) {
```

```
 System.err.println("warning: Parser does not support feature ("
 + XINCLUDE_FIXUP_BASE_URIS_FEATURE_ID + ")");
  }

try {
parser.setFeature(XINCLUDE_FIXUP_LANGUAGE_FEATURE_ID,
 xincludeFixupLanguage);
 } catch (SAXException e) {
System.err.println("warning: Parser does not support feature ("
+ XINCLUDE_FIXUP_LANGUAGE_FEATURE_ID + ")");
        }

    }

    public static Document parseXMIDocument() {

        // variables
        Document document = null;
        ParserWrapper parser = null;

        parser = createParser();

        // set parser features
        setParserFeatures(parser);

        // parse file
        try {

  // document = parser.parse("C:/XMIFiles/indust2diagrams.xml");
 // document = parser.parse("C:/XMIFiles/moodle.xml");
 document = parser.parse("C:/XMIFiles/claros2.xml");
        }

        catch (SAXParseException e) {
            // ignore
        } catch (Exception e) {
```

```
            System.err.println("error: Parse error occurred - "
                    + e.getMessage());
            Exception se = e;
            if (e instanceof SAXException) {
                se = ((SAXException) e).getException();
            }
            if (se != null)
                se.printStackTrace(System.err);
            else
                e.printStackTrace(System.err);
        }
        return document;
    }

    public static ArrayList parseWebModel() {

        // variables
        ArrayList diagrams = new ArrayList();
        ClassDiagram classDiagram = new ClassDiagram();
        Document document = null;
        ParserWrapper parser = null;

        parser = createParser();

        // set parser features
        setParserFeatures(parser);

        // parse file
        try {

// document = parser.parse("C:/XMIFiles/moodleModel.xml");
document = parser.parse("C:/XMIFiles/claroscontactsModel.xml");
        }

        catch (SAXParseException e) {
            // ignore
```

```
        } catch (Exception e) {
            System.err.println("error: Parse error occurred - "
                    + e.getMessage());
            Exception se = e;
            if (e instanceof SAXException) {
                se = ((SAXException) e).getException();
            }
            if (se != null)
                se.printStackTrace(System.err);
            else
                e.printStackTrace(System.err);
        }


NodeList elements = document.getElementsByTagName("classdiagram");

 for (int i = 0; i < elements.getLength(); i++) {
 Element element = (Element) elements.item(i);
 classDiagram.setName
(element.getAttribute("name"));
classDiagram.setServerpage
(getWebModelElement(element, "serverpage"));
classDiagram.setFormpage
(getWebModelElement(element, "formpage"));
 classDiagram.setClientpage
(getWebModelElement(element, "clientpage"));
 classDiagram.setClientscript
(getWebModelElement(element,"clientscript"));
 classDiagram.setTag
(getWebModelElement(element, "tag"));
 classDiagram.setInterfaceclass
(getWebModelElement(element,"interfaceclass"));
 diagrams.add(classDiagram);
  classDiagram = new ClassDiagram();

  }
```

```
  return diagrams;
    }

 private static ArrayList getWebModelElement(Element element,
 String elementName) {
 ArrayList elementList = new ArrayList();
 NodeList serverPageElements =
element.getElementsByTagName(elementName);
 for (int j = 0; j < serverPageElements.getLength(); j++) {
  Element servelement = (Element) serverPageElements.item(j);
   elementList.add(servelement.getTextContent());
  System.out.println(servelement.getTextContent());
   }
  return elementList;
    }

    public static void setXMIIds(ClassDiagram classDiagram,
Document document) {
   HashMap xmiIDMap = new HashMap();
  ArrayList listOfServerPElements = new ArrayList();
 ArrayList listOfClientPElements = new ArrayList();
  ArrayList listOfClientScriptElements = new ArrayList();
  ArrayList listOfFormPElements = new ArrayList();
  ArrayList listOfInterfaceClassElements = new ArrayList();
  ArrayList listOfTags = new ArrayList();
  listOfServerPElements = classDiagram.getServerpage();
   listOfClientPElements = classDiagram.getClientpage();
  listOfClientScriptElements = classDiagram.getClientscript();
  listOfFormPElements = classDiagram.getFormpage();
   listOfInterfaceClassElements = classDiagram.getInterfaceclass();
  listOfTags = classDiagram.getTag();
  NodeList elements = document.getElementsByTagName("UML:Class");

   for (int i = 0; i < elements.getLength(); i++) {
    Element element = (Element) elements.item(i);
     for (int j = 0; j < listOfServerPElements.size(); j++) {
```

```
    if ((element.getAttribute("name"))
.equalsIgnoreCase((String) listOfServerPElements.get(j))) {
 System.out.println("Web Elements NAME = "
 + element.getAttribute("name"));
 System.out.println("Web Elements XMI ID = "
+ element.getAttribute("xmi.id"));
 xmiIDMap.put(element.getAttribute("xmi.id"), element
 .getAttribute("name"));
                }
            }

for (int j = 0; j < listOfClientPElements.size(); j++) {
if ((element.getAttribute("name"))
.equalsIgnoreCase((String) listOfClientPElements.get(j))) {
System.out.println("Web Elements NAME = "
 + element.getAttribute("name"));
System.out.println("Web Elements XMI ID = "
+ element.getAttribute("xmi.id"));
xmiIDMap.put(element.getAttribute("xmi.id"), element
.getAttribute("name"));
                }
            }

for (int j = 0; j < listOfClientScriptElements.size(); j++) {
if ((element.getAttribute("name"))
.equalsIgnoreCase((String) listOfClientScriptElements
.get(j))) {
 System.out.println("Web Elements NAME = "
+ element.getAttribute("name"));
System.out.println("Web Elements XMI ID = "
+ element.getAttribute("xmi.id"));
xmiIDMap.put(element.getAttribute("xmi.id"), element
.getAttribute("name"));
    }
 }
```

```
for (int j = 0; j < listOfFormPElements.size(); j++) {
 if ((element.getAttribute("name"))
 .equalsIgnoreCase((String) listOfFormPElements.get(j))) {
  System.out.println("Web Elements NAME = "
 + element.getAttribute("name"));
 System.out.println("Web Elements XMI ID = "
 + element.getAttribute("xmi.id"));
 xmiIDMap.put(element.getAttribute("xmi.id"), element
.getAttribute("name"));
    }
  }


 for (int j = 0; j < listOfInterfaceClassElements.size(); j++) {
if ((element.getAttribute("name"))
.equalsIgnoreCase((String) listOfInterfaceClassElements
 .get(j))) {
   System.out.println("Web Elements NAME = "
   + element.getAttribute("name"));
  System.out.println("Web Elements XMI ID = "
 + element.getAttribute("xmi.id"));
  xmiIDMap.put(element.getAttribute("xmi.id"), element
  .getAttribute("name"));
    }
  }

 for (int j = 0; j < listOfTags.size(); j++) {
 if ((element.getAttribute("name"))
 .equalsIgnoreCase((String) listOfTags.get(j))) {
 System.out.println("Web Elements NAME = "
 + element.getAttribute("name"));
 System.out.println("Web Elements XMI ID = "
 + element.getAttribute("xmi.id"));
   xmiIDMap.put(element.getAttribute("xmi.id"), element
 .getAttribute("name"));
    }
```

```
    }

    }classDiagram.setXmiIDMap(xmiIDMap);
     }

}

// GetElementsByTagName.print
(out, document, "UML:Association", "xmi.id");
// xmi.id = 'S.161.1015.17.38'
// NodeList elements =
document.getElementsByTagName("UML:Stereotype");
// NodeList elements =
document.getElementsByTagName("UML:Operation");
// NodeList elements =
 document.getElementsByTagName("UML:Attribute");
// NodeList elements =
document.getElementsByTagName("UML:Class");
/*
 * NodeList elements =
document.getElementsByTagName("UML:Association");
 *
 *
 * for (int i = 0; i < elements.getLength(); i++)
{ Element element =
 * (Element)elements.item(i);
NodeList attrElements =
 * element.getElementsByTagName("UML:Operation");
for (int j = 0; j <
 * attrElements.getLength(); j++)
{ Element elementattr =
 * (Element)attrElements.item(j);
 //if
 * (element.getAttribute("name").
equalsIgnoreCase("visionPasswdClient")) {
 * System.out.println(elementattr.getNodeName());
```

```
 * System.out.println(elementattr.getTextContent());
 * System.out.println(elementattr.getAttribute("name")); } }
 *
 * System.out.println(element.getNodeName());
 * System.out.println(element.getTextContent());
 * System.out.println(element.getAttribute("name")); }
 */
```

D.3 MetricProcessor

```
package webmetrics.utils;

import java.io.PrintWriter; import java.util.ArrayList;

import org.w3c.dom.Document; import org.w3c.dom.NamedNodeMap; import
org.w3c.dom.Node; import org.w3c.dom.Text;

import webmetrics.domain.ClassDiagram; import
webmetrics.domain.MetricEntry; import dom.ParserWrapper;

/**
 * This file computes the metric for the webMetric tool.
 *
 * @author Emad Ghosheh
 *
 * @version 1.0: MetricProcessor.java
 */
public class MetricProcessor {

    /** Number of elements. */
    protected long fElements;

    /** Number of attributes. */
    protected long fAttributes;
```

```
/** Number of characters. */
protected long fCharacters;

/** Number of ignorable whitespace characters. */
protected long fIgnorableWhitespace;

/** Document information. */
protected ParserWrapper.DocumentInfo fDocumentInfo;

//
// Public methods
//

public ArrayList execute() {
    double totalCoupling = 0.0;
    double totalNumberOfElements = 0.0;
    ArrayList diagrams = new ArrayList();
    diagrams = XMLFileParser.parseWebModel();
    Document document = XMLFileParser.parseXMIDocument();
    MetricEntry mEntry = new MetricEntry();
    ArrayList metricResult = new ArrayList();
    ClassDiagram classDiagram = new ClassDiagram();
    for (int j = 0; j < diagrams.size(); j++) {
        classDiagram = (ClassDiagram) diagrams.get(j);
        totalNumberOfElements = totalNumberOfElements
                + classDiagram.getClientpage().size()
                + classDiagram.getClientscript().size()
                + classDiagram.getFormpage().size()
                + classDiagram.getInterfaceclass().size()
                + classDiagram.getServerpage().size()
                + classDiagram.getTag().size();
    }

    classDiagram = new ClassDiagram();
    for (int j = 0; j < diagrams.size(); j++) {
        classDiagram = (ClassDiagram) diagrams.get(j);
```

```
            // set the XMI ID HashMap
            XMLFileParser.setXMIIds(classDiagram, document);

mEntry.setNa
(MetricComputation.calculateNA(classDiagram, document));

mEntry.setNagg
(MetricComputation.calculateNAgg(classDiagram,document));

 mEntry.setNassoc
(MetricComputation.calculateNAssoc(classDiagram,document));

mEntry.setNbuildsr
(MetricComputation.calculateNBuildsR(classDiagram, document));

mEntry.setNc
(MetricComputation.calculateNC(classDiagram, document));

mEntry.setNclientp
(MetricComputation.calculateNClientP( classDiagram, document));

mEntry.setNclientscriptscomp
(MetricComputation .
calculateNClientScriptsComp(classDiagram, document));

mEntry.setNforme
(MetricComputation.calculateNFormE(classDiagram, document));

 mEntry.setNformp
(MetricComputation.calculateNFormP(classDiagram,document));

mEntry.setNforwardr
(MetricComputation.calculateNForwardR(classDiagram, document));

mEntry.setNincluder
(MetricComputation.calculateNIncludeR(classDiagram, document));
```

```
 mEntry.setNlinkr
(MetricComputation.calculateNLinkRTest(classDiagram, document));

mEntry.setNm
(MetricComputation.calculateNM(classDiagram, document));

mEntry.setNserverp
(MetricComputation.calculateNserverP( classDiagram, document));

mEntry.setNserverscriptscomp(MetricComputation.
calculateNServerScriptsComp(classDiagram, document));

 // fixing NserverScripts since it includes number of client scripts
 // String nClientTemp = mEntry.getNclientscriptscomp();
 // String nFormETemp = mEntry.getNforme();
 // int nServInt = Math.abs(Integer.parseInt(nFormETemp) -
  // Integer.parseInt(nClientTemp));
  // mEntry.setNserverscriptscomp(String.valueOf(nServInt));

mEntry.setNsubmitr
(MetricComputation.calculateNSubmitR(classDiagram, document));

mEntry.setNuseTagr
(MetricComputation.calculateNUseTagR(classDiagram, document));

mEntry.setNwebp
(MetricComputation.calculateNWebP(classDiagram,document));

mEntry.setWebcontrolcoupling
(MetricComputation.
calculateWebControlCoupling(classDiagram, document));

mEntry.setWebdatacoupling
(MetricComputation
.calculateWebDataCoupling(classDiagram, document));
```

```
mEntry.setWebreusability
(MetricComputation.calculateWebReusability(
 classDiagram, document));

mEntry.setCoupentropy
(MetricComputation.calculateCoupEntropy(
classDiagram, document, mEntry, totalNumberOfElements));

mEntry.setDiagram
(classDiagram.getName());
totalCoupling = totalCoupling
+ Double.parseDouble(mEntry.getCoupentropy());

            metricResult.add(mEntry);
            classDiagram = new ClassDiagram();
            mEntry = new MetricEntry();

        }

        for (int k = 0; k < metricResult.size(); k++) {
            mEntry = (MetricEntry) metricResult.get(k);
            mEntry.setCohesionentropy(MetricComputation
                    .calculateCohesionEntropy(mEntry, totalCoupling));
        }
        return metricResult;
    }

    /** Sets the parser wrapper. */
    public void setDocumentInfo
(ParserWrapper.DocumentInfo documentInfo) {
        fDocumentInfo = documentInfo;
    } // setDocumentInfo(ParserWrapper.DocumentInfo)

    /** Traverses the specified node, recursively. */
    public void count(Node node) {
```

```
// is there anything to do?
if (node == null) {
    return;
}

int type = node.getNodeType();
switch (type) {
case Node.DOCUMENT_NODE: {
    fElements = 0;
    fAttributes = 0;
    fCharacters = 0;
    fIgnorableWhitespace = 0;
    Document document = (Document) node;
    count(document.getDocumentElement());
    break;
}

case Node.ELEMENT_NODE: {
    fElements++;
    NamedNodeMap attrs = node.getAttributes();
    if (attrs != null) {
        fAttributes += attrs.getLength();
    }
    // drop through to entity reference
}

case Node.ENTITY_REFERENCE_NODE: {
    Node child = node.getFirstChild();
    while (child != null) {
        count(child);
        child = child.getNextSibling();
    }
    break;
}
```

```
        case Node.CDATA_SECTION_NODE: {
            fCharacters += ((Text) node).getLength();
            break;
        }

        case Node.TEXT_NODE: {
            if (fDocumentInfo != null) {
                Text text = (Text) node;
                int length = text.getLength();
                if (fDocumentInfo.isIgnorableWhitespace(text)) {
                    fIgnorableWhitespace += length;
                } else {
                    fCharacters += length;
                }
            }
            break;
        }
        }

    } // count(Node)

    /** Prints the results. */
    public void printResults
(PrintWriter out, String uri, long parse,
 long traverse1, long traverse2, int repetition) {

// filename.xml:
//631/200/100 ms (4 elems, 0 attrs, 78 spaces, 0 chars)

        out.print(uri);
        out.print(": ");
        if (repetition == 1) {
            out.print(parse);
        } else {
            out.print(parse);
            out.print('/');
```

```
            out.print(repetition);
            out.print('=');
            out.print(parse / repetition);
        }
        out.print(';');
        out.print(traverse1);
        out.print(';');
        out.print(traverse2);
        out.print(" ms (");
        out.print(fElements);
        out.print(" elems, ");
        out.print(fAttributes);
        out.print(" attrs, ");
        out.print(fIgnorableWhitespace);
        out.print(" spaces, ");
        out.print(fCharacters);
        out.print(" chars)");
        out.println();
        out.flush();

    } // printResults(PrintWriter,String,long,long,long)

}
```

D.4 MetricComputation

```
/**
 *
 */
package webmetrics.utils;

import java.util.ArrayList; import java.util.HashMap;

import org.w3c.dom.Document; import org.w3c.dom.Element; import
org.w3c.dom.NodeList;
```

```
import webmetrics.domain.ClassDiagram; import
webmetrics.domain.MetricEntry;

/**
 * @author Emad Ghosheh 2008 This class computes all the metrics
 *
 */
public class MetricComputation {

public static String calculateNserverP
(ClassDiagram classDiagram,
 Document document) {

String nserverP = "";

 int numberOfServerP = 0;
 ArrayList listOfServerPages = new ArrayList();

 listOfServerPages =
 classDiagram.getServerpage();

NodeList elements =
 document.getElementsByTagName("UML:Class");

 for (int i = 0; i < elements.getLength(); i++) {
 Element element = (Element) elements.item(i);
for (int j = 0; j < listOfServerPages.size(); j++) {
if ((element.getAttribute("name"))
.equalsIgnoreCase((String) listOfServerPages.get(j))) {
 numberOfServerP++;
        }
       }
   }

 nserverP = String.valueOf(numberOfServerP);
```

```
  return nserverP;

}

 public static String calculateNWebP(ClassDiagram classDiagram,
  Document document) {
  String nWebP = "";

 int nWebPagesCount = Integer.parseIn
t(calculateNClientP(classDiagram,
 document))
    + Integer.parseInt
(calculateNserverP(classDiagram, document));
return nWebP = String.valueOf(nWebPagesCount);
    }


public static String calculateNFormP(ClassDiagram classDiagram,
   Document document) {
        String nFormP = "";

        int numberOfFormP = 0;

 ArrayList listOfFormPages = new ArrayList();
  listOfFormPages = classDiagram.getFormpage();

 NodeList elements =
document.getElementsByTagName("UML:Class");

 for (int i = 0; i < elements.getLength(); i++) {

 Element element = (Element) elements.item(i);

for (int j = 0; j < listOfFormPages.size(); j++) {
```

```
if ((element.getAttribute("name"))

.equalsIgnoreCase((String) listOfFormPages.get(j))) {

numberOfFormP++;
                }
            }
        }

nFormP = String.valueOf(numberOfFormP);


return nFormP;
   }


public static String calculateNFormE(ClassDiagram classDiagram,

Document document) {

String nFormE = "";


int numberOfFormElements = 0;

ArrayList listFormPages = new ArrayList();

listFormPages = classDiagram.getFormpage();

NodeList elements =
document.getElementsByTagName("UML:Class");

for (int i = 0; i < elements.getLength(); i++) {

Element element = (Element) elements.item(i);
```

```
 for (int j = 0; j < listFormPages.size(); j++) {

 if ((element.getAttribute("name"))

.equalsIgnoreCase((String) listFormPages.get(j))) {

 NodeList elementsAttributes = element

.getElementsByTagName("UML:Attribute");

numberOfFormElements = numberOfFormElements

 + elementsAttributes.getLength();
                }

            }

        }


 nFormE = String.valueOf(numberOfFormElements);

 return nFormE;
    }


public static String calculateNLinkR(ClassDiagram classDiagram,

Document document) {

 String nLinkR = "";


 int numberOfLinkR = 0;
```

```
 HashMap sterotypeHashmap = createSterotypeHashMap(document);
NodeList elements = document.getElementsByTagName("UML:Association");
 for (int i = 0; i < elements.getLength(); i++) {
 Element element = (Element) elements.item(i);
 NodeList elementsAssiationEnd = element
.getElementsByTagName("UML:AssociationEnd");

 boolean isParticipate = isParticipateElement(classDiagram,
 elements, elementsAssiationEnd);
  if (isParticipate) {
 System.out.println(" association sterotype attr : "
+ element.getAttribute("stereotype"));
  if (element.getAttribute("stereotype") != null
&& element.getAttribute("stereotype").length() > 0) {
 String xmiStereotype = element.getAttribute("stereotype");
 if (sterotypeHashmap.get(xmiStereotype) != null
&& ((String) sterotypeHashmap.get(xmiStereotype))
.equalsIgnoreCase("link")) {
 numberOfLinkR++;
```

```
 }

  }

}

 }

nLinkR = String.valueOf(numberOfLinkR);


return nLinkR;
    }


 public static String calculateNLinkRTest
(ClassDiagram classDiagram,

Document document) {

String nLinkR = "";


 int numberOfLinkR = 0;

 HashMap sterotypeHashmap = createSterotypeHashMap(document);

  NodeList elements = document.getElementsByTagName("UML:Association");

for (int i = 0; i < elements.getLength(); i++) {

Element element = (Element) elements.item(i);

NodeList elementsAssiationEnd = element
```

```
.getElementsByTagName("UML:AssociationEnd");


boolean isParticipate = isParticipateElementAssoc(classDiagram,

elements, elementsAssiationEnd);

if (isParticipate) {

System.out.println(" association sterotype attr : "

+ element.getAttribute("stereotype"));

if (element.getAttribute("stereotype") != null

&& element.getAttribute("stereotype").length() > 0) {

String xmiStereotype = element.getAttribute("stereotype");

if (sterotypeHashmap.get(xmiStereotype) != null

&& ((String) sterotypeHashmap.get(xmiStereotype))

.equalsIgnoreCase("link")) {


  numberOfLinkR++;
                    }
                }
            }
        }
        nLinkR = String.valueOf(numberOfLinkR);

        return nLinkR;
    }
```

```
    public static String calculateNSubmitR(ClassDiagram classDiagram,
Document document) {
String nSubmitR = "";

int numberOfSubmitR = 0;
HashMap sterotypeHashmap = createSterotypeHashMap(document);
 NodeList elements = document.getElementsByTagName("UML:Association");
for (int i = 0; i < elements.getLength(); i++) {
 Element element = (Element) elements.item(i);
NodeList elementsAssiationEnd = element
 .getElementsByTagName("UML:AssociationEnd");

 boolean isParticipate = isParticipateElement(classDiagram,
  elements, elementsAssiationEnd);
 if (isParticipate) {
 System.out.println(" association sterotype attr : "
 + element.getAttribute("stereotype"));
 if (element.getAttribute("stereotype") != null
&& element.getAttribute("stereotype").length() > 0) {
```

```
String xmiStereotype = element.getAttribute("stereotype");

if (sterotypeHashmap.get(xmiStereotype) != null

&& ((String) sterotypeHashmap.get(xmiStereotype))

.equalsIgnoreCase("submit")) {

numberOfSubmitR++;
                    }
                }
            }
        }

nSubmitR = String.valueOf(numberOfSubmitR);

        return nSubmitR;
    }


public static String calculateNBuildsR(ClassDiagram classDiagram,

Document document) {

 String nBuildsR = "";

int numberOfBuildsR = 0;

HashMap sterotypeHashmap = createSterotypeHashMap(document);

NodeList elements = document.getElementsByTagName("UML:Association");

for (int i = 0; i < elements.getLength(); i++) {

Element element = (Element) elements.item(i);
```

```
 NodeList elementsAssiationEnd = element

.getElementsByTagName("UML:AssociationEnd");


 boolean isParticipate = isParticipateElement
(classDiagram,

elements, elementsAssiationEnd);

 if (isParticipate) {

 System.out.println(" association sterotype attr : "

+ element.getAttribute("stereotype"));

  if (element.getAttribute("stereotype") != null

&& element.getAttribute("stereotype").length() > 0) {

  String xmiStereotype = element.getAttribute("stereotype");

 if (sterotypeHashmap.get(xmiStereotype) != null

&& ((String) sterotypeHashmap.get(xmiStereotype))

.equalsIgnoreCase("Build")) {

numberOfBuildsR++;
                    }
                }
            }
        }

 nBuildsR = String.valueOf(numberOfBuildsR);
```

```
return nBuildsR;
    }


 public static String calculateNClientP
(ClassDiagram classDiagram,

Document document) {

 String nClientP = "";

 int numberOfClientP = 0;

ArrayList listOfClientPages = new ArrayList();

listOfClientPages = classDiagram.getClientpage();

NodeList elements = document.getElementsByTagName("UML:Class");

 for (int i = 0; i < elements.getLength(); i++) {

 Element element = (Element) elements.item(i);

 for (int j = 0; j < listOfClientPages.size(); j++) {

 if ((element.getAttribute("name"))

.equalsIgnoreCase((String) listOfClientPages.get(j))) {

 numberOfClientP++;
                }
            }
        }
        nClientP = String.valueOf(numberOfClientP);
        return nClientP;
```

```
    }

 public static String calculateNForwardR
(ClassDiagram classDiagram,
            Document document) {
        String nForwardR = "";


 int numberOfForwardR = 0;

 HashMap sterotypeHashmap =
 createSterotypeHashMap(document);

NodeList elements =
document.getElementsByTagName("UML:Association");

  for (int i = 0; i < elements.getLength(); i++) {

Element element = (Element) elements.item(i);

NodeList elementsAssiationEnd = element

 .getElementsByTagName("UML:AssociationEnd");


 boolean isParticipate = isParticipateElement(classDiagram,

elements, elementsAssiationEnd);

 if (isParticipate) {

System.out.println(" association sterotype attr : "

+ element.getAttribute("stereotype"));

 if (element.getAttribute("stereotype") != null
```

```
 && element.getAttribute("stereotype").length() > 0) {

String xmiStereotype = element.getAttribute("stereotype");

  if (sterotypeHashmap.get(xmiStereotype) != null

&& ((String) sterotypeHashmap.get(xmiStereotype))

.equalsIgnoreCase("forward")) {

numberOfForwardR++;
                    }
                }
            }
        }
        nForwardR = String.valueOf(numberOfForwardR);

        return nForwardR;
    }

    public static String calculateNIncludeR
(ClassDiagram classDiagram,
  Document document) {
 String nIncludeR = "";
 int numberOfIncludeR = 0;
 HashMap sterotypeHashmap = createSterotypeHashMap(document);
 NodeList elements =
 document.getElementsByTagName("UML:Association");

for (int i = 0; i < elements.getLength(); i++) {

 Element element = (Element) elements.item(i);

 NodeList elementsAssiationEnd = element
```

```
 .getElementsByTagName("UML:AssociationEnd");


 boolean isParticipate = isParticipateElement
(classDiagram,

elements, elementsAssiationEnd);

  if (isParticipate) {

  System.out.println(" association sterotype attr : "

 + element.getAttribute("stereotype"));

  if (element.getAttribute("stereotype") != null

  && element.getAttribute("stereotype").length() > 0) {

 String xmiStereotype = element.getAttribute("stereotype");

if (sterotypeHashmap.get(xmiStereotype) != null

 && ((String) sterotypeHashmap.get(xmiStereotype))

.equalsIgnoreCase("Include")) {

 numberOfIncludeR++;
                    }
                }
            }
        }

 nIncludeR = String.valueOf(numberOfIncludeR);

return nIncludeR;
    }
```

```
 private static boolean isParticipateElement
(ClassDiagram classDiagram,

 NodeList elements, NodeList elementsAssiationEnd) {

boolean isParticipate = false;


for (int k = 0; k < elementsAssiationEnd.getLength(); k++) {

 Element elementAssociationEnd = (Element) elementsAssiationEnd

.item(k);

if (classDiagram.getXmiIDMap().get(

 elementAssociationEnd.getAttribute("participant")) != null) {

isParticipate = true;
            }

        }
        return isParticipate;
    }

    private static boolean isParticipateElementAssoc
(ClassDiagram classDiagram,

  NodeList elements, NodeList elementsAssiationEnd) {

 boolean isParticipate = false;

int count = 0;
```

```
 for (int k = 0; k < elementsAssiationEnd.getLength(); k++) {

 Element elementAssociationEnd = (Element) elementsAssiationEnd

  .item(k);

  if (classDiagram.getXmiIDMap().get(

  elementAssociationEnd.getAttribute("participant")) != null) {

  count++;

  }

            if (count == 2) {
                isParticipate = true;
            }

        }
        return isParticipate;
    }

    private static boolean isAggregationRelationShip
(NodeList elements,

 NodeList elementsAssiationEnd) {

 boolean isAggregate = false;

 for (int k = 0; k < elementsAssiationEnd.getLength(); k++) {

 Element elementAssociationEnd = (Element) elementsAssiationEnd

.item(k);

if (elementAssociationEnd.getAttribute("aggregation") != null
```

```
   && elementAssociationEnd.getAttribute("aggregation")
.equalsIgnoreCase("aggregate")) {
isAggregate = true;
            }

        }
        return isAggregate;
    }

    public static String calculateNUseTagR(ClassDiagram classDiagram,
 Document document) {
String nUseTagR = "";
int numberOfUseTagR = 0;

HashMap sterotypeHashmap = createSterotypeHashMap(document);
NodeList elements = document.getElementsByTagName("UML:Association");
 for (int i = 0; i < elements.getLength(); i++) {
 Element element = (Element) elements.item(i);
NodeList elementsAssiationEnd = element
.getElementsByTagName("UML:AssociationEnd");

 boolean isParticipate = isParticipateElement(classDiagram,
```

```
elements, elementsAssiationEnd);

 if (isParticipate) {

 System.out.println(" association sterotype attr : "

+ element.getAttribute("stereotype"));

if (element.getAttribute("stereotype") != null

&& element.getAttribute("stereotype").length() > 0) {

 String xmiStereotype = element.getAttribute("stereotype");

if (sterotypeHashmap.get(xmiStereotype) != null

&& ((String) sterotypeHashmap.get(xmiStereotype))

.equalsIgnoreCase("useTag")) {

 numberOfUseTagR++;
                    }
                }
            }
        }
        nUseTagR = String.valueOf(numberOfUseTagR);

        return nUseTagR;
    }

    public static String calculateNClientScriptsComp
(ClassDiagram classDiagram,
            Document document) {
        String nClientScriptsComp = "";
```

```
int numberOfClientScriptComp = 0;

  ArrayList listClientScriptPages = new ArrayList();

 listClientScriptPages = classDiagram.getClientscript();

NodeList elements = document.getElementsByTagName("UML:Class");

 for (int i = 0; i < elements.getLength(); i++) {

 Element element = (Element) elements.item(i);

for (int j = 0; j < listClientScriptPages.size(); j++) {

 if ((element.getAttribute("name"))

 .equalsIgnoreCase((String) listClientScriptPages.get(j))) {

 NodeList elementsAttributes = element

.getElementsByTagName("UML:Attribute");

 NodeList elementsOperations = element
                                .
getElementsByTagName("UML:Operation");

numberOfClientScriptComp = numberOfClientScriptComp

+ elementsAttributes.getLength()

 + elementsOperations.getLength();
                }

            }

        }
```

```
nClientScriptsComp = String.valueOf(numberOfClientScriptComp);


return nClientScriptsComp;
    }


public static String calculateNServerScriptsComp
(ClassDiagram classDiagram,

 Document document) {

String nServerScriptsComp = "";


 int numberOfServerScriptsComp = 0;

  ArrayList listServerScriptPages = new ArrayList();

 listServerScriptPages = classDiagram.getServerpage();

NodeList elements = document.getElementsByTagName("UML:Class");

for (int i = 0; i < elements.getLength(); i++) {

 Element element = (Element) elements.item(i);

 for (int j = 0; j < listServerScriptPages.size(); j++) {

if ((element.getAttribute("name"))

.equalsIgnoreCase((String) listServerScriptPages.get(j))) {

 NodeList elementsAttributes = element
```

```
.getElementsByTagName("UML:Attribute");

NodeList elementsOperations = element

.getElementsByTagName("UML:Operation");

numberOfServerScriptsComp = numberOfServerScriptsComp

+ elementsAttributes.getLength()

+ elementsOperations.getLength();
                }

            }

        }


 nServerScriptsComp = String.valueOf(numberOfServerScriptsComp);


return nServerScriptsComp;
    }


public static String calculateWebControlCoupling
(ClassDiagram classDiagram,
            Document document) {

 String webControlCoupling = "";

double webControlCouplingNumber = 0;

double totalNumberOfRelationship = 0;
```

```
 double toatlNumberOfWebPages = 0;

totalNumberOfRelationship = Integer.parseInt(calculateNBuildsR(

classDiagram, document))

+ Integer.parseInt(calculateNForwardR(classDiagram, document))

 + Integer.parseInt(calculateNLinkR(classDiagram, document))

 + Integer.parseInt(calculateNSubmitR(classDiagram, document))

 + Integer.parseInt(calculateNUseTagR(classDiagram, document))

 + Integer.parseInt(calculateNIncludeR(classDiagram, document));


 toatlNumberOfWebPages = Integer.parseInt(calculateNWebP(classDiagram,

 document));

 if (toatlNumberOfWebPages != 0) {

 webControlCouplingNumber = totalNumberOfRelationship

/ toatlNumberOfWebPages;

 } else {
             webControlCouplingNumber = 0;
         }

 webControlCoupling = String.valueOf(webControlCouplingNumber);
         return webControlCoupling;
     }

     public static String calculateWebDataCoupling
```

```
(ClassDiagram classDiagram,
            Document document) {
        String webDataCoupling = "";

        double webDataCouplingNumber = 0;
        double totalNumberOfFormElements = 0;
        double totalNumberOfServerPages = 0;
        totalNumberOfFormElements = Integer.parseInt(calculateNFormE(
                classDiagram, document));

        totalNumberOfServerPages = Integer.parseInt(calculateNserverP(
                classDiagram, document));
        if (totalNumberOfServerPages != 0) {
            webDataCouplingNumber = totalNumberOfFormElements
                    / totalNumberOfServerPages;
        } else {
            webDataCouplingNumber = 0;
        }
        webDataCoupling = String.valueOf(webDataCouplingNumber);

        return webDataCoupling;
    }

    public static String calculateWebReusability
(ClassDiagram classDiagram,
            Document document) {
        String webReusability = "";

        double webReusabilityNumber = 0;
        double totalNumberOfIncludeRelationship = 0;
        double totalNumberOfWebPages = 0;
        totalNumberOfIncludeRelationship =
 Integer.parseInt(calculateNIncludeR(
                classDiagram, document));

        totalNumberOfWebPages =
```

```
Integer.parseInt(calculateNWebP(classDiagram,
                    document));
            if (totalNumberOfWebPages != 0) {
                webReusabilityNumber =
totalNumberOfIncludeRelationship
                        / totalNumberOfWebPages;
            } else {
                webReusabilityNumber = 0;
            }
            webReusability =
String.valueOf(webReusabilityNumber);

            return webReusability;
        }

        public static String calculateNC
(ClassDiagram classDiagram,
                Document document) {
            String nC = "";
            int numberOfClasses = 0;
            ArrayList listOfIntefaceClasses = new ArrayList();
            listOfIntefaceClasses =
classDiagram.getInterfaceclass();
            NodeList elements =
document.getElementsByTagName("UML:Class");

for (int i = 0; i < elements.getLength(); i++) {
                Element element =
(Element) elements.item(i);
for (int j = 0; j < listOfIntefaceClasses.size(); j++) {
 if ((element.getAttribute("name"))
  .equalsIgnoreCase((String) listOfIntefaceClasses.get(j))) {
  numberOfClasses++;
                    }
                }
            }
```

```
        nC = String.valueOf(numberOfClasses);
        return nC;
    }

    public static String calculateNA
(ClassDiagram classDiagram,
            Document document) {
        String nA = "";

        int numberOfAttributes = 0;
        ArrayList listClasses = new ArrayList();
        listClasses =
classDiagram.getInterfaceclass();
        NodeList elements =
document.getElementsByTagName("UML:Class");

for (int i = 0; i < elements.getLength(); i++) {
            Element element =
 (Element) elements.item(i);

for (int j = 0; j < listClasses.size(); j++) {

 if ((element.getAttribute("name"))
                        .
equalsIgnoreCase((String) listClasses.get(j))) {

NodeList elementsAttributes = element

.getElementsByTagName("UML:Attribute");

 numberOfAttributes = numberOfAttributes

+ elementsAttributes.getLength();
                }

            }
```

```
        }

        nA = String.valueOf(numberOfAttributes);

        return nA;
    }


 public static String calculateNM
(ClassDiagram classDiagram,

 Document document) {
        String nM = "";


 int numberOfMethods = 0;

 ArrayList listClasses = new ArrayList();

 listClasses = classDiagram.getInterfaceclass();

 NodeList elements = document.getElementsByTagName("UML:Class");

 for (int i = 0; i < elements.getLength(); i++) {

  Element element = (Element) elements.item(i);

for (int j = 0; j < listClasses.size(); j++) {

 if ((element.getAttribute("name"))

 .equalsIgnoreCase((String) listClasses.get(j))) {

NodeList elementsMethods = element
```

```
 .getElementsByTagName("UML:Operation");

umberOfMethods = numberOfMethods

+ elementsMethods.getLength();
                }

            }

        }

        nM = String.valueOf(numberOfMethods);

        return nM;
    }

    public static String calculateNAssoc
(ClassDiagram classDiagram,
            Document document) {
  String nAssoc = "";

 int numberOfAssocR = 0;
NodeList elements = document.getElementsByTagName("UML:Association");
 for (int i = 0; i < elements.getLength(); i++) {
Element element = (Element) elements.item(i);
  NodeList elementsAssiationEnd = element
   .getElementsByTagName("UML:AssociationEnd");

   boolean isParticipate = isParticipateElementAssoc(classDiagram,
   elements, elementsAssiationEnd);
  if (isParticipate) {
   System.out.println(" association sterotype attr : "
    + element.getAttribute("stereotype"));
   if (element.getAttribute("stereotype") == null
   || element.getAttribute("stereotype").length() == 0) {
  if (!isAggregationRelationShip(elements,
```

```
 elementsAssiationEnd)) {
numberOfAssocR++;
                    }
                }
            }
        }
        nAssoc = String.valueOf(numberOfAssocR);

        return nAssoc;
    }
public static String calculateNAgg(ClassDiagram classDiagram,
            Document document) {
String nAgg = "";

  int numberOfAggR = 0;
   NodeList elements =
document.getElementsByTagName("UML:Association");
for (int i = 0; i < elements.getLength(); i++) {
  Element element = (Element) elements.item(i);
  NodeList elementsAssiationEnd = element
 .getElementsByTagName("UML:AssociationEnd");

   boolean isParticipate = isParticipateElementAssoc(classDiagram,
  elements, elementsAssiationEnd);
 if (isParticipate) {
  System.out.println(" association sterotype attr : "
  + element.getAttribute("stereotype"));
  if (element.getAttribute("stereotype") == null
   || element.getAttribute("stereotype").length() == 0) {
  if (isAggregationRelationShip(elements,
                        elementsAssiationEnd)) {
                    numberOfAggR++;
                }
            }
        }
```

```
        nAgg = String.valueOf(numberOfAggR);

        return nAgg;
    }

  public static String calculateCoupEntropy
(ClassDiagram classDiagram,
Document document, MetricEntry mEntry,
 double totalNumberOfElements)
{
        String coupEntropy = "";

 // double totalNumberOfElements =
classDiagram.getClientpage().size() +


double totalNumberOfRelationShips =
Double.parseDouble(mEntry.getNagg())
 + Double.parseDouble(mEntry.getNassoc())
  + Double.parseDouble(mEntry.getNbuildsr())
  + Double.parseDouble(mEntry.getNforwardr())
  + Double.parseDouble(mEntry.getNincluder())
 + Double.parseDouble(mEntry.getNlinkr())
  + Double.parseDouble(mEntry.getNsubmitr())
  + Double.parseDouble(mEntry.getNuseTagr());
        System.out.println("totalNumberOfRelationShips = "

+ totalNumberOfRelationShips);

 double result = 1.0 / totalNumberOfElements

 * (-Math.log10(1 / (1 + totalNumberOfRelationShips)));

 coupEntropy = String.valueOf(result);

  return coupEntropy;
```

```
    }

    public static String calculateCohesionEntropy
(MetricEntry mEntry,
            double totalCoupling) {

String cohesionEntropy = "";

double moduleCoupling =
Double.parseDouble(mEntry.getCoupentropy());
        double cohesion = totalCoupling / moduleCoupling;
        cohesionEntropy = String.valueOf(cohesion);
        return cohesionEntropy;
    }

public static HashMap createSterotypeHashMap(Document document) {
  HashMap sterotypeHashmap = new HashMap();
   NodeList elements = document.getElementsByTagName("UML:Stereotype");
  for (int i = 0; i < elements.getLength(); i++) {
  Element element = (Element) elements.item(i);
 sterotypeHashmap.put(element.getAttribute("xmi.id"), element
 .getAttribute("name"));
        }
        return sterotypeHashmap;

    }

}
```