Web-based tool for software architecture metrics

Israr Pasha Shaik
West Virginia University

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Recommended Citation
https://researchrepository.wvu.edu/etd/1721

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.
Web-Based Tool for Software Architecture Metrics

Israr Pasha Shaik

Thesis submitted to the College of Engineering and Mineral Resources at West Virginia University in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering

Hany H Ammar, Ph.D., Chair
Katerina Goseva Popstanjanova, Ph.D.
Ali Mili, Ph.D.(CCS, NJIT)

Lane Department of Computer Science and Electrical Engineering
Morgantown, West Virginia
2006


© Israr Pasha Shaik
Abstract

Web-Based Tool for Software Architecture Metrics

Israr Pasha Shaik

The study of software architectures is gaining importance due to its role in various aspects of software engineering like product line engineering, component based software engineering and other emerging paradigms. Various architectural attributes including error propagation, change propagation, requirements propagation provide a wealth of information about software architectures.

For the past few years, researchers at West Virginia University and New Jersey Institute of Technology have been working on the project, Software Architectural level metrics. As the result of which, various software metrics have been proposed to analyze and evaluate the design qualities of software architectures. The goal of this research is to develop a web based environment for Software Architecture Metrics.

Various algorithms for computing Error propagation, change propagation and size of change on UMLRT models have been designed and implemented as part of this research work. These algorithms have been integrated into the web based tool. In this research, we also show that change propagation probability (CP) is helpful and effective in assessing the design quality of software architectures. We experiment with two different architectures (one that employs design patterns versus one that does not) for the same application. We also analyze and compare change propagation metric with respect to other coupling-based metrics.
Acknowledgements

I would like to thank Dr. Hany H Ammar for assigning me to work on the funded projects by NASA in Software Engineering, which helped me to finance all of my tuition fees and earn my living. His guidance and advises throughout this project have helped me to finish my thesis in time.

I am very thankful to my research team members Walid and Rajesh. Initial help from Rajesh when I started my research work and the never-ending support from Walid are really praise worthy. I would also like to thank my friends Madhukar, Satya Srinivas, Kiran and Satyadev for their invaluable help and support.

Finally I like to thank my committee members for their time and suggestions which helped to improve my thesis.
Dedication

I would like to dedicate this thesis to my parents and sisters who have always encouraged and stood by me throughout my life. I would like to express my gratitude to my parents for their hard work and continuous support, which helped me in pursuing higher studies and fulfilling my desire.
# Table of Contents

Abstract ............................................................................................................................... ii  
Acknowledgements ........................................................................................................... iii  
Dedication ........................................................................................................................... iv  
Table of Contents ............................................................................................................. v  
List of Figures and Illustrations ....................................................................................... viii  

CHAPTER ONE: .................................................................................................................1  
INTRODUCTION ...............................................................................................................1  
1.1 Problem Statement .....................................................................................................1  
1.2 Research Objectives ..................................................................................................1  
1.3 Research Tasks ..........................................................................................................2  
1.4 Organization of thesis ...............................................................................................3  

CHAPTER TWO: ................................................................................................................5  
BACKGROUND .................................................................................................................5  
2.1 Software metrics ........................................................................................................5  
2.1.1 Object Oriented (OO) metrics ..............................................................................5  
2.1.2 Error propagation probability metric (EP) .............................................................6  
2.1.3 Change propagation probability metric (CP) .........................................................9  
2.1.4 Size of Change metric (SC) .................................................................................11  
2.2 Design Patterns ........................................................................................................12  
2.3 Technologies used in developing web based tool .....................................................12  
2.3.1 J2EE Framework ................................................................................................12  
2.3.2 Web Services .....................................................................................................13  
2.3.3 Web Service Description Language (WSDL) .......................................................14  
2.3.4 Enterprise JavaBeans Technology (EJB) .............................................................14  
2.3.5 Java Server Pages Technology (JSP) .................................................................14  
2.3.6 Sun Java System Application Server ..................................................................15  
2.3.7 Java Native Interface (JNI) ................................................................................15  

CHAPTER THREE: ............................................................................................................16  
ARCHITECTURE OF SOFTWARE ARCHITECTURE METRICS TOOL (SAMT) ....16  
3.1 Functional description ..............................................................................................16  
3.1.1 Perform static Analysis .......................................................................................17  
3.1.2 Retrieve Analysis Information ............................................................................18  
3.1.3 Estimate Error propagation probability .............................................................19  
3.1.4 Estimate Change propagation probability .........................................................27  
3.1.5 Estimate size of change .....................................................................................31  
3.2 Structural description of SAMT ..............................................................................32  
3.2.1 Metamodel layer ................................................................................................32  
3.2.2 Metrics layer .....................................................................................................34  
3.2.3 User interface layer ...........................................................................................34
List of Tables

Table 1. Error propagation matrix of pacemaker ............................................................. 26
Table 2. Change propagation matrix of pacemaker .......................................................... 30
Table 3. Size of Change matrix of pacemaker ................................................................. 32
List of Figures and Illustrations

Figure 1. Use case diagram of SAMT tool ................................................................. 18
Figure 2. SAMT Metamodel....................................................................................... 19
Figure 3. A simple UMLRT model............................................................................ 21
Figure 4. A figure depicting capsule’s state change ................................................. 21
Figure 5. Top-level software architecture of the system......................................... 22
Figure 6. The architecture of the pacemaker subsystem........................................ 23
Figure 7. A sample of a message protocol (Between two components)................. 24
Figure 8. State diagram of CommunicationGnome component ............................ 24
Figure 9. A substate diagram of component CommunicationGnome....................... 25
Figure 10. Message passing between two capsules ................................................. 28
Figure 11. Protocol role in message passing......................................................... 29
Figure 12. Protocol specification between components 1 and 3............................ 31
Figure 13. SAMT Architecture.............................................................................. 33
Figure 14. Structure of the web based tool ............................................................. 37
Figure 15. Two tier client/server architecture with J2EE technology .................... 38
Figure 16. Sequence diagram for uploading files for UMLRT Module................... 39
Figure 17. Sequence diagram for uploading files for Java Understand Module ....... 40
Figure 18. Sequence diagram for computing EP on UMLRT models....................... 42
Figure 19. Sequence diagram for computing CP and SC on UMLRT models......... 42
Figure 20. Sequence diagram for computing CP and SC on Java based Applications.. 43
Figure 21. Sequence diagram for displaying the metrics to the user....................... 44
Figure 22. Class diagram of Job Application before applying strategy pattern.......... 48
Figure 23. Class diagram of Job Application after applying strategy pattern ................. 49
Figure 24. Change propagation of Job Application before applying strategy ............... 50
Figure 25. Change propagation of Job Application after applying strategy ................. 50
Figure 26. Weighted Methods per Class and McCabe Cyclomatic Complexity for Job Application case study ........................................................................................................ 51
Figure 27. Class diagram of an initial design ................................................................ 53
Figure 28. Class diagram of a design that uses mediator design pattern ....................... 53
Figure 29. CP for the simple design on case study Colleague States ............................ 54
Figure 30. CP for the case study on architecture employing mediator pattern ............... 54
Figure 31. Weighted Methods per Class and McCabe Cyclomatic Complexity for Colleague States case study ........................................................................................................ 55
Figure 32. CBO for the case studies on Colleague States and Job Application ............. 57
Figure 33. RFC for the case studies Colleague States and Job Application .................. 57
Figure 34. MPC for the case studies on Job Application and Colleague States ............ 58
Figure 35. Screen shot of CP values for pacemaker case study ................................... 66
Figure 36. Screen shot of CP values for game of life case study ................................... 67
Figure 37. Screen shot of CP values for callbacks case study ...................................... 68
Figure 38. Screen shot of CP values for IsrExample case study ................................... 69
Figure 39. Screen shot of CP values for DynamicForwarding case study .................... 70
Figure 40. Screen shot of CP values for ObserverPattern case study .......................... 71
CHAPTER ONE:

INTRODUCTION

The study of software architectures is gaining importance due to its role in various aspects of software engineering like product line engineering; component-based software engineering and other emerging paradigms. Various architectural metrics attributes like error propagation, change propagation, requirements propagation were proposed to evaluate software architectures [12].

This work is part of a larger effort to analyze and validate various architectural attributes like error propagation [2], change propagation [3], requirements propagation [17], diagonality [12], product line properties [18], etc.

1.1 Problem Statement

The following problems were identified

[1]. There was no end to end tool for a user to compute metrics on various models using the metrics proposed and defined in the project Software Architectural Level metrics.

[2]. To assess the design quality of software architectures and to show that change propagation probability (CP) complements the information and provides a different perspective than the one provided by the traditional object-oriented metrics.

1.2 Research Objectives

This research work can be thought of as divided into three main objectives,

[1]. To design, develop and implement a web based tool for the project Software Architectural Level Metrics.
The second objective is to implement algorithms to estimate various metrics like error propagation, change propagation and size of change metrics on both Java based applications and UMLRT models.

The third objective is to work with the specific architectural attribute change propagation (CP). The main aim is to show the role and use of CP in assessing the design quality (when we say design quality, we mean the terms like extensibility, maintainability, reusability etc.,) of software architectures.

1.3 Research Tasks

The research tasks for the research work done as part of thesis are as follows

- Building a client-server web based tool for the project Software Architectural Level Metrics. An end to end tool which helps the clients to measure and estimate various metrics using the server at West Virginia University.
- The web based tool to be integrated with a third party tool Espress Chart for the client to view results in bar chart format.
- Estimate Size of change metric on architecture information extracted from Java based applications.
- Estimate Error propagation, Change propagation and Size of change for architectures defined using Unified Modeling Language Real Time (UMLRT) models.
- Assess the design quality of software architectures using the change propagation metric.
• Compare change propagation metric with other coupling based object oriented metrics and show that change propagation metric provides a different perspective and complements the usage of other object oriented metrics.

1.4 Organization of thesis

The research work done as part of thesis is organized as follows

Chapter 1: This chapter gives an overview of Software Architectural Level Metrics project and the purpose of the research work done.

Chapter 2: Chapter 2 contains the background and definitions of various object oriented metrics and describes various J2EE technologies used in this thesis like JSP, JNI, EJB, and WSDL. This chapter also presents an overview of concepts such as design patterns, web services and Frameworks.

Chapter 3: This chapter describes the architecture of the Software Architecture Metrics Tool (SAMT) and both functional and structural descriptions of the SAMT are also provided. The methodology and procedure followed in estimating the metrics like error propagation, change propagation and size of change is also provided in this chapter.

Chapter 4: This chapter provides the introduction of the web based tool and design and implementation of the tool SAMT with the help of sequence and class diagrams. This chapter also explains all the modules in SAMT and their behaviour.

Chapter 5: This chapter presents the research done in assessing the design quality of software architectures using the metric, change propagation. Change propagation metric is compared with other coupling-based object oriented metrics.

Chapter 6: This chapter presents the testing done on the web based tool. Various problems faced during the testing of the tool have been described and the methods
adopted to solve them have also been given in this chapter. Alternative ways have been suggested to overcome some of the problems for which solutions weren’t found.

**Chapter 7**: This chapter presents the conclusion and suggests future work to be done on web based tool SAMT.
CHAPTER TWO: BACKGROUND

This chapter gives the background and definitions of the various concepts that have been used in this research. This chapter also provides a basic methodology and rationale that has been followed throughout this research work.

2.1 Software metrics

Software metrics play a vital role in determining the design quality of software architectures. Generally these metrics are useful in every phase of a software development life cycle. These reflect various design aspects like complexity, maintainability, reusability, flexibility etc about the components of an architecture.

2.1.1 Object Oriented (OO) metrics

Traditional metrics like lines of code, cyclomatic complexity, comment ratio weren’t sufficient to provide architectural information required by an analyst. So, various OO metrics like Coupling Between Objects (CBO), Response For a Class (RFC), Weighted Methods per Class (WMC), Message Passing Coupling (MPC), McCabe Cyclomatic Complexity (MCC) have been proposed to validate software architectures.

Coupling Between Objects (CBO)

CBO is the count of the number of components coupled to a particular component. A component is said to be coupled to another component if it either uses the methods or attributes of another component or both.

Response For a Class (RFC)

Response For a Class (RFC) is the sum of all the methods that can be invoked in response to a message to an object of a component or by a method of the component. It is given as
RFC = |RS|

= \{M\} \cup \{R_i\}

Where \{M\} is the set of methods of a component and \{R_i\} is the set of methods called by method i and RS is the response set.

**Weighted Methods per Class (WMC)**

WMC is the sum of the complexities of all the methods in a component.

\[
WMC = \sum_{i=1}^{n} C_i
\]

where \(C_1, C_2, \ldots, C_n\) are the complexities of the methods \(M_1, M_2, \ldots, M_n\) respectively.

**McCabe Cyclomatic Complexity (MCC)**

MCC gives the number of independent paths in a flow graph.

\[
MCC = E - N + 2P
\]

Where \(E\) is the number of edges, \(N\) is the number of nodes and \(P\) is the number of flow graphs.

**Message Passing Coupling (MPC)**

MPC is defined as the number of method calls made by all the methods in a component.

**2.1.2 Error propagation probability metric (EP)**

In this section, the definition of error propagation followed by a few equations to estimate error propagation is introduced. Let us consider a software architecture modeled by components and connectors. Error propagation probability (EP) for an architecture is the conditional probability that an error originating in one component of the architecture is
propagated by another component. The estimation of the elements error propagation EP is based on the following definition [2]:

**Definition.** Let A, B be two components of an architecture and let X be the connector that carries information from A to B. The error propagation probability from component A to component B is denoted by EP(A,B) and defined as the following conditional probability

\[
EP(A,B) = \Pr([B](x)) \neq [B](x') | x \neq x',
\]

Where \([B]\) denotes the function of component B and \(x\) is an element of the connector \(x\) from A to B. By extending this definition, we can say EP(A,A) is equal to 1, which is the probability that an error in A causes an error in A. We let EP(A,B) be a NxN matrix such that the entry at row A and column B is the error propagation probability from A to B.

**Estimating Error Propagation**

Analytically, the error propagation probability can be expressed in terms of the probabilities of the individual A to B messages and states, using the following formula [2].

\[
EP(A \rightarrow B) = \frac{1 - \sum_{x \in S_B} P_B(x) \sum_{y \in S_B} P_{A \rightarrow B}[F_x^{-1}(y)]^2}{1 - \sum_{v \in V_{A \rightarrow B}} P_{A \rightarrow B}[v]^2}
\]

Where,

- \(P_B(x)\) is the probability of observing component B in state \(x\).
- \(P_{A \rightarrow B}[F_x^{-1}(y)]^2\) is the probability of transmission of a message(from component A to component B) that causes component B to transit from state \(x\) to state \(y\).
\( P_{A \rightarrow B}[v] \) is the probability that message \( v \) is sent over the connector from component A to component B.

\( F_{x}^{-1}(y) = \{ v \in V_{A \rightarrow B} / F_{x}(v) = y \} \) denotes a transit of state, where \( x \) and \( y \) are the states and \( v \) is the message causing the state transit.

We assume a probability distribution \( P_{B} \) on the set of states \( S_{B} \) of component B and a probability distribution \( P_{A \rightarrow B} \) on the set of messages \( V_{A \rightarrow B} \) passed from A to B. On assuming that the states of B and the messages passing through the connector from A to B are equiprobable, then the above equation is reduced as shown below [2],

\[
\text{EP}(A \rightarrow B) = \frac{1 - \frac{1}{S_{B} \| V_{A \rightarrow B} \|^{2}} \sum_{x \in S_{x}} \sum_{y \in S_{y}} | F_{x}^{-1}(y) |^{2}}{1 - \frac{1}{| V_{A \rightarrow B} |}}
\]

Where,

\( \text{EP}(A \rightarrow B) \) is the error propagation from component A to component B.

\( S_{B} \) is the total number of states of component B.

\( V_{A \rightarrow B} \) is the total number of messages passing from component A to component B through a connector.

\( F_{x}^{-1}(y) \) denotes a transit of state of component B from state \( x \) to state \( y \) due to a message \( v \) transmitted from component A to component B

Once, the \( N \times N \) matrix of error propagation probabilities is computed for an architecture with \( N \) components the following conclusions can be drawn.
i) If the row A has high values, then it is likely that an error in A can propagate widely to other components in the architecture. So, one has to take steps to minimize faults in A at design or development time so as to minimize errors at run time.

ii) If the column B has high values, then it is likely that B gets easily affected by errors in other components in the architecture. So, one has to take steps to immunize B against errors, e.g., by providing it with fault tolerance capabilities like error detection, error recovery, damage assessment and others.

iii) An idealistic conditional error propagation matrix is one where only the diagonal cells are 1s and the rest of the cells are 0s, i.e., an identity matrix. In such a case, components don’t propagate errors to other components. At the other extreme, the worst possible matrix is one for which all cells of the conditional error propagation matrix are all 1s. In this case, when ever an error occurs in a component, it propagates to all other components.

2.1.3 Change propagation probability metric (CP)

In this section, the definition of change propagation followed by a few equations to estimate change propagation is introduced. Let us consider a software architecture modeled by components and connectors. We are interested in the maintainability of the products instantiated from it. Change propagation probability $CP = [cp_{ij}]$ for an architecture is the conditional probability that a change originating in one component of the architecture requires changes to be made to other components. The estimation of the elements $cp_{ij}$ of the change propagation matrix CP is based on the following definition [3]:
**Definition.** Given components $C_i$ and $C_j$ of a system $S$, the *change propagation probability* from $C_i$ to $C_j$ is denoted by $cp_{ij}$ and defined as the following conditional probability

$$cp_{ij} = \Pr((\text{[}C_j\text{]} \neq \text{[}C'_j\text{]}) | (\text{[}C_i\text{]} \neq \text{[}C'_i\text{]})) \land ([S] = [S'])$$

(1)

where $[X]$ denotes the functionality of component/system $X$ and $S'$ is the system obtained from $S$ by changing $C_i$ into $C_i'$ (and possibly $C_j$ into $C_j'$ as a consequence).

An architecture can be seen as a collection of components $C_i$, $i=1,…,N$. With every component $C_i$, we associate the set $V_i$ of the interface elements of the provided functions of $C_i$. We determine the *usage coefficient* value $\pi_{ij}^\nu$ for every interface element $\nu \in V_i$ and every other component $C_j$, $j \neq i$. They take binary values:

- $\pi_{ij}^\nu = 1$, if the interface element $\nu$ provided by $C_i$ is required by $C_j$. This means that any signature change in component $C_i$ associated with interface element $\nu$ will propagate to component $C_j$.
- $\pi_{ij}^\nu = 0$, otherwise.

In [3], AbdelMoez et al proposed the following for estimating the change propagation probability $cp_{ij}$, for every pair of components $C_i$ and $C_j$, $i \neq j$, based on the values of the *usage coefficients* $\pi_{ij}^\nu$ by:

$$cp_{ij} = \frac{1}{|V_i|} \sum_{\nu \in V_i} \pi_{ij}^\nu$$

**Change propagation coefficient (CPC)**
Since matrices are hard to compare, we introduce Change Propagation Coefficient (CPC) that attempts to capture the information of a matrix in a single scalar, doubtless with much loss of information. CPC is a scalar that reflects the potential of an architecture to insulate its components from each other’s changes. Obviously, the idealistic change propagation coefficient corresponds to a conditional change propagation matrix where only the diagonal cells are 1s and the rest of the cells in the matrix are 0s, i.e. an identity matrix. In such a case, components don’t propagate changes to other components. At the other extreme, the worst possible CPC is one for which all cells of the conditional change propagation matrix are 1s. In this case, whenever a change arises in a component, it propagates to all other components. We wish to define the change propagation coefficient in such a way as to reflect, using a value between 0 and 1, how close we are to the idealistic matrix and how far we are from the worst possible matrix. We estimate the change propagation coefficient by:

\[
CPC = \frac{\sum_{i} \sum_{j \neq i} c_{pij}}{N^2 - N}, \tag{3}
\]

where \( N \) is the number of components in the architecture. A low CPC value indicates an architecture where changes do not easily propagate between its components.

2.1.4 Size of Change metric (SC)

Size of Change, SC \((C_i, C_j)\) is defined as the ratio of the number of affected methods of component \(C_j\) due to the changes in the interface of component \(C_i\) to the number of methods \(M_i\) in the receiving component \(C_j\).

\[
SC(C_i, C_j) = \frac{1}{|M_j|} \sum_{\nu \in V_i} \phi_{\nu j}
\]
Where \( \sum_{v \in V_i} \phi_{v_j} \) is the number of methods in component \( C_j \) that got affected by the number of changes in the variables \( v \in V_i \) in the interface of the component \( C_i \).

### 2.2 Design Patterns

One of the objectives of this thesis is to assess the design quality of software architectures. To show that CP complements the usage of other object oriented metrics, two different architectures (one that employs patterns versus one that does not) for the same application are studied.

A design pattern [8] is a solution to a general design problem that occurs in a context. Since the idea was first communicated, design patterns have been considered an important aspect in the design of software architectures. The practice of ad-hoc software design is slowly shifting towards pattern-oriented development [16].

Designers can try to identify the various aspects of design problems associated with the application, and then they can choose some of the successful design patterns from a pattern repository to solve the problems. There are many advantages of using the various design patterns with respect to software architectures. Some of these advantages include: improving reusability, extensibility, and maintainability of software architecture.

### 2.3 Technologies used in developing web based tool

Following sections illustrate the J2EE technologies used in developing the web based tool, software architecture metrics.

#### 2.3.1 J2EE Framework

The Java 2 Platform, Enterprise Edition (J2EE) [24] is Sun Microsystems distributed computing framework. It consists of the following
• J2EE platform: A standard platform for hosting the J2EE applications.
• Reference Implementation: An application server that supports the latest specification.
• Compatibility Test Suite: A utility that verifies that an application server is compatible with J2EE. This helps to provide the benefit of write once and run anywhere functionality in the J2EE platform.
• Application Programming Model (APM) Blueprint: A reference provided to teach individuals, so that they can develop their own distributed applications using J2EE.

2.3.2 Web Services

Web services [19] are web-based applications that use XML based standards and transport protocols to interact with the calling clients. The basic purpose of using web services is to make a programming model available to any person in the world through the internet. He does not need to know the complicated stuff that runs in the server side. It saves lot of time for a client by making him independent of being aware of a number of technologies. Such a service is thus really helpful for a novice user. It saves a lot of time for the client, he does not have to download the code and go through various manuals to install the software and then worry about any bugs that might be present in the downloaded stuff. Web services offer flexibility to developers, for example one web service can call another web service. Web services are easy to access, it can have a dedicated access to a single client or it can be accessed by anyone in the world using browsers, wireless devices and others.
2.3.3 **Web Service Description Language (WSDL)**

WSDL [19] is a standardized XML format for describing web services. It contains the name of the service, the port information, the locations of the service and the ways through which it can be accessed. It also specifies the messages exchanged by the service, operations supported by the service and the endpoints of the service.

2.3.4 **Enterprise JavaBeans Technology (EJB)**

Enterprise JavaBeans (EJB) [24] components run in the EJB container. They are the core of the J2EE application and provide functions like transaction management, state management, resource pooling and security checks. They should be able to support various API’s like Java 2 Standard Edition (J2SE), Java Message Server (JMS), Java Database Connectivity (JDBC) and many others. It contains business logic that operates on enterprise data, mediates client access, contains metadata such as transaction and security attributes separate from the bean and it is maintained and managed by a container.

2.3.5 **Java Server Pages Technology (JSP)**

The Java Server Pages technology [21] provides a simplified, fast way to create web pages that display dynamically generated content. JSP technology was designed to make it easier and faster to build web-based applications that work with wide variety of web servers, application servers, browsers and other development tools. The advantages of using a JSP technology are, it is compatible to work with any web or application server, separates the application logic from the appearance of the page. It also allows fast development, testing and simplifies the process of developing interactive web-based applications.
2.3.6 Sun Java System Application Server

The application server can be downloaded at [20]. This application server supports various J2EE technologies and tools for the developers. More information about the application server can be obtained at [19].

2.3.7 Java Native Interface (JNI)

Java Native Interface (JNI) [22] is used with native methods. A native method is a Java method whose implementation is written in other programming languages. JNI thus helps the Java code to operate with other applications and libraries written in languages like C, C++ and others. This helps code to be portable across a variety of platforms.
CHAPTER THREE:

ARCHITECTURE OF SOFTWARE ARCHITECTURE METRICS TOOL (SAMT)

Software architecture metrics tool (SAMT) is a web-based tool that is used to generate various architecture metrics like error propagation probability, change propagation probability and size of change. In the following sections both the tool’s functional and structural description are presented.

3.1 Functional description

The SAMT supports currently two types of applications. The applications with source code in Java and UMLRT models. We have developed a case tool that supports these applications. The tool is web-based which gives three types of metrics error propagation probability, change propagation probability and size of change on the UMLRT models and change propagation probability, size of change on applications with source code in Java. We have implemented the tool as a web service using Java and J2EE. Consider the use-case diagram shown in figure 1.

The various actors depicted in the figure 1 are

[1]. Analyst: Analyst is a user who uses the software architecture metrics tool to compute metrics like error propagation, change propagation and size of change. He needs also to upload the architecture description files to the server.

[2]. Repository: This is a local disk at the server side, where the files uploaded by the analyst and the results to be displayed to the analyst are stored.
[3]. Architecture description files of UMLRT models is the aggregation of the actors Capsules and EPInput, which contains artifacts of a UMLRT model like components, connectors, message transitions between the components etc.

[4]. Architecture description files of Java source code is the aggregation of the actors like Classes, functions, variables and sharp, which contains the information of the architecture like the list of components, list of functions, variables and the message passing between the components respectively.

3.1.1 Perform static Analysis

The first step in generating the metrics is to analyze statically the various models like UMLRT models and applications with source code in Java at the client side. For this two static analysis tools are used, one is Understand for Java and the other is Rational rose real time. Script files are written to generate various outputs which will be later used as inputs for SAMT. The outputs of the script files are the architecture artifacts such as the list of components of the system, attributes and their methods. These script files also provide various functional dependencies among the system components.
3.1.2 Retrieve Analysis Information

The outputs of the various script files are then uploaded to the server. The tool is then used to parse these outputs to obtain a metamodel which consists of components, data-types and connectors as shown in figure 2 in a centralized repository. The centralized repository is a local disk at the server side.
3.1.3 Estimate Error propagation probability

The error propagation probability is estimated from the artifacts of the repository by applying the equation,

$$EP(A \rightarrow B) = 1 - \frac{1}{|V_{A \rightarrow B}|} \sum_{x \in S_A} \sum_{y \in S_B} |F^{-1}_x(y)|^2$$

(1)

Where the states of B and the messages that pass from A to B through the connector are considered to be equi-probable.

Now consider a case study-pacemaker device [25]. A pacemaker is a life critical real-time application. An error in the software operation of the device can cost a patient’s life. Therefore it is necessary to design an executable model of the system which incorporates the timing and deadline constraints. We use such executable models that are provided
along with the static analysis tool Rational rose real time to validate various metrics like error propagation, change propagation and size of change metrics. A cardiac pacemaker is an implanted device for patients with pathologies related to low heart beats. It runs in either a programming mode or in one of the various operational modes. The discussion is limited pertaining to the work that has been done in this research. More information about this case study can be obtained at [25].

An executable UML rose real time (UMLRT) model consists of components which are otherwise known as capsules, ports, protocols and connectors. Two capsules communicate with each other by exchanging messages through the ports. A connector connects any two capsules by connecting a port of one capsule with the port of another as shown in the figure 3. Further each capsule has its state machine. These state machines process various messages that are received. Each message has a signal that determines the transition that gets triggered and thus a transition of a state occurs for a capsule from the current to a state that matches with the message received, this is shown in the figure 4. More information of these executable models can be obtained from the rose tutorials by downloading the tool at [13].
The system has a hierarchical architecture. The top level software architecture of the system in shown in the figure 5, followed by the software architecture of the pacemaker subsystem in the figure 6. Using these artifacts one can identify the components (otherwise known as capsules) and connectors of the component based systems. These components names can then be used to name the rows and columns of an EP matrix.
Figure 5. Top-level software architecture of the system
Figure 7 shows a message protocol coilcomm between a pair components CoilDriver and CommunicationGnome. This artifact provides the message sets \( V_{\text{CoilDriver} \rightarrow \text{CommunicationGnome}} (V_{A \rightarrow B}) \) and \( V_{\text{CommunicationGnome} \rightarrow \text{CoilDriver}} (V_{B \rightarrow A}) \) that are going between the two components. Similarly, one can get all such message sets between each pair of components in the system using the Rose-RT tool with the help of the script files in summit basic. Figures 8 and 9 show the set of states of the component CommunicationGnome, where the figure 9 is the substate diagram of the state Enabled in figure 8. This gives us the number of states \( S_B \) of the component CommunicationGnome. Similarly, one can get all the state sets of the components in the system using the summit basic script files with the help of the static analysis tool Rose-RT.
Figure 7. A sample of a message protocol (Between two components).

Figure 8. State diagram of CommunicationGnome component
Considering the system in figure 6 and on executing the script file on the UMLRT model, one can get the set of states $S_B$ and messages $V_{A\rightarrow B}$ from the artifacts of the system specification. By using the equation (1), the error propagation probability can be obtained for the system(Table 1).
Table 1. Error propagation matrix of pacemaker

To illustrate the computing of the error propagation matrix in detail, the computation of the error propagation from component CoilDriver to CommunicationGnome $\text{EP}_{\text{CoilDriver} \rightarrow \text{CommunicationGnome}}$ ($\text{EP}_{2 \rightarrow 3}$) is shown in the following steps:

i) The set of states $S_3$ of component CommunicationGnome is obtained from figures 8 and 9 as $\{\text{Disabled}, \text{Transmit}_\text{Bytes}, \text{Idle}, \text{Receive}_\text{Byte}\}$ and the message set $V_{\text{CoilDriver} \rightarrow \text{CommunicationGnome}}$ ($V_{2 \rightarrow 3}$) is obtained from the figure 7 as $\{\text{Enqueue}_\text{Byte}, \text{ByteTimeOut}, \text{DoneTrans}_\text{Byte}\}$. Here EnqueueByte is represented by $\text{En}_\text{Byte}$ and ByteTimeOut by $t_1$ in figure 9. The messages Enable_Communication and Disable_Communication, DoneTransByte are not considered as the source of first two is CoilDriver and source of the latter does not belong to any of the components in the system.

ii) The states and the messages exchanged between the two components are considered to be equiprobable.
iii) From figures 8 and 9 for each pair of states of component CommunicationGnome we get $F_x^{-1}(y) = \{ v \in V_{2\rightarrow3} / F_x(v) = y \}$. For example, consider that the current state is Receive_Byte and a transition has occurred resulting in a different state Transmit_Bytes due to the message ByteTimeOut which is a subset of the message set $V_{2\rightarrow3}$ discussed above. Then we assign $F_{\text{Receive}_\text{Byte}}^{-1}(\text{Transmit}_\text{Byte}) = \{\text{ByteTimeOut}\}$ and the term is $P_{\text{CoilDriver} \rightarrow \text{CommunicationGnome}}[\{\text{ByteTimeOut}\}]^2$, which will depend on the number of messages in the message set. The value here is one.

iv) Similarly, we obtain the message set for $F_{s1}^{-1}(s2)$ and compute $P_{2\rightarrow3}[\{\text{message set}\}]^2$ where $s1$ and $s2$ are the states of a component. This is repeated for all the states in a component. Consider that the current state is Disabled then, $F_{\text{Disabled}}^{-1}(\text{Disabled}) = \{\text{ByteTimeOut, EnqueueByte}\}$ and the value of the term is 4. The reason for two messages being in the set are on receiving any of these messages, the capsule remains in its Disabled state.

v) Since, $V_{\text{CoilDriver} \rightarrow \text{CommunicationGnome}} (V_{2\rightarrow3})$ and $S_3$ are the number of messages in the message set and number of states of the component CommunicationGnome and after repeating the step (iv) for all the states, we get $EP_{\text{CoilDriver} \rightarrow \text{CommunicationGnome}}$

$EP_{2\rightarrow3} = 0.5$.

3.1.4 Estimate Change propagation probability

The change propagation probability is estimated from the artifacts of the repository as the ratio of the number of the messages in the interface of component $C_j$ that affect the
component $C_i$ to the number of messages in the interface of component $C_j$. The SAMT then produces a bar chart showing the CP of all components in a given system.

Consider the sub state system of pacemaker in figure 6 discussed in section 3.1.3. Computation of the CP values is explained basing on the pacemaker case study. The following two diagrams are provided for providing a basic explanation of how two components (otherwise) capsules communicate with each other.

![Figure 10. Message passing between two capsules](image)

Two capsules in a UMLRT model communicate with each other through the ports via a protocol. These two ports are connected with the help of a connector as shown in figure 3. The communication between these capsules is exclusively based on message passing as shown in the figure 10. Figure 11 is just a conceptual diagram showing the role of a protocol in message passing between any two capsules in an UMLRT model. A protocol role defines all the set of messages that are sent and received by a capsule through its port. The sent messages are named as outgoing and the received messages as the incoming messages in the figure 11. Generally, a protocol role specifies the perspective of one of the capsules in such a communication.
More information about this description and the figures of this UMLRT models can be obtained from the rose tutorials by downloading the rose tool at [13].

The architecture of the pacemaker case study is shown in figure 6. Using these artifacts one can identify the components (otherwise known as capsules) and connectors of the component based systems. These components names can then be used to name the rows and columns of CP matrix.

Figure 7 shows a message protocol coilcomm between a pair components CoilDriver and CommunicationGnome. This artifact provides the message sets $V_{\text{CoilDriver} \rightarrow \text{CommunicationGnome}} (V_{A \rightarrow B})$ and $V_{\text{CommunicationGnome} \rightarrow \text{CoilDriver}} (V_{B \rightarrow A})$ that are going between the two components. Similarly, one can get all such message sets between each pair of components in the system using the Rose-RT tool with the help of the script files in summit basic.
Considering the system in figure 6 and on executing the script file on the UMLRT model, one can get the set of messages $V_A$ that are present in the interface component $C_A$ and messages $V_{A\rightarrow B}$ that are the subset of the message set $V_A$ and which affect the component $C_B$ from the artifacts of the system specification. By using the equation (1), the change propagation probability can be obtained of the system (Table 2).

<table>
<thead>
<tr>
<th></th>
<th>Reed Switch</th>
<th>Coil Driver</th>
<th>Communication Gnome</th>
<th>Atrial Model</th>
<th>Ventricular Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reed Switch</td>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Coil Driver</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Communication Gnome</td>
<td>0</td>
<td>0.14</td>
<td>1</td>
<td>0.43</td>
<td>0.43</td>
</tr>
<tr>
<td>Atrial Model</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Ventricular Model</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 2. Change propagation matrix of pacemaker**

To illustrate the computing of the change propagation matrix in detail, the computation of the change propagation from component ReedSwitch to CommunicationGnome $C_{P_{ReedSwitch\rightarrow CommunicationGnome}} (CP_{1\rightarrow 3})$ is shown in the following steps:

i) Message set $V_{ReedSwitch\rightarrow CommunicationGnome} (V_{1\rightarrow 3})$ is obtained as $\{\text{CommEnable, CommDisable}\}$ from the figure 12 and the message set $V_2$ is obtained from the Rose RT tool as $\{\text{CommEnable, CommDisable, CommEnable, CommDisable}\}$. The latter two signals in the message set $V_2$ are exchanged between the components ReedSwitch and CoilDriver ($V_{1\rightarrow 2}$).
ii) All the messages are considered to be equiprobable and the messages that are exchanged between any two capsules are considered to be unique. Therefore messages with a similar name are considered to be different if they are exchanged between two different sets of capsules via different protocols.

iii) Since $V_{1\rightarrow 3}$ and $V_2$ are the number of messages, we get $CP_{1\rightarrow 3} = 0.5$.

![Protocol Specification for commEnableCoil](image)

Figure 12. Protocol specification between components 1 and 3.

3.1.5 Estimate size of change

The size of change (SC) is estimated from the artifacts of the repository as the ratio of the number of affected messages of component $C_j$ due to the changes in the interface of component $C_i$ to the number of messages in the component $C_j$. The SAMT then produces a bar chart showing the SC values of all the components in a given system.
On the similar lines in section 3.1.4, we can get the size of change matrix as shown in the table 3.

<table>
<thead>
<tr>
<th></th>
<th>Reed Switch</th>
<th>Coil Driver</th>
<th>Communication Gnome</th>
<th>Atrial Model</th>
<th>Ventricular Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reed Switch</td>
<td>1</td>
<td>0.5</td>
<td>0.4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Coil Driver</td>
<td>0</td>
<td>1</td>
<td>0.6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Communication Gnome</td>
<td>0</td>
<td>0.5</td>
<td>1</td>
<td>0.67</td>
<td>0.67</td>
</tr>
<tr>
<td>Atrial Model</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.33</td>
</tr>
<tr>
<td>Ventricular Model</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.33</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3. Size of Change matrix of pacemaker

3.2 Structural description of SAMT
The SAMT is divided into 3 layers, they are a metamodel layer, metrics layer and a user interface layer. These layers are shown in the form of packages in figure 13.

3.2.1 Metamodel layer
Once all the inputs are uploaded to the server after performing the static analysis using tools like Understand for Java by Scientific Toolworks Inc and Rational rose real time by IBM Rational software. This metamodel layer captures information like data-types, connectors and components of the system.
The types of inputs need to be uploaded to the server are based upon the option selected by the user. If the user selects the option for calculating error propagation, change propagation and size of change on UMLRT models, couple of text files obtained by the script files by using static analysis tool rational rose real time are needed to be uploaded.

i) These inputs files contain information like list of capsules, protocols, connectors, ports and messages for calculating change propagation and size of change on UMLRT models.

ii) The other input file contains list of capsules, state information, list of signals for calculating the error propagation on UMLRT models.

**Figure 13. SAMT Architecture [28]**
If the user selects the option for calculating the change propagation and size of change on applications with source code in Java, the files needed to be uploaded to the server are

i) Object cross reference report obtained with the help of the static analysis tool Understand for Java. This file contains a list of objects along with declaration and usage references.

ii) The other inputs to be uploaded are the files that contain the list of components, methods and attributes respectively. These are obtained with the help of the script files which are executed while analyzing the applications with source codes in Java with the help of the tool Understand for Java.

3.2.2 Metrics layer
In this layer, basing on the generated model various algorithms are applied to obtain corresponding metrics. These include error propagation, change propagation and size of change on UMLRT models and change propagation, size of change metrics on applications with source code in Java.

3.2.3 User interface layer
Finally, in this layer various outputs are presented to the user with the help of graphical user interface in the form of tables and bar charts.
CHAPTER FOUR:
SOFTWARE ARCHITECTURE METRICS TOOL (SAMT) DESIGN AND IMPLEMENTATION

This chapter explains in detail the design and implementation of the web-based Software Architecture Metrics Tool (SAMT). The case tool is available at the following URL:


4.1 SAMT Server-side Requirements

The basic requirements needed at the server-side are

Hardware Requirements

- A multimedia computer

For good performance, the system should have a 2.0-3.0 GHz PIII CPU with a minimum of 512 MB RAM, 20GB of hard disk. The current version of the tool is built on Windows XP professional system.

Software Requirements

- The server side code should be mounted on the Sun Java System Application Server.

- A platform with Java virtual machine 1.2 or higher is needed.

- We have integrated a third party commercial tool, the Espress Chart from the Quadbase Systems Inc [23] to plot and display results for the various software metrics discussed in this work.
4.2 SAMT Client-side Requirements

As such there are no hardware requirements for the client side, the software requirements are the following,

- A Java enabled web browser supporting JDK 1.4 or higher can be used but Internet Explorer with version above 5.0 is best suitable for viewing the outputs of the metrics tool.
- Third party commercial tools like Understand for Java [15] and Rose Real time [13] are required for the static analysis of the UMLRT models and applications with source code in Java respectively.

4.3 Introduction of web based tool

The home page of the environment is designed in such a way as to give the user a brief introduction of the web based tool. The user can then navigate through various links provided in the home page to make himself aware with the working of the software architecture metrics tool and its uses. There are a few software requirements on the client side that are needed by the user and apart from this, the user has to have the knowledge of using the static analysis tools described in section 4.2. There are also links provided in the home page to navigate to if the user wants to know in detail about the ongoing research work and publications on the software architectural level metrics project. The structure of the web based tool is shown in the figure 14.

The tool follows a two tier architecture with J2EE technology as shown in figure 15. Sun application server forms tier one with business logic and other functionalities supported by any server in general. Tier-2 is the client tier. Clients access the functionalities using a browser and they can also view the results with the help of applets.
Figure 14. Structure of the web based tool
4.4 Upload Module

In this module the user has to upload all the required text files to the local disk of the server. All the files can be uploaded to the server side using the servlet uploadBean. The war file which is deployed on the application server has been obtained at [26]. Various jsp’s and html pages that show up as the user navigates through this web based tool are integrated into the servlet. The applet code which displays various results to the user after computing software metrics like error propagation, change propagation and size of change is also mounted on to the application server by integrating into the servlet uploadBean.

To upload files and view results, the user has to first select either the unified modeling language real time (UMLRT) module or Java Understand module from the web based tool environment. After selecting one of these modules, the user will then be educated about the procedure to be followed to compute and view the results of the software metrics.
4.4.1 **UMLRT Module**

The user has to download two script files UMLRT_CP.ebs and UMLRT_EP.ebs written in summit basic script from the server. User has to open UMLRT model using the static analysis tool rational rose real time and execute these two script files. These script files generate two text files Capsules.txt and EPInput.txt and are stored in the C drive of the user. If the user wants to compute just change propagation and size of change metrics on his UMLRT module, he needs to upload the Capsules.txt file to the server and if he wants to compute error propagation metric on the UMLRT model, he has to upload both the Capsules.txt and EPInput.txt files to the server.

Figure 16 is the sequence diagram depicting the user uploading the text files required to obtain the results on UMLRT models.

![Sequence diagram](image)

**Figure 16. Sequence diagram for uploading files for UMLRT Module**
4.4.2 Java Understand Module

If the user wants to compute change propagation and size of change metrics on the Java based applications, he has to select the Java understand module. The user first has to download 3 script files files.pl, functions.pl, variables.pl written in perl. The user has to open the application with the help of the static analysis tool Java understand and execute these script files on the application. These script files generate three text files Classes.txt, variables.txt and functions.txt. Sharp.pux file can be obtained by the user by opening the application with Java Understand tool by entering the application’s name as Sharp in the tool prompt and by executing the reports generate command in the tool. All these four text files are to be uploaded to the server.

![Sequence diagram for uploading files for Java Understand Module](image)

Figure 17. Sequence diagram for uploading files for Java Understand Module
Figure 17 shows the sequence diagram of a user uploading all the required files to compute results for the applications in Java code.

4.5 Metrics Module

In this module metrics are computed on the uploaded files. The outputs are square matrices of size N x N, where N is the total number of components in the uploaded model. These outputs are generated as text files which are saved on the local disk of the server. These text files are later displayed in both table and bar chart formats using an applet. To compute metrics like error propagation, change propagation and size of change on Java based applications and UMLRT models for a remote client, a web service is used. The ear file is packaged and deployed onto the Sun Java System Application server. All the required algorithms needed to compute the metrics are integrated into the web service. To compute the metrics and view the results, the user has to select one of the two modules, Java Understand module or the UMLRT module.

4.5.1 UMLRT Module

After uploading the two files, the user has to use the calculateUMLRT button to compute the metrics. Figures 18 and 19 are the sequence diagrams that depict the sequence of actions that occur when the user clicks calculateUMLRT button. Figure 18 shows the sequence of messages that are invoked for the computation of the metric Error propagation on UMLRT models, whereas figure 19 shows the messages that are invoked for the computation of the metrics Change propagation and Size of change metrics on the UMLRT models.
4.5.1.1 Computation of Error Propagation metric

Figure 18. Sequence diagram for computing EP on UMLRT models

4.5.1.2 Computation of Change Propagation and Size of Change metrics

Figure 19. Sequence diagram for computing CP and SC on UMLRT models
4.5.2 Java Understand module

After uploading the files, the user has to use the calculateJavaUnderstand button to compute change propagation and size of change metrics on applications in Java. Figure 20 is the sequence diagram which depicts the sequence of actions that occur when the user clicks the calculateJavaUnderstand button.

![Sequence diagram for computing CP and SC on Java based Applications](image)

Figure 20. Sequence diagram for computing CP and SC on Java based Applications

4.6 Graphical user interface Module

In this module, the user has to first have the applet running in order to view the metrics. The user can have the applet running by clicking DisplayMetrics button on the DisplayGui java server page. Once the applet starts running in the browser, the user can choose to view any or all of the metrics but one at a time basing on the module chose. The user can view Error Propagation, Change Propagation and Size of Change, if he has
selected the mode UMLRT and Change Propagation and Size of Change if Java Understand mode selected. Figure 21 is the sequence diagram showing the sequence of messages that pass between the objects when the user views various metrics depending on the mode chosen.

**Figure 21. Sequence diagram for displaying the metrics to the user**
CHAPTER FIVE:
ASSESSING QUALITY ATTRIBUTES OF SOFTWARE ARCHITECTURES
USING THE TOOL

The study of software architectures is gaining importance due to its role in various aspects of software engineering like product line engineering, component based software engineering and other emerging paradigms. With the increasing emphasis on design patterns, the traditional practice of ad-hoc software construction is slowly shifting towards pattern-oriented development. Various architectural attributes like error propagation, change propagation, requirements propagation provide a wealth of information about software architectures. In this research work, change propagation probability (CP) is shown to be helpful and effective in assessing the design quality of software architectures. Two different architectures (one that employs patterns versus one that does not) for the same application are studied. Change propagation metric has been analyzed and compared with respect to other coupling-based metrics.

Main objective here is to show the role and use of CP in assessing the design quality (design quality here, is considered in terms of extensibility, maintainability, reusability etc..) of software architectures. Illustrate of how, the perspective and insights offered by CP complement the information provided by traditional object-oriented metrics is also provided.

An experiment is designed that compares architectures using object-oriented metrics and change propagation matrices. The goal of the experiment is to assess to what extent the object-oriented metrics on one hand and the CP matrices on the other hand are good predictors of architectural quality attributes. To this effect, sample applications are considered and two candidate architectures are derived for each: one that is based on
design patterns (hence is presumably of higher quality) and one that is design ad-hoc, without predefined patterns. The following methodology has been applied:

1. Prepare a pair of architectures for the same application. One of the architectures is designed using design patterns while the other has no patterns.

2. Apply the CP metric on both architectures.

3. Apply other object-oriented metrics on both architectures.

4. Analyze and compare the results. The architecture that employs software patterns should have a better quality in terms of extensibility and maintainability.

It is worth noting that the comparison of design quality has been carried for the architectures with the help of the Software Architecture Metrics Tool (SAMT). Currently tool supports applications with source codes in Java and unified modeling language real time (UMLRT) models but plan to extend it to support various other languages like C, C++, etc is going on. Inputs to the tool are obtained with the help of the static analysis tools Understand for Java tool [15] and Rational rose real time, SAMT generates the CP matrix of components in the architecture.

5.1 Case Studies

Experiments on the source code are carried out on two case studies using their architectural information. For each case study, two different architectures are experimented, one that employs patterns versus one that does not. Then, the change propagation metric is compared with other coupling-based, object-oriented metrics before and after applying the design pattern. Finally, evaluation of change propagation coefficient $CPC$ for the entire architecture and comparison of the numerical indices of the two versions are performed.
5.1.1 Job Application

First case study is a simple application where an employer is seeking applications for the various jobs available. There are two versions; one version is a simple switch case whereas the other version is implemented using the strategy pattern. Strategy design pattern comes into play when there are different implementations of an algorithm. The subclasses of the abstract class define the algorithm and define the implementations according to their needs. More flexibility is introduced when applying the pattern, if there are new positions to be filled rather than modifying the switch cases we just add another subclass to the abstract class that fulfills the new criteria.

Analysis is restricted to the components that exist before and after the application of the pattern. Figure 22 and Figure 24 show the reverse engineered class diagram and the change propagation probabilities of the case study when using switch cases.
Figure 22. Class diagram of Job Application before applying strategy pattern. [29]
Figure 23. Class diagram of Job Application after applying strategy pattern [29]
Figure 24. Change propagation of Job Application before applying strategy [29]

Figure 25. Change propagation of Job Application after applying strategy [29]
Figure 26. Weighted Methods per Class and McCabe Cyclomatic Complexity for Job Application case study [29]

Figure 23 and Figure 25 show the reverse engineered class diagram and the change propagation probabilities of the case study after applying the strategy design pattern. The Change Propagation Coefficient (CPC) for the architectures before and after using the design pattern are 0.18 and 0.11 respectively. From the CPC values, the architecture, which employs a design pattern, is better in design quality than the one which does not. Figure 26 shows the Weighted Methods per Class (WMC) and McCabe Cyclomatic Complexity metrics (MCC) for the components before and after applying the pattern. The JobApplicantForm component has been improved by employing strategy design pattern.

5.1.2 Colleague States

The second case study is an application that tracks the states of colleague components. Each colleague will update its state according to its current state and the changes to the states of the other colleagues. Reverse engineering has been performed on
the architecture of the case study using Rational Rose tool [13]. Figure 27 shows the class diagram of the initial design with the colleague components directly coupled to each other. In Figure 28, the mediator pattern [7] is used to let the interactions of the colleagues be more independent with respect to each other and facilitate the addition of new colleagues to the architecture. Generally, the mediator pattern provides a mean of encapsulating the various interactions of the other objects.

Figure 29 and Figure 30 show the change propagation probabilities of the architectures before and after applying the mediator pattern. In Figure 27, recognize that the three colleague components are tightly coupled to one another. In Figure 28, adding the mediator pattern decreased coupling between the colleague components. The three colleague components are completely decoupled with respect to one another.

The Change Propagation Coefficient CPC for the architecture that does not employ any design pattern is 0.11, where as CPC value for the one, which employs mediator design pattern is 0.05. From the CPC values, the architecture, which employs a design pattern, is better in design quality when compared to the same architecture that does not.

One important point here is that the change propagation coefficient CPC is calculated only to the components, which are present in both the architectures. Actually these are the components which are most important because these are the ones that have the basic functionality of the application. The other components are just present due to the added design patterns and whose function is mainly to glue all the components present in the architecture.
Figure 27. Class diagram of an initial design [29]

Figure 28. Class diagram of a design that uses mediator design pattern [29]
Figure 29. CP for the simple design on case study Colleague States [29]

Figure 30. CP for the case study on architecture employing mediator pattern [29]
Figure 31 shows the Weighted Methods per Class (WMC) and McCabe Cyclomatic Complexity (MCC) metrics for the architecture before and after using the mediator pattern. All the three colleague components have been improved in terms of complexity when a mediator design pattern has been employed. Both the metrics WMC and MCC show this improvement as a decrease in value for the components in the architecture that employs a design pattern.

One of the disadvantages of using the mediator design pattern is the localization of the behavior of the components. The distributed behavior of the components is localized to few components in the architecture that employs a mediator design pattern. Even though, the architecture that employs a mediator design pattern is more extensible,
maintainable and reusable than the one which does not. When adding new colleagues to
the architecture, we need only to change the components of mediator pattern. All the
other colleagues will not be affected as they will not be directly coupled to the new
colleague component.

5.2 Comparison of Change Propagation Metric with Other Metrics

In this section, the change propagation metric is compared with respect to three other
coupling-based, object-oriented metrics: Coupling Between Objects (CBO), Response
For a Class (RFC) and Message Passing Coupling (MPC) [5]. The analysis is restricted to
the components that exist before and after the application of the pattern. Figure 32 shows
the Coupling Between Objects (CBO) for the two case studies before and after the
application of the design pattern. Observing figure 32, one can conclude that CBO metric
is not sufficient to compare and state that an architecture is better in design quality when
compared with another. Though there are improvements in the CBO values for the three
colleague components in the Colleague States case study, there is an increase in the CBO
value for the JobApplicationForm component in the architectures which implements the
strategy pattern for case study Job Application.

In Figure 33, the Response For a Class (RFC) metric could not show the difference
between the two architectures in the Job Application case study. In fact, it showed that
the architecture that employs a simple design is better in design quality than the
architecture that employs strategy design pattern. On the other hand, for the Colleague
States case study the RFC metric confirmed a relative improvement of using design
patterns. This shows that RFC metric may not be a good choice to compare between two
candidate architectures.
Figure 32. CBO for the case studies on Colleague States and Job Application [29]

Figure 33. RFC for the case studies Colleague States and Job Application [29]
Figure 34. MPC for the case studies on Job Application and Colleague States [29]

Furthermore, the MPC metric showed no difference in design when applied to both the architectures on each of the two case studies. Figure 34 shows the values for the MPC metric computed on the Job Application and the Colleague States case studies. MPC metric could not show the difference between the architectures in both cases.

On the other hand the change propagation probability metric shows these variations and points out that one of the architectures is better than the other. Check the change propagation probability values for the three colleagues components in the Colleague States case study, figures 29 and 30, and JobApplicationForm component in the Job Application case study, figures 24 and 25. Change propagation probability CP metric can show different perspective and complements the usage of other object oriented metrics: CBO, MPC and RFC in both case studies.
5.3. Related Work

The work done by authors in [14], [11], [4] and [9] was more towards establishing the known benefits of design patterns like reusability, flexibility, extensibility etc. Since, there has been less insight and less work done in the area of design patterns so far, any amount of research done in this field is really helpful to a novice programmer. The various concepts related to design patterns are best understandable with the help of examples with source code as in [14], [4] and [9]. Though the examples considered in [14] were UML class diagrams, but were live examples.

Authors in [14] and [9] have done similar work when compared to the research work documented here. In [14], the authors have shown that the architecture employing a design pattern is more flexible than a simple design. However, our concern here is to show the use of change propagation metric in the assessment of the design quality of software architectures.

In [9], the authors talk about the reduction of change proneness among the various components of the source code that employ design patterns but were not able to show that the architecture is less coupled. In general when the components of architecture are said to be less change prone, then it is implied that they are less coupled and this is what is shown in this research work. This leads to believe that only two object-oriented metrics – WMC (Weighted Methods per Class) and RFC (Response For a Class) –out of the six Chidamber and Kemerer set of object oriented metrics support the fact that the source code that employs design pattern is better in design quality. In addition, the MPC metric, when applied to the case studies, did not show any measured improvement. All these
results show that there is a need of a metric that can state the difference between two architectures in terms of criteria of better design.[1]
CHAPTER SIX: TESTING AND RESULTS

The testing on Software Architecture Metrics Tool (SAMT) was done on different test cases and various scenarios. The testing on the web-based tool was done by verifying the working of various modules of the tool. All the modules have been tested and most of the problems have been solved. The web based tool was also tested from various systems by accessing the service both remotely and locally. The local accessibility was tested by invoking the tool on the systems using the LAN in the SERC (Software Engineering Research Center) at West Virginia University. The tool was also accessed remotely. The purpose of this testing was to test for the accessibility of the server located at West Virginia University and the tool was found to be working fine. The tool was also tested for both local and remote access for accessibility of the services provided using a browser. This testing was essential as the browser had to support displaying various images, downloadable zip files, uploading files and third party tools like Espress Chart and displaying applets etc. All the tests and their results have been presented in the following sections.

6.1 Testing Scenarios for Upload Module

Problems and errors encountered and methods adopted to solve

- When files were uploaded to the local disk of the server and if there were any files existent in the server with similar names, the uploaded files were saved with a ticket number appended to the file name. So, the destination folder in the local disk of the server had to be purged before uploading files to the server for each session the user logs into.
To keep track of all the files uploaded to the server, a vector has been used with the scope-session. Thus the tracking of the uploaded files was made possible and displayed to the user for each session he logs into.

**Successful Test Cases:**

- The user could upload files to the server and compute the metrics only on the uploaded files but not on the files saved as cache.
- Using variables with scope session enabled the user to keep track of the files uploaded to the server.

### 6.2 Testing Scenarios for Metrics Module

**Problems and errors encountered and methods adopted to solve**

- To compute metrics, algorithms have been written in both C++ and Java. Since we have deployed applications using Sun Java System Application Server which supports J2SE and J2EE, we had to integrate C++ code into Java using JNI. While creating the .h file, if the Java file has been packaged, then the command to be run should be `javah packagename.HelloWorld` instead of `javah HelloWorld`.
- To load a shared library named example, use the code `System.load(“rootDirectory\example”)` instead of `System.loadLibrary(“example”`). If these above two conditions haven’t been met the program cannot be able to find the path to run and produce the intended results.
Successful Test Cases

- The user was able to use the web based tool successfully both with local and remote access. He could compute all the metrics on both the Java based applications and UMLRT models using the tool.
- Integration of C++ code into Java was done using JNI and this made possible for embedding the legacy code into tool.

6.3 Testing Scenarios for Graphical User Interface Module

Problems and errors encountered and methods adopted to solve

The possible problems in the GUI module were the one’s that were encountered while displaying the applet in the browser at client side. They are as follows

- A major/minor version error might occur if the java files are compiled with different versions of J2SDK and deployed on the server, so it’s better to follow a convention and use a specific version of J2SDK.
- The applet class files have to be placed on top of the working directory. Example working directories are WEB-INF for Sun Java Application Server and web-apps for Tomcat. If this has not been met, the applet will never run in the browser at client side.

A few other problems have been encountered while using the third party tool Espress Chart for displaying bar charts. The Espress Chart can be used in many ways to plot different types of charts. One of them could be to use Espress Manager, but while using this, the user has to place the espressmanager.cfg file either with working directory or in the directory containing the applet class file. The alternate way is to use API’s provided by the Espress Chart to plot charts.
• The error occurred was espressmanger.cfg file could not be found and in such a case, the espress manger uses the software in the local host to plot charts. This results in displaying charts only for users who uses the tool locally. To overcome this problem, the API’s provided by Espress Chart have been used to plot and display charts to clients located remotely.

• Both the server and the client have to have the time set correctly. If either the server or client will fail to take care of this, an error can occur as license of Espress Chart not valid and this will result in the applet failing to display the chart.

**Successful Test Cases**

• The client was able to access the applet through his browser successfully and was able to display and view the metrics in both table and chart formats.

• The Espress Chart was successfully embedded in the web-based tool Software Architecture Metrics Tool(SAMT) and the clients were able to view the results in bar chart format also. The remote access was also made possible for the client to view bar charts.

**6.4 Problems encountered while using Sun Java Application Server**

**Methods adopted to overcome the problems**

Following are the problems that occur generally when using the Sun Java Application Server.

• Shutting down the Sun Java Application Sever is very crucial before turning off the power of the system on which the server has been deployed, if failed to take care of this, an error can occur which says Default server cannot be started while
trying to start the server again. Various documents have been gone through to overcome this problem but a proper solution wasn’t found. An alternative to overcome this problem however can be to uninstall the application server and reinstall.

- As given in section 6.3, the applet classes which execute on the client side have to be on top of working directory of the server. While adding the applet classes to the server using the application server, the .class files get added to the directory workingdirectory\Classes automatically and now the developer has to explicitly select these applet class files and drop on to the tree such that it falls on top of the working directory.

- All the files are uploaded to the directory on the local disk of the server and if these files have to be deleted, they are not to be deleted manually. If deleted manually can result in automatic shut down of the application server. The possible solution to this problem is to delete the files from the directory using session.remove() call from the jsp pages before uploading new files.

6.5 Problems encountered at client side while using browsers

Methods adopted to overcome the problems

- The web based tool was tested using the browsers IE and Firefox. The firefox was not able to support the images used in the user interface of the web based tool. IE worked fine and it’s the best browser to access the web based tool provided that it has JRE and proper security settings set.
6.6 Testing the tool on various case studies

The tool has been tested on various case studies for obtaining metrics such as error propagation, change propagation and size of change. The following screen shots shows the results on UMLRT models for change propagation metric. More information about these models can be obtained by downloading the tool at [13].

6.6.1 Case study- Pacemaker

![Screen shot of CP values for pacemaker case study]

Figure 35. Screen shot of CP values for pacemaker case study

66
6.6.2 Case study- Game of life

Figure 36. Screen shot of CP values for game of life case study
6.6.3 Case study- Callbacks

Figure 37. Screen shot of CP values for callbacks case study
6.6.4 Case study- IsrExample

Figure 38. Screen shot of CP values for IsrExample case study
6.6.5 Case study- DynamicForwarding

Figure 39. Screen shot of CP values for DynamicForwarding case study
6.6.6 Case study- ObserverPattern

Figure 40. Screen shot of CP values for ObserverPattern case study
CHAPTER SEVEN: CONCLUSIONS AND PROSPECTS

7.1 Conclusions

From this thesis, the following conclusions can be drawn

[1]. Software Architecture Metrics Tool (SAMT) is designed and implemented as a web based tool for computing and analyzing software metrics like Error propagation, Change propagation and Size of change on UMLRT models and Java based applications.

[2]. In this research, various algorithms for computing metrics like Error propagation, Change propagation and Size of change metrics on UMLRT models have been designed and implemented. These algorithms are then integrated into the web based tool. The algorithms in C++ code for computing Change propagation and Size of change metrics on Java based applications have also been integrated into the web based tool using JNI.

[3]. The applicability of the change propagation probability CP in assessing the design quality of software architectures is also shown. CP proved to give a different perspective than the one provided by the classical object-oriented metrics like the CBO, RFC and MPC. Metrics like (CBO, RFC and MPC) are shown not to be sufficient to state that an architecture is measurably better in design quality when compared to another architecture.

This research is part of a larger project to analyze and validate various architectural attributes like Error Propagation (likelihood that an error in one of the components propagates to other components), Change Propagation (likelihood that a change in one of
the components requires changes to be done to other components) and Requirements Propagation (likelihood that a change in the requirements in one of the components needs a change in the requirements in the other components), Diagonality etc. Significant work has been done on the architectural attribute Error Propagation, Change Propagation while work is continuing on the Requirements Propagation.

7.2 Future Work

The present version of the Software Architecture Metrics Tool (SAMT) can be improved to provide other services. Work is ongoing for supporting models other than UMLRT and Java based applications.

- UML 2 has the concepts similar to that of the Rational rose real time of considering a system as composed of components and connectors. Research is going on in this aspect, so that tool can also enable users to compute metrics on models following UML 2.0 version.

- StarUML is an open source project that is available for free from sourceforge.net website at [27] unlike tools like Rational rose, Microsoft Visio and others. Research is going on to provide an addin for starUML, which on plugging in gives metrics like Error propagation, Change propagation and Size of change on UML models.

- The present tool supports models with .rtmdl and files with extensions .java. Work is going on to extend the model to support models with file extensions of .XMI and .XML. The intermediate results are written onto text files to be parsed. The performance and accuracy can be increased by using XML documents instead of text files.
• If the model to be analyzed is complex and huge, then the parsing algorithms might take up lot of time to compute metrics and display in applet. This can be cut down by having a ticketing service which emails the users when the results are ready to be viewed.

• In respect to analyzing the applicability of change propagation probability in assessing the design quality of software architectures, future prospects include studying larger case studies and automating the steps of the analysis methodology to facilitate applying it to more complex case studies.
REFERENCES


Attributes of Software Architectures” in 4th ACS/IEEE International Conference on Computer Systems and Applications, Sharjah, UAE, 2006
APPENDIX

1. WSDL file for computing error propagation, change propagation and size of change metrics on UMLRT models and Java based applications.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="changepropagationService" targetNamespace="urn:Foo"
xmlns:tns="urn:Foo" xmlns="http://schemas.xmlsoap.org/wsdl/"
  <types/>
  <message name="ChangePropagationIF_webcall_a">
    <part name="String_1" type="xsd:string"/>
  </message>
  <message name="ChangePropagationIF_webcall_aResponse">
    <part name="result" type="xsd:string"/>
  </message>
  <message name="ChangePropagationIF_webcall_b">
    <part name="String_1" type="xsd:string"/>
  </message>
  <message name="ChangePropagationIF_webcall_bResponse">
    <part name="result" type="xsd:string"/>
  </message>
  <portType name="ChangePropagationIF">
    <operation name="webcall_a" parameterOrder="String_1">
      <input message="tns:ChangePropagationIF_webcall_a"/>
      <output message="tns:ChangePropagationIF_webcall_aResponse"/>
    </operation>
    <operation name="webcall_b" parameterOrder="String_1">
      <input message="tns:ChangePropagationIF_webcall_b"/>
      <output message="tns:ChangePropagationIF_webcall_bResponse"/>
    </operation>
  </portType>
  <binding name="ChangePropagationIFBinding" type="tns:ChangePropagationIF">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="webcall_a">
      <soap:operation soapAction=""/>
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
namespace="urn:Foo"/>
      </input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
namespace="urn:Foo"/>
      </output>
    </operation>
    <operation name="webcall_b">
      <soap:operation soapAction=""/>
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
namespace="urn:Foo"/>
      </input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
namespace="urn:Foo"/>
      </output>
    </operation>
  </binding>
</definitions>
```
<soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<service name="ChangepropagationService">
<port name="ChangePropagationIFPort">
<soap:address
location="REPLACE_WITH_ACTUAL_URL"/></port></service></definitions>

2. The web based tool – Software Architecture Metrics can be accessed using the link


The screen shot shows the home page for using the tool and executing demos.
Screen shots of the various steps to be followed by an analyst to obtain the metrics are shown below. Screen shots for obtaining change propagation metric on UMLRT model using the web based tool – Software architecture metrics is shown below.

Step-1: Open model
Step-1 Contd……

Top level view of the architecture of the Pacemaker case study.
Step-1 Contd……

Screen shot of the sub level architecture of the Pacemaker case study.
Step-2: Open script

The script file to be executed on the UMLRT model can be downloaded at http://157.182.197.19:8080/uploadBean/downloads/UMLRTScripts.zip.
Step-2 Contd……..

Execute the script using the start icon from the Rational rose tool.
Step-2 Contd…………..

Save the output of the script file with the name Capsules.txt.
Step-2 Contd……..

Snapshot of the information extracted using the VB script on the model.
Step-2 Contd.........

Snapshot of the information (captured using the script file) contained in the file Capsules.txt.
Step-3: Upload Capsules.txt file to the server

Files can be uploaded to the server using the link

Step-3 Contd……

Select the file Capsules.txt and upload to the server.
Step-4: Compute metrics using the button “Calculate metrics on UMLRT”.
Step-5 : Screen shot of the applet displayed in the browser when the analyst clicks “View Results” button.
Step-6: In the applet, the analyst has to click the desired metric to view the results. Here the analyst has to click “Change Propagation on UMLRT” button from the drop down menu to view the results.
Step-6 Contd………

Snap shot of the change propagation metric values on Pacemaker case study in both table and bar chart formats.