N-Model Methodology for Enhancement of Object-Oriented Software

by

ANAM SAHOO

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2019

To my family who constantly provided unconditional love and support especially to my grandmother who planted the seed to make me what I am today.

ACKNOWLEDGEMENTS

I would like to extend my sincerest gratitude to my supervising professor Dr. David Kung for his unwavering support and guidance during the course of my doctoral studies. I would also like to thank Dr. Christoph Csallner, Dr. Yu Lei, Dr. Bahram Khalili for their interest in my research, generously sharing their ideas and time to serve in the committee.

I am grateful to Mehereb Noshirwan Irani for helping me in conducting the experimental evaluation of the N-model methodology discussed in this thesis. Further, I would also like to thank my students for their participation in the experimental evaluation. I would also like to thank Department of Computer Sciences and Engineering for providing me all the opportunities for doctoral studies and allowing me to share my industry experience by teaching as an adjunct staff here at UT Arlington.

I would like to thank all my teachers who taught me during this graduate program here at The University of Texas at Arlington. I am also grateful to all my teachers for giving me the knowledge and inspiration throughout my life in India and here in United States.

Finally, I would like to express my love and appreciation for my family, parents and grand parents who have supported me through my studies all throughout my life. I am grateful for my wife and children and the sacrifices they have made to allow me to progress to this point in my career.

May 8, 2019

ABSTRACT

N-Model Methodology for Enhancement of Object-Oriented Software

Anam Sahoo, Ph.D.

The University of Texas at Arlington, 2019

Supervising Professor: David C Kung

Software maintenance typically consumes an average of 60% of software life costs, of which more than 60% are spent on enhancements. These are a challenge for the software community, in which hundreds of millions of lines of legacy code need to be modified during enhancement maintenance. Unfortunately, our extensive literature survey and industrial experiences show that there is a lack of a systematic methodology for software reengineering and for enhancement. As a consequence, software engineers use ad hoc approaches to enhance a legacy system. This dissertation presents an agile process, called the N-model process and methodology, for enhancing object-oriented legacy systems. The process consists of a release planning phase to quickly identify release changes, followed by an iterative enhancement phase to implement the changes, and finally a formal system validation phase to ensure that the changes are properly incorporated. The methodology details the steps to perform these phases. This thesis has defined a set of three categories of ten metrics for evaluating an enhancement methodology, and applied them to evaluate our N-model methodology in comparison with ad hoc approaches for enhancing and evolving legacy systems. Although the experiment is limited in scope, it shows that the N-model methodology significantly outperforms ad hoc approaches.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Software maintenance typically consumes an average of 60% of software life costs, of which more than 60% are spent on enhancements [20, 46]. These costs are a challenge for the current software community, in which hundreds of millions of lines of legacy code need to be modified during enhancement maintenance. The problem becomes even direr when the enhancement project is performed by engineers who do not have sufficient knowledge of the legacy system and documentation is inadequate or non-existent. Software enhancement needs a process and a methodology. Figure 1.1 highlights the difference between the two [29]. First, a process, also called a process model, specifies *when to do what*, but not how to do them. According to Cockburn, "A process describes how activities fit together over time, often with pre- and postconditions for the activities" [14] (page 151). For example, the waterfall process states that one should first complete project planning, followed by requirements analysis, then design, and so on. It does not specify how to carry out these activities. Unlike a process, a methodology details the steps of *how* to carry out the activities of a process. It can be viewed as an implementation of a process. This definition coincides with that of Cockburn: "I used the word methodology as found in the Merriam-Webster dictionaries: a series of related methods or techniques" [14] (page 149). Processes are paradigm independent but methodologies are not. In simple terms, a paradigm is a style, or a way to view the world [30]. For example, the Structured Analysis and Structured Design (SA/SD) paradigm views the world as consisting of business processes or transforms interacting with each other, while the object-oriented (OO) paradigm views the world as consisting of objects. As a consequence, a process does not dictate

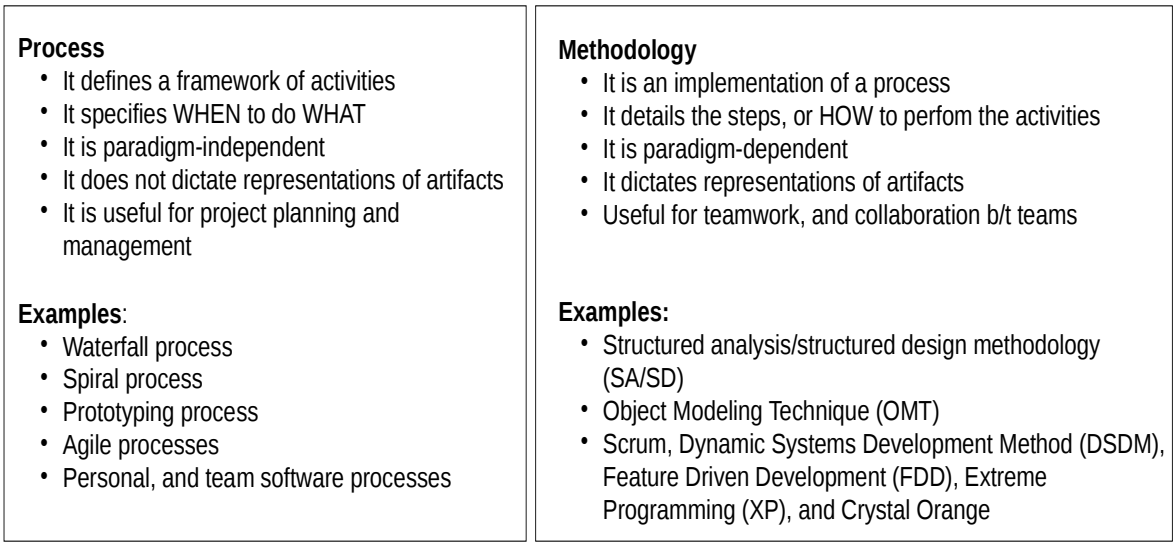


Figure 1.1: Difference between process and methodology

representations of artifacts, but a methodology does. For example, SA/SD methodologies often use data flow diagrams and structured charts [19, 65], while most OO methodologies use UML [6, 34, 29]. A process facilitates project planning and project management because it defines when to do what. A methodology facilitates teamwork, and communication and collaboration between teams and team members because the same set of steps and procedures as well as the same representation of artifacts are used by all participating members.

According to Cockburn and Maier and Rechtin [14, 36], methodologies fall into four categories, ranging from least to most desired as follows:

1. *heuristic methodologies*, which are based on lessons learned,
2. *participative methodologies*, which are stakeholder-based and capture aspects of customer involvement,
3. *rational methodologies*, which are based on method and technique,
4. *normative methodologies*, which are based on solutions or sequences of steps known to work for the discipline.

Unfortunately, *"most of current software development is still in the stage where heuristic methodologies are appropriate*," as pointed out by Cockburn [14]. Indeed, our extensive literature survey reveals that there is a lack of non-heuristic methodologies for enhancing OO software systems [7].

We summarize our related literature survey on object-oriented software enhancements to the following four categories;

1. Methodologies
2. Processes
3. Frameworks
4. Generalized enhancement activities

Firstly, in our extensive research survey of enhancement methodologies, we came across few papers [44, 61, 23] that included the word *methodology* in their titles or contents, but we did not find any heuristic methodology matching Cockburn's above definition. However a framework-based agile enhancement process using rational unified process(RUP) called PARFAIT for customer relationship management software is described in Cagnin [11], and our prelimenary work on N-Methodology in Sahoo [48]. Secondly, Pighin [44], Zhenchang [63], Rickman [47], and Polo [45] elucidate the evolution of software enhancement process. Frameworks, the third category, are discussed by Hyland-Wood [23], Tahvildari [54] and Vora [58, 59]. Finally, on the category of generalized enhancement activity, we find several papers on reverse engineering and reengineering, including Pashov [43], Hong [22], Caludia [13], Serebrenik [49], Zaidi [69] , Tonella [57], and Parsons [42]. We will shed some light on each of these works in the related work section 2 below.

In this thesis, a process and a methodology are presented for enhancing object oriented software. The process consists of three distinct phases: a *release planning* phase, an *iterative enhancement* phase, and a *system validation* phase as shown in Figure 1.2. Each

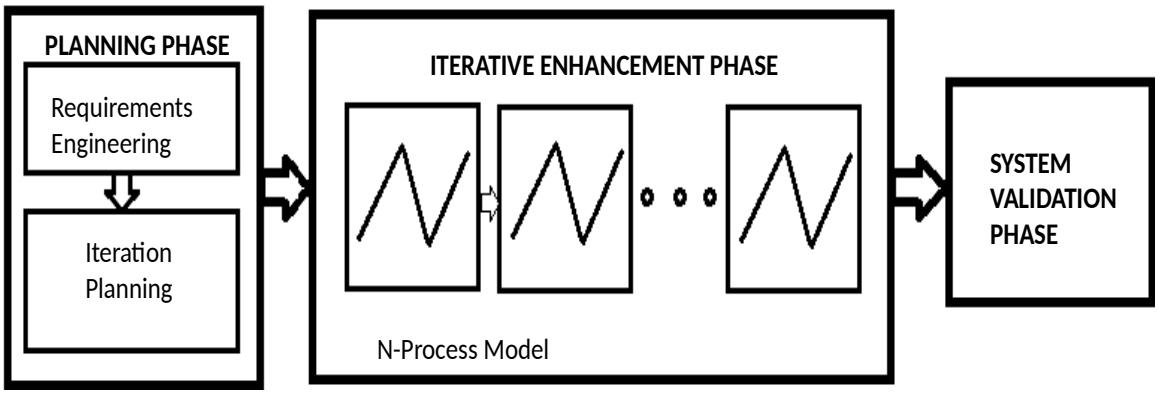


Figure 1.2: Overview of the enhancement process

product release begins with a quick agile planning phase, which consists of two activities. First, enhancement requirements are identified and prioritized by applying information collection techniques. Second, new use cases and changes to existing use cases are derived. Finally, planning for release iterations is performed to produce a roadmap to guide the iterative enhancement activities. The iterative enhancement phase consisting of a series of iterations. Each iteration has three legs: the reverse engineering leg, the reincarnation leg, and the iteration validation leg as shown in Figure 1.3. The reverse engineering leg is performed if analysis and design documentation of the existing system is non-existent, outdated, or inadequate. It recovers the as-built analysis and design artifacts, and represents them using UML diagrams. This higher-level abstraction of the existing software facilitates reasoning about changes to the existing system. In particular, the incarnation leg uses the higher-level abstraction to identify classes and methods to add, modify and delete, taking into account the so-called ripple effect or change impact to other classes when modifications are made to a class or method [31]. This middle leg also implements the identified changes. Finally, during the iteration validation leg, integration testing, regression testing and beta testing are performed to validate the changes.

At the end of the release, a formal system validation phase is performed before the software is deployed to the client site. Since the shape of these three legs looks like a

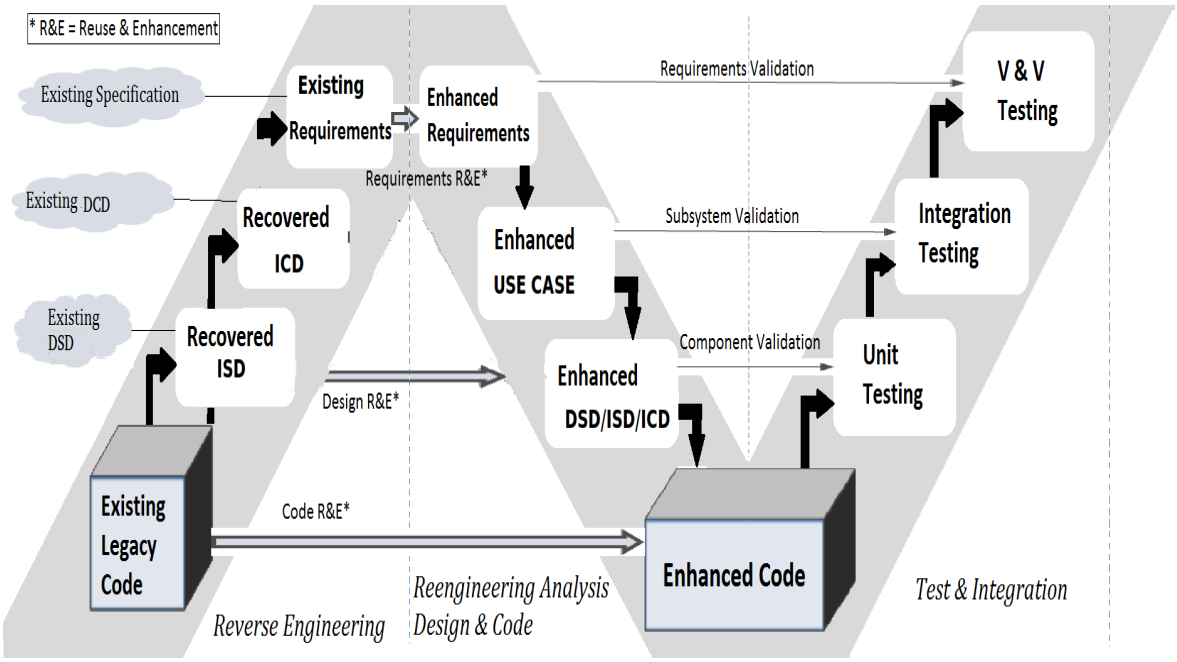


Figure 1.3: Each iteration of the enhancement phase

capital "N," as shown in Figure 1.3, we call this process model the N-process model, or N-model for short. Our methodology that implements this process is called the N-model methodology.

The contributions of this thesis are summarized as follows. First, current software development including software enhancement is mostly based on lessons learned, as pointed out by Cockburn [14]. The proposed methodology will greatly improve the current practice by providing a systematic approach for enhancing OO systems. Second, we present three categories of metrics for assessing a software enhancement methodology. These are: *process metrics, design metrics*, and *testability metrics*. Finally, we have applied these metrics to evaluate the efficacy of the proposed methodology. The evaluation results show that the proposed methodology significantly improves students' performance.

The layout of the thesis is as follows. Chapter 2, describes the related literature survey and discusses the difference between this work and existing work. Chapter 3 presents

the N-model methodology for enhancing a legacy OO software. A legacy academic advising software is used as the running examples to illustrate the steps of the methodology. Chapter 4 evaluates the effectiveness of the N-model process and methodology via a series of case studies. Conclusions and future work are presented in Chapter 5.

CHAPTER 2

LITERATURE REVIEW

First of all, we need to make it clear what we mean by the term *methodology* because this term has been used with different meanings in the literature. In particular, the word is interchangeably used for process, method and framework among others causing confusion within the academic and industrial communities alike. In this paper, we adopt the definition proposed by Alistair Cockburn. That is, a methodology is "a series of related methods or techniques for *how to carry out* the activities that allow for coordination of peoples actions on a team for better outcome" [14] (page 149). Numerous papers include the word *methodology* in their titles or contents but after reading these articles we found that their works did not meet the definition for methodology as proposed by Cockburn. For example, the paper titled "A Methodology and Heuristics for Re-Architecting a Legacy System" [61] documents a model-based systems engineering process to re-architecture a legacy avionics system. As discussed in Section 1, a process (also called a process model) specifies *when to do what* but not how to do them. This is in line with Cockburn's definition — "a process describes how activities fit together over time, often with pre- and postconditions for the activities" [14] (page 151). Another paper titled "Towards a Software Maintenance Methodology Using Semantic Web Techniques and Paradigmatic Documentation Modelling" [23] describes a *framework* for software maintenance using RDF, OWL and SPARQL web techniques to encode the system metadata for software enhancement. Pighin's paper titled "The New Methodology for Component Reuse and Maintenance" presents a cataloguing technique for insertion and retrieval of information about software components that can be automatically built and organized from existing software [44]. Our extensive literature

search revealed that none of the plethora of works on enhancement of OO software presents a methodology. To the best of our knowledge, the N-model methodology presented in this paper is the first on this topic. In the following, we present the related works in four categories: software enhancement process, maintenance framework, reverse engineering, and reengineering.

2.1 Software Enhancement Process

The first category is software enhancement process. According to Cockburn's definition, "a process describes how activities fit together over time, often with pre- and post-conditions for the activities." [14] (page 151). For example, the waterfall process dictates that one must first complete project planning, followed by requirements analysis, then design, and so on. However, it does not specify how to carry out these activities. Xing [63] elucidates a process to study the evolution of an object oriented system software by identifying the types and styles of changes and constructing a change tree. A sequence of such change trees constitutes the system's evolution profile that provide valuable insight for software enhancement and project management. Besides Pighin's [44] cataloging technique, mentioned earlier in context to methodology (could be categorized as a process), Briand [8] delineates a software evolution process that has six steps starting from identifying organizational entities with which the maintenance team interacts, identifying the phases involved in creation of new phases, identifying the generic activities in each phase, analyzing the past releases, analyzing problems that occurred in past phases and building an organizational structure that lead to problems. Rickman [47] describes a process for combining an OO model and a structured model. The paper presents a process for the system architecture and requirements engineering (PSARE), which then leads to an enhanced requirement model and finally results in an enhanced software. Polo [45] proposes a software mainte-

nance process, MANTEMA, which uses ISO/IEC 12207 as a tailoring process for software maintenance and evolution. This process has four maintenance phases, starting from urgent corrective, perfective, preventive and finally adaptive. Yongchang [64]describes four software maintenance models, namely the quick modify model, Boehm model, IEEE model and Iterative enhancement model. Kanchana [28] proposes a process for software quality enhancement using process optimization and the Taguchi method. Ioannis [25] proposes enhancements using the Agile Unified Process (AUP) for a banking software. AUP is a hybrid approach combining Rational Unified Process (RUP) with agile methods and has the same four phases; Inception, Elaboration, Construction and Transition as that in RUP. Again,we see that several papers propose processes and not methodologies.

2.2 **Software Enhancement Framework**

The second category of related works is software enhancement framework, commonly defined as a generic platform which can be specialized as needed to improve the maintainability of a software system. Hyland-Wood [23] describes a framework for software maintenance using RDF, OWL and SPARQL web techniques to encode the system metadata for software enhancement. The RDF graph can be used to enable the language neutral relational navigation of software system that aids in software understanding and maintenance. Zou [70] presents an interactive and incremental migration framework in which a legacy procedural code is reengineered into an OO platform. The framework allows for the representation of the legacy code in the form of XML based syntax tree from the original source code that can be used to identify the classes, associations and aggregations. Tahvildari [54] proposes a framework for the development of a quality driven enhancement of an OO software based on design patterns. This framework starts with the creation of a catalogue of design modifications, followed by transformation of design

patterns from primitive transformations and finally the encoding of non-functional requirements using the softgoal interdependency graph. Vora [58, 59] presents CFFES, an architectural framework for software enhancement. CFFES is divided into five components: Temporal Meta-Data, Process Controller, Rule Base, Archiving Engine, and Application Architecture. Again,the framework generalizes the platform but does not quite address the specific concern of how to perform the enhancement activity on an object oriented legacy software system, demanding the need for a step by step methodology.

2.3 Software Reverse Engineering

The third category concerns reverse engineering. Understanding the existing software is an essential part of any software maintenance and enhancement. Missing artifacts such as requirements, architecture, domain model, use cases and design artifacts have always been an issue for any enhancement or reengineering projects. Many techniques, however, have been presented for reverse engineering some of these artifacts. One such technique for software architecture recovery using a feature modeling technique is described by Pashov [43]. Hong [22] presents a tool JBOORET, which uses a parser based approach to recover the high level design and source models from the system artifacts. Caludia [13] describes a framework to reverse engineer use case diagrams from JAVA code in the context of Model Driven Architecture (MDA) focusing on transformations at model and metamodel level. Serebrenik [49] proposes an algorithm for reverse engineering UML sequence diagrams from Enterprise Java Beans (EJB) enterprise applications involving business method interceptors. Zaidi [69] presents an approach for reverse engineering sequence diagram by analyzing execution traces produced dynamically from an object oriented application. It uses k-tail merging algorithm to produce a Labeled Transition System (LTS) that merges the collected traces, which then get translated into a sequence diagram. Tonella [57] de-

scribes a static analysis algorithm for the extraction of the class diagram from the source code, and Parsons [42] takes a critical look at a range of representative approaches for extracting component interactions from the source code and also outlines relative advantages and disadvantages to these approaches. Finally, Dugerdil [17] describes how to use the traditional Rational Unified Process (RUP) to reverse engineer a legacy system.

2.4 **Software Re-engineering**

The fourth category is reengineering. Transforming the existing software to a different form or platform without changing its functionality is occasionally needed to ease future enhancements. Several incremental approaches to migrate function oriented software to object oriented paradigms are described by Wong [62], Suenobu [52], Tan [56], and Siddik [50]. A framework-based agile reengineering process named PARFAIT is defined by Cagnin [11]. PARFAIT uses the Rational Unified Process(RUP) and GREN framework for re-engineering Business Resource Management(BRM) software. Two other reengineering processes to transform the existing procedural software to a reusable object oriented form are described one by deBaud [16] using domain model and Cagnin [10] using design patterns. Another reengineering process for migrating legacy object-oriented system to component based system is described by Lee [35]. Improving maintainability by refactoring using clustering techniques are described by Mathur [37] and Alkhalid [2]. A catalog of software components can be built and organized from existing software for future reuse called ontology. An ontology-based approach to reengineer enterprise software to cloud computing is explained by Zhou [68]. Finally, Ebner [18] presents a Microsoft Windows-based RETH tool and describes how to trace artifacts all around in software enhancement projects.

CHAPTER 3

THE N-MODEL PROCESS AND METHODOLOGY

As described in Chapter 1, the enhancement process consists of three major phases as shown in Figure 1.2: the release planning phase, the enhancement phase, and the system validation phase. In this chapter, we describe in detail how these phases are to be performed. To facilitate understanding, we present here an academic advising software, called MavAppoint, to serve as the running example to illustrate the steps of the methodology. We then present the steps for each of the three major phases in the following sections.

MavAppoint was written in Java and Java Server Pages (JSP), and uses Mysql as the database management system (DBMS). It supports three types of users: department administrators, academic advisers, and students. Table 3.1 lists some of the existing requirements and use cases that will be used to illustrate the methodology. This legacy system will be enhanced with enhancement requirements, to be described in the following sections.

Table 3.1: Legacy requirements and use cases of the running example

Legacy Requirements	Legacy Use Cases
R1: An administrator can create, edit, and delete advisors and define their privileges.	UC1: Create Advisor UC2: Edit Advisor UC3: Delete Advisor
R2: Advisors can login and specify their advising time.	UC4: Login User UC5: Update Schedule
R3: Student can create accout, login to it and schedule an appointment with an advisor.	UC6: Create Student UC4: Login User UC7: Schedule Appointment

3.1 Planning Phase

The planning phase involves two activities. First, enhancement requirements are identified and prioritized by applying information collection techniques. Their impact on existing use cases is assessed, resulting in new, to-be-modified, and to-be-deleted use cases, respectively. Second, planning for release iterations is performed to assign the new, to-be-modified and to-be-deleted use cases to the release iterations.

3.1.1 Identify and Prioritize Enhancement Requirements

Correctly identifying and prioritizing enhancement requirements are critical to the success of an enhancement project. To identify and prioritize enhancement requirements, information collection techniques such as customer presentation, user survey, user interview, and literature survey are used. For MavAppoint, we assume that the following enhancement requirements are identified. We also assume that they have the highest priority:

R4. Users shall be able to change passwords.

R5. Users shall be forced to change system-generated temporary passwords when logging in for the first time.

R6. User ID and passwords must be encrypted before storing them in the database.

3.1.2 Derive New, To-Be-Modified and To-Be-Deleted Use Cases

The second step of the planning phase is to derive new, to-be-modified and to-be-deleted use cases from the enhancement requirements and existing use cases. This step is carried out by examining each of the enhancement requirements and existing use cases as follows:

3.1.2.1 **Deriving New Use Cases**

A new use case can be derived if an enhancement requirement introduces a new business process to be implemented. Such a business process is often indicated by a verb-noun phrase that is application-specific or domain specific. Examples are "change passwords" in R4, and "deposit fund" for an automated teller machine (ATM) application. To derive new use cases, we highlight all such verb-noun phrases in the enhancement requirements. Each such verb-noun phrase must satisfy all of the following conditions:

1. The verb-noun phrase must denote a complete end-to-end business process of the application or the application domain.
2. The business process must begin with a user (or actor).
3. The business process must end with the user (or actor).
4. The business process must accomplish a business task for the user (or actor).

Example. From the enhancement requirement R4, we identify the verb-noun phrase "change password." Clearly, changing passwords is a business process that is application-specific. It begins with the user, and ends with the user seeing a "password is changed successfully" message. It also accomplishes a business task for the user. Therefore, we derive a new use case *UC8: Change Password*.

3.1.2.2 **Deriving To-Be-Modified Use Cases**

Obviously, enhancement requirements may affect existing use cases, which need to be modified to fulfill the enhancement requirements. To identify such use cases, one of the following two approaches can be used:

1. *Using a requirement-use case traceability matrix.* If a requirement-use case traceability document such as a traceability matrix is available, then the affected use cases can be derived from the matrix. A traceability matrix M has rows representing re-

quirements, and columns representing use cases. If a use case UC_j is derived or related to a requirement R_i, then $M_{i,j}$=1 (otherwise, it is blank). If an enhancement requirement affects an existing requirement R_i, then all use cases with $M_{i,j}$=1 could be affected and modification may be needed.

2. *Examining the impact of enhancement requirements.* If a requirement-use case traceability matrix or equivalent does not exist, then one has to examine each of the enhancement requirements and each of the existing use cases to determine which use cases are affected and need to be modified.

   Example. The enhancement requirement R4 merely introduces a new use case (*UC8: Change Password*). It does not affect existing use cases. Enhancement requirement R5 (users must change system-generated temporary passwords) affects existing use cases *UC4: Login User*. This is because UC4 must be modified to force the user to change the temporary password. Enhancement requirement R6 affects *UC1: Create Advisor* and *UC6: Create Student*. This is because these two use cases must be modified to encrypt student ID and the system-generated temporary passwords. Moreover, R6 also affects *UC7: Schedule Appointment*, which must be modified to encrypt student ID.

3.1.2.3 **Deriving To-Be-Deleted Use Cases**

Usually, existing use cases to be deleted are explicitly specified. These can be easily identified. For our running example, no use case need be deleted.

3.1.3 **Assigning Use Cases to Release Iterations**

Table 3.2 shows the impact of the enhancement requirements to use cases. The new, to-be-modified, and to-be-deleted use cases are assigned to iterations in this step. It is beyond the scope of this thesis to detail this step. Therefore, it is only briefly described. First,

Table 3.2: New, to-be-modified and to-be-deleted use cases

Enhancement Requirements	Use Cases	Category
R4: Users can change passwords.	UC8: Change Password	New
R5: Users must change system-generated temporary passwords when logs in for the first time.	UC4: Login User UC8: Change Password	Modified New
R6: Student ID and all passwords must be encrypted before storing them in the database.	UC1: Create Advisor UC6: Create Student UC4: Login User UC7: Schedule Appointment UC8: Change Password	Modified Modified Modified Modified New

an agile estimation technique such as the poker game method [4] is applied to obtain an effort estimate for each of the use cases. An order to design, implement, modify, delete, and test the use cases is derived based on their dependencies and priorities. The use cases are then assigned to iterations according to the order. For this paper, we assume that requirements R4-R6 have the highest priority and must be completed in the first enhancement iteration.

3.2 **Iterative Enhancement Phase**

The iterative enhancement phase consists of a series of iterations. Each iteration has three legs: the reverse engineering leg, the reincarnation leg, and the validation leg. Reverse engineering is performed only if design documentation is nonexistent, outdated, or inadequate. The reverse engineering leg uses various tools to produce various UML diagrams from the legacy code. It is performed only for the to-be-modified, and to-be-deleted use cases. During the incarnation leg, new UML design diagrams are constructed for new use cases, and the reverse-engineered UML diagrams are modified for the to-be-modified use cases. The reverse-engineered UML diagrams are used to identify impact of to-be-deleted use cases to existing classes of the legacy system. modify and execute existing test cases,

and perform regression testing to ensure that the new, to-be-modified and to-be-deleted use cases are implemented correctly. Finally, the validation leg generates and run new test cases. Table 3.3 summarizes the needed reverse engineering, reincarnation, and validation artifacts and activities for each of new, to-be-modified, and to-be-deleted use cases.

Table 3.3: Enhancement artifacts and activities for use case categories

Artifacts/Activities	*Category of Use Case*		
	New	*To-Be-Modified*	*To-Be-Deleted*
Reverse engineer Implemented Sequence Diagram(ISD)	Not needed.	Yes, needed to take into account of the enhancement requirements.	Yes, needed, to identify potential classes and methods to delete.
Reverse engineer Implemented Class Diagram (ICD)	Yes, needed, to identify classes and methods to reuse or extend.	Yes, needed, to identify classes and methods to reuse or extend.	Yes, needed, to identify classes and methods to delete.
Construct Expanded Use Case (EUC)	Yes, needed.	May be needed if actor-system interaction behavior need be changed.	No, not needed,
Construct/Modify (DSD)/ISD	New DSD - with new and existing classes from ICD. May apply software design patterns (SDP) such as adapter, controller, and facade.	Modified ISD, consider reusing existing classes from ICD.	Use ISD to identify classes and methods to delete.
Modify ICD	Add new classes and modify classes of ICD according to the DSD. Enhance design with SDP.	Add new classes and modify classes of ICD according to modified ISD. Enhance design with SDP such as adapter and facade.	Delete identified classes and methods from ICD.
Create New Test Cases	Needed for new classes and new methods, and existing classes affected by changes.	Needed for modified classes and methods, and classes affected by the changes.	No new test cases needed.
Do Regression Test	Yes, needed.	Yes, needed.	Yes, needed.

3.2.1 **Reverse Engineering**

During the planning phase, the new, to-be-modified and to-be-deleted use cases are identified. Moreover, these use cases are assigned to iterations so that each iteration will handle only a few of these use cases. Reverse engineering is the first step of the iterative enhancement phase. That is, it is the first thing to do during each enhancement iteration. Reverse engineering is performed only if design documentation for the legacy system is nonexistent, inadequate or outdated. It will produce an as-built class diagram and as-built sequence diagrams. The former is referred to as the implementation class diagram (ICD), and the latter is referred to as the implementation sequence diagram (ISD). The ICD is produced for the whole system or selected subsystems or components. It is useful for identifying reusable classes of the legacy system, as well as identifying change impact when an existing class or method is changed. The ISDs are useful for identifying changes to be made to an existing use case, as well as identifying classes and methods to be removed when deleting an existing use case.

This reverse engineering step of each iteration of the iterative enhancement phase performs the follow activities for the new, to-be-modified and to-be-deleted use cases allocated to the current iteration:

1. *Reverse engineering implementation class diagram (ICD).*
2. *Reverse engineering implementation sequence diagram (ISD).*

How to perform these two activities are detailed as follows.

3.2.1.1 **Reverse Engineer Implementation Class Diagram**

Many existing tools can be used to reverse engineer the implementation class diagram. Techniques and tools for the reverse engineering code to produce ISD and ICD

Table 3.4: Actor-system interaction scenario for Login User

Actor: User	System: MavAppoint
1. The user clicks the Login button on the home page.	2. MavAppoint shows a login page with user name and password fields.
3. The user enters the user name and password, and clicks the Submit button.	4. MavAppoint shows the user's dashboard if the user authenticates successfully.
5. The user sees his dashboard.	

are found in the following cited tools and papers [24, 33, 34, 38, 42, 49, 53, 55] and [24, 33, 41, 53, 55, 57], respectively. Some of these tools could be used to reduce effort.

3.2.1.2 **Reverse Engineer Implementation Sequence Diagrams**

The steps for reconstructing an implementation sequence diagram are described as follows:

- *Step 1.* Observe how a user interacts with the legacy system to carry out the to-be-modified or to-be-deleted use case, and describe the actor-system interaction scenario using a two-column table. The left-column entries of the table describe the user input and user actions, and the right-column entries present the corresponding system responses. For example, the actor-system interaction scenario for the *Login User* use case can be described as shown in Table 3.4.
- *Step 2.* Identify nontrivial steps in the actor-system interaction scenario. A nontrivial step is an entry in the right column of the actor-system interaction table. It requires background processing to produce the system response. A right-column entry is a nontrivial step if one of the following rules can be applied:
  1. If the step requires background processing, then it is a nontrivial step.

2. If the system responses of the step are different for different users, then it is a nontrivial step. This is because background processing is required to produce the different system responses for different users.

Example. In Table 3.4, the nontrivial step is step 4.

- *Step 3.* Identify the button or menu item clicked or selected by the user in left-column entry preceding the nontrivial step. In the legacy code, identify the corresponding button or menu item and its action listener. Trace the action listener handler code for objects that send and receive messages between them. Construct a sequence diagrams to show the messages sent and received between the objects. This is the implementation sequence diagram (ISD) that is reverse engineered for the to-be-modified or to-be-deleted use case. There are many existing tools for generating an ISD from legacy code Techniques and tools for reverse engineering code to produce ISD and ICD are found in the previously cited tools and papers [24, 33, 34, 38, 42, 49, 53, 55, 24, 33, 41, 53, 55, 57]

Example. The implementation sequence diagram (ISD) that is produced from the MavAppoint legacy code is shown in Figure 3.2. The partial LoginServlet legacy code from where the ISD was reverse engineered is given in the Figure 3.1,

```
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	HttpSession session;
	/**
	 * @see HttpServlet#doGet(HttpServlet request, HttpServletResponse response)
	 */
	protected void doGet(HttpServlet request, HttpServletResponse response) throws ServletException, IOException {
		session = request.getSession();
		if(session.getAttribute("failedAttempts")==null){
			session.setAttribute("failedAttempts",0);
			session.setAttribute("userFailed","temp");
		}
		request.getRequestDispatcher("/WEB-INF/jsp/views/login.jsp").forward(request,response);
	}
	/**
	 * @see HttpServlet#doPost(HttpServlet request, HttpServletResponse response)
	 */
	protected void doPost(HttpServlet request, HttpServletResponse response) throws ServletException, IOException {
		session = request.getSession();
		String  userID= request.getParameter("userID");
		String password = request.getParameter("password");
		GetSet sets = new GetSet();
		sets.setUserID(userID);
		sets.setPassword(password);
		try{
			//call db manager and authenticate user, return value will be 0 or
			//an integer indicating a role
			DatabaseManager dbm = new DatabaseManager();
			LoginUser user = dbm.getUser(sets);
			if(user != null){
				retCode = user.CheckUser();
				if (retCode == no_such_uid)
				{
					session.setAttribute("no_such_uid", user);
					response.sendRedirect("index");
				}
				else if(retCode == no_passwd_match)
				{
					session.setAttribute("no_passwd_uid", user);
					response.sendRedirect("index");
				}
				else
				{
					session.setAttribute("user", user);
					response.sendRedirect("index");
				}
			}
			else{
				//redirect back to login if authentication fails
				//need to add a "invalid username or password" response
				if(dbm.userExists(userID)){
					if(((String) session.getAttribute("userFailed")).equalsIgnoreCase(useerID)){
						int attempts = (int) session.getAttribute("failedAttempts");
						session.removeAttribute("failedAttempts");
						session.setAttribute("failedAttempts", attempts+1);
						if(attempts>=3){
							dbm.invalidateUser(sets.getUserID());
						}
					}else{
						session.setAttribute("userFailed", userID);
						session.setAttribute("failedAttempts", 1);
					}
				}else{
					session.setAttribute("failedAttempts", 0);
				}
				response.sendRedirect("login");
			}
		}
		catch(Exception e){
			System.out.println(e);
			session.setAttribute("failedAttempts", 0);
			response.sendRedirect("login"); }}}
```

Figure 3.1: LoginServlet partial code segment for Login User

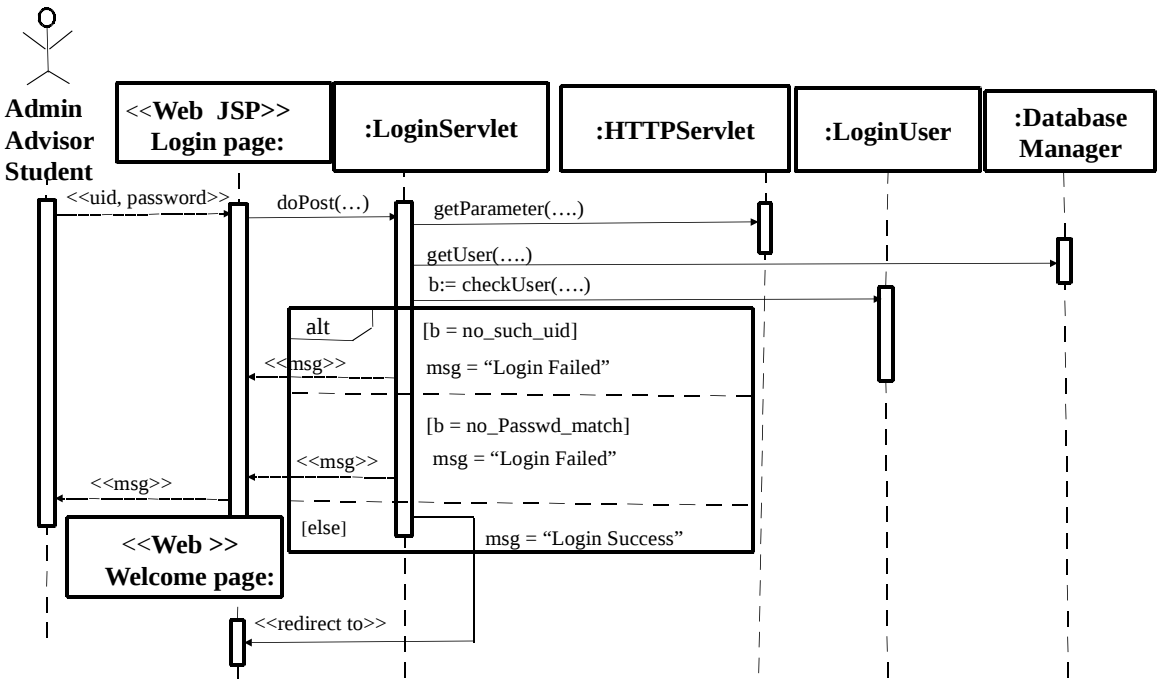


Figure 3.2: Implementation sequence diagram for nontrivial step of Login User

3.2.2 **Reincarnation**

During the planning phase, the new, to-be-modified and to-be-deleted use cases are identified. Moreover, these use cases are assigned to iterations so that each iteration will handle only a few of these use cases. In addition, an implementation class diagram (ICD) and implementation sequence diagrams (ISDs) for the new, to-be-modified and to-be-deleted use cases allocated to current iteration have been produced. In this reincarnation step, we will design and implement the new use cases, and modify and delete existing use cases assigned to the current iteration. The details on how to accomplish these is as follows.

3.2.2.1 **Dealing with New Use Cases**

Treatment for a new use case is similar to forward engineering, except that existing legacy code and some of the test cases may be reused. Forward engineering has been

Table 3.5: Actor-system interaction scenario for Change Password

Actor: User	System: MavAppoint
1. The user clicks the Change Password button on the home page.	2. MavAppoint shows a change password page with user name, password, new password, and confirm new password fields.
3. The user enters the fields as shown, and clicks the Submit button.	4. MavAppoint shows a password changed successfully message.
5. The user sees the password changed successfully message.	

described in various publications [6, 9, 29, 34, 46, 51]. The design and implementation of a new use case involves the following steps, which are detailed in the paragraphs that follow:

- *Step 1.* Produce an actor-system interaction scenario. The actor-system interaction scenario should describe how an actor or user will interact with the system to carry out the new use case. The result of this step is a two-column table similar to the one shown in Table 3.4. For the running example, we have one new use case, that is, *Change Password*. The actor-system interaction scenario for this new use case for nontrivial step is shown in Table 3.5.
- *Step 2.* Produce a design sequence diagram (DSD) for each nontrivial step of the actor-system interaction scenario. The DSD should describe how software objects will interact with each other through function calls to carry out the background processing in order to produce the system response from the user input. During this step, software reuse of the legacy code should be taken into account. The implementation class diagram (ICD) is useful in this regard. That is, if one were to look up the class in the ICD, if the class is found, then it should be reused, possibly with methods and attributes added to the class as per the following logic: (a) If a class is not in the ICD,

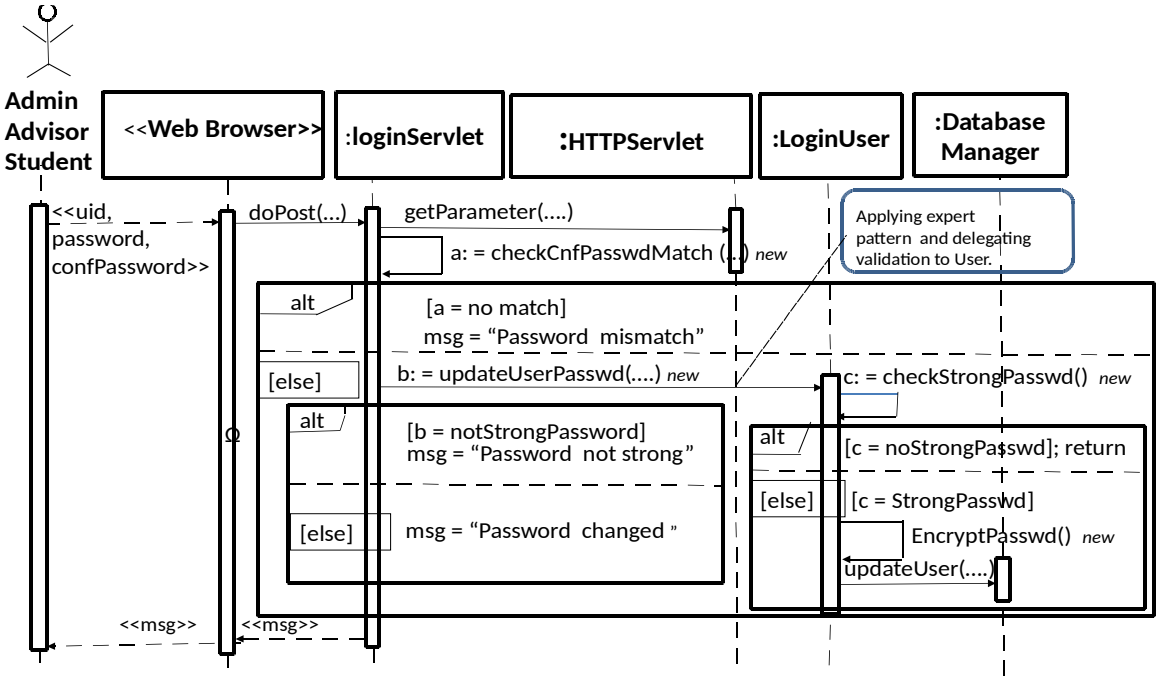


Figure 3.3: Design sequence diagram for change password use case

then add the class to the ICD along with all its methods and attributes extracted from the DSDs; (b) If a class is in the ICD but some of its methods/attributes extracted from the DSD are not in the ICD, then add these to the class in the ICD; (c) If a relationship is not in the ICD, then add the relationship to the ICD.

- *Step 3.* Modify the existing ICD with new classes, new methods and new relationships.
- Step 4. Implement and test the new classes, new methods and new relationships for nontrivial steps.

Example. Figure 3.3 shows a design sequence diagram for the nontrivial step of the Change Password use case. In this design sequence diagram, the LoginServlet and LoginUser classes are reused classes. One new method (i.e., checkCnfPasswdMatch(...)) is added to the LoginServlet class, and three new methods (i.e., updateUserPasswd(...),

checkStrongPasswd(...) and encryptPasswd(...)) are added to the LoginUser class. These new methods are indicated in the design sequence diagram in Figure 3.3.

3.2.2.2 **Dealing with To-Be-Modified Use Cases**

During the reverse engineering phase of each iteration, we produce the as-built actor-system interaction scenario, which is a two-column table such as shown in Table 3.4. We also produce the as-built implementation sequence diagram (ISD). These reverse-engineered artifacts are useful for handling to-be-modified use cases. The steps for treating each of the to-be-modified use case are described as follows:

- *Step 1.* Examine the as-build actor-system interaction scenario as well as the ISD(s) for the to-be-modified use case, and identify places that need to be changed in order to fulfill the enhancement requirements relevant to the to-be-modified use case.
- *Step 2.* Copy and modify the copies of the as-build actor-system interaction scenario as well as the ISD(s) accordingly, and ensure that the result satisfies the relevant enhancement requirements. During this step, software reuse should be taken into account and handled as follows. Before creating a new class, look for a reusable class in the design sequence diagrams created and the ISDs modified in the current iteration and then in the ICD. Addition or modification to the reusable class may be needed.

  Example. Figure 3.2 shows a recovered implemented sequence diagram for the non-trivial step of the Login User use case. In this implemented sequence diagram, a new method, getUser(...) is added, keeping in mind the expert controller pattern, and the checkUser(...) method of LoginUser class is modified by calling another internal new method, the checkIfFirstAttempt(...). Additionally, if it is the first time login, the new changePasswrd use case shown in Figure 3.3 is invoked inside the doPost(...) of the LoginServlet class. Here modified to LoginUser classes are reused classes.

These new and modified methods are illustrated in the modified sequence diagram in Figure 3.4.

- *Step 3.* Identify new classes as well as modifications to existing classes from the modified actor-system interaction scenario and modified ISD(s), and update the ICD accordingly.
- *Step 4.* Implement the new classes as well as modifications to existing classes. Identify change impact to existing classes and change the impacted classes accordingly. Repeat this step until all change impacts are addressed.

  Example. Figure 3.5 shows the code segment of all new and modified classes and methods of LoginUser use case.

  The change impact can be computed according to the algorithm described by Kung [32]. To explain the algorithm, let $ICD = G(V, E)$ be a directed graph, where $V = \{C_1, C_2, ..., C_n\}$ is the set of classes, and $E = I \cup A \cup F$ the inheritance relationships (I), aggregation relationships (A) and function call relationships (F) between the classes. More specifically, a pair of classes $(C_1, C_2) \in E$ if C_1 is a subclass of C_2, or C_1 is an aggregate of C_2, or a function of C_1 invokes a function of C_2. Clearly, if C_2 is changed, then C_1 is affected and may need to be changed and retested. Since such a relationship is a transitive relationship, we can compute the change impact by computing the transitive closure of the class that is changed. Let $CI(C_j)$ denote the change impact or set of classes affected by changes to class C_j. Then the change impact $CI(C_j)$ is defined as follows, note that $C_j \in CI(C_j)$:

  $CI(C_j) = \{C_i | C_i = C_j \vee (C_i, C_j) \in E \vee (\exists C_k)((C_i, C_k) \in E \wedge (C_k, C_j) \in E)\}$.

  As a matter of fact, many existing reverse engineering tools such as the Object Relation Diagram (ORD) described by Kung [32] can compute the change impact for a class that is changed or deleted.

- *Step 5.* Test the new, modified as well as affected classes with new test cases and reusable test cases.

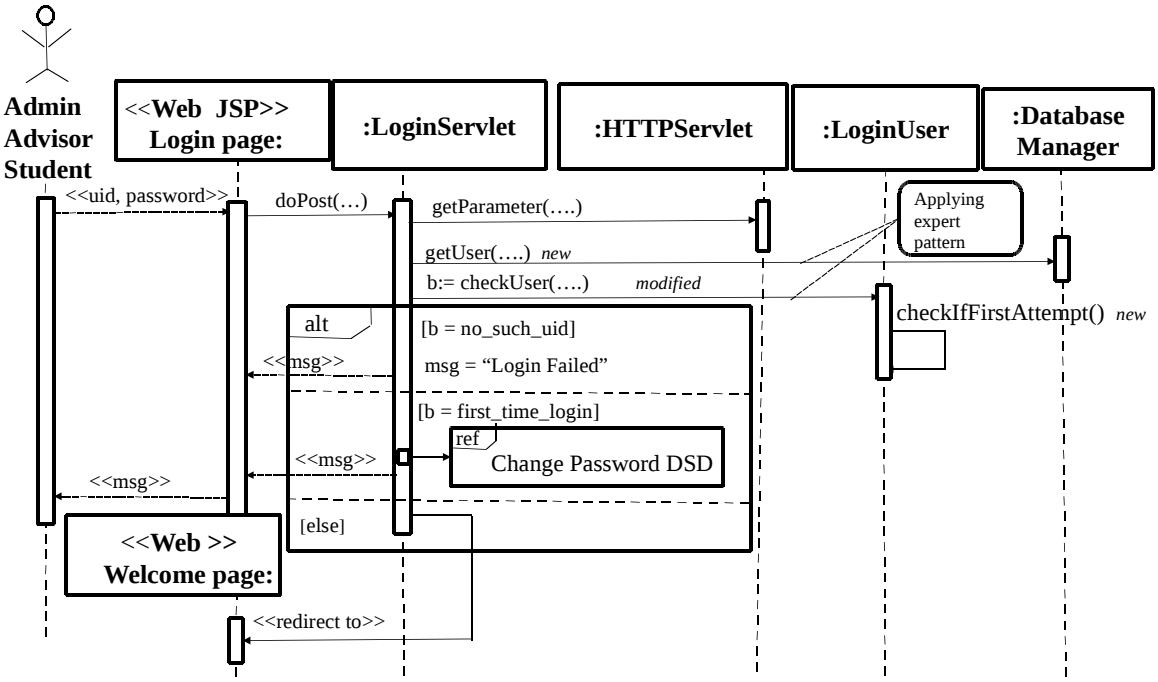


Figure 3.4: Login User ISD with marked enhancement changes

```
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	HttpSession session;
	/**
	 * @see HttpServlet#doGet(HttpServlet request, HttpServletResponse response)
	 */
	protected void doGet(HttpServlet request, HttpServletResponse response) throws ServletException, IOException {
		session = request.getSession();
		if(session.getAttribute("failedAttempts")==null){
			session.setAttribute("failedAttempts",0);
			session.setAttribute("userFailed","temp");
		}
		request.getRequestDispatcher("/WEB-INF/jsp/views/login.jsp").forward(request,response);
	}
	/**
	 * @see HttpServlet#doPost(HttpServlet request, HttpServletResponse response)
	 */
	protected void doPost(HttpServlet request, HttpServletResponse response) throws ServletException, IOException {
		session = request.getSession();
		String  userID= request.getParameter("userID");
		String password = request.getParameter("password");
		GetSet sets = new GetSet();
		sets.setUserID(userID);
		sets.setPassword(password);
		 HttpServlet newRequest; HttpServletResponse neResponse; int retCode;
		try{
			DatabaseManager dbm = new DatabaseManager();
			LoginUser user = dbm.getUser(sets);
			if(user != null){
				retCode = user.CheckUser();
				if(retCode == no_passwd_match)
				{
				    session.setAttribute("no_passwd_uid", user);
				    response.sendRedirect("index");
				}
				else if(retCode == first_time_login)
				{
					doGet(newRequest,newResponse);
					retcode=checkConfPasswdMatch(newRequest,newResponse);
					/* update user parameters and error checks  omited here… */
					user.updateUserPassword();
				 }
				else
				{
				   session.setAttribute("user", user);
				   response.sendRedirect("index");
				}
			}
			else{
				if(dbm.userExists(userID)){
					if(((String) session.getAttribute("userFailed")).equalsIgnoreCase(useerID)){
						int attempts = (int) session.getAttribute("failedAttempts");
						session.removeAttribute("failedAttempts");
						session.setAttribute("failedAttempts", attempts+1);
						if(attempts>=3){
							dbm.invalidateUser(sets.getUserID());
						}
					}else{
						session.setAttribute("userFailed", userID);
						session.setAttribute("failedAttempts", 1);
					}
				}else{
					session.setAttribute("failedAttempts", 0);
				}
				response.sendRedirect("login");
			}
		}
		catch(Exception e){
			System.out.println(e);
			session.setAttribute("failedAttempts", 0);
			response.sendRedirect("login"); }}}
```

Figure 3.5: LoginServlet enhanced partial code segment for Login User

3.2.2.3 **Dealing with To-Be-Deleted Use Cases**

Deleting a use case means deleting classes and methods that are only used by the use case. In the following, we present an algorithm for identifying classes that can be deleted. To identify classes to delete, we begin with the reverse-engineered implementation sequence diagram(s) (ISD) for each use case to be deleted. Intuitively, a class that appears in the ISD could be deleted if no other class depends on it. Therefore, we first identify classes that appear in the ISD. These include:

- classes of objects in the ISD that send or receive messages;
- classes of objects that are used as parameters to function calls;
- classes of objects that are return values of function calls.

For each class C that appears in the ISD, we compute the change impact $CI(C)$ of C. A class $C_i \in CI(C)$ can be deleted if no other class C_j depends on it. If class $C_k \in CI(C)$ has C_i as the only dependent, then C_k can also be deleted. This is summarized by the purge($CI(C)$) algorithm shown in Figure 3.6. The algorithm repeatedly removes classes of $CI(C)$ that have no dependent classes. In the algorithm, we use indegree(C_i) to denote the indegree of C_i, that is, the number of classes that depend on C_i. Formally, indegree(C_i)=$|\{C_j|(C_j, C_i) \in E\}|$. Obviously, if indegree(C_i) equals zero, then no other class depends on C_i, and hence, C_i can be deleted.

```
purge(CI(C)) {
        Input: CI(C)=change impact of class C.
        Output: CI(C) with not used classes removed.

        while ((∃C_i ∈ CI)(indegree(C_i)==0)) {
                CI = CI − {C_i};
        }
}
```

Figure 3.6: Deleting classes that are not used by other classes

Consider, for example, the ISD and ICD shown in Figure 3.7 and Figure 3.8, respectively. The former is the ISD for the *Change Password* use case, while the latter is part of the ICD. If this use case were to be deleted, then according to the above, we compute the change impact for each class in the ISD. The result is as given in Table 3.6. The LoginServlet's indegree value being zero, this class can safely be deleted based on the purge algorithm given in Figure 3.6.

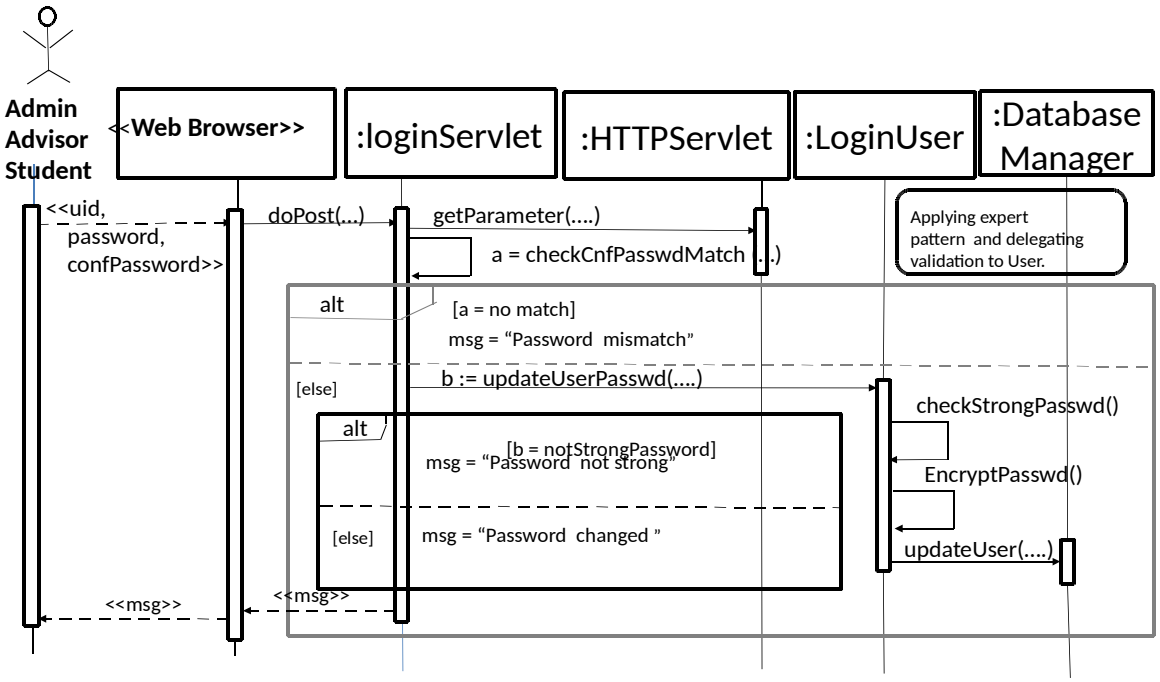


Figure 3.7: Implemented sequence diagram for change password use case

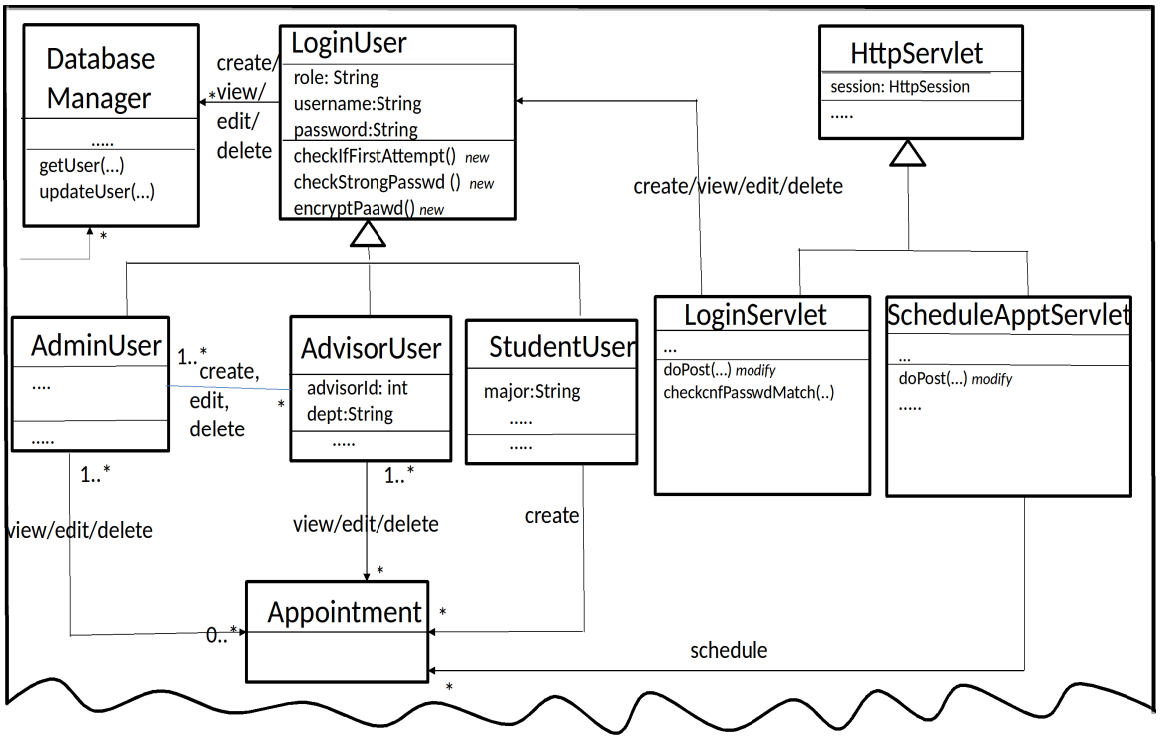


Figure 3.8: Partial Implemented class diagram

Table 3.6: Change impact classes for to-be-deleted change password use cases

Classes on ISD	Indegrees	Change Impact Class List
LoginServlet	0	LoginServlet
HTTPServlet	2	LoginUser, AdminUser, AdvisorUser, StudentUser, LoginServlet
LoginUser	4	LoginUser, AdminUser, AdvisorUser, StudentUser, LoginServlet
DatabaseManager	Many	Many

The algorithm for identifying methods to delete is similar, except that a call graph is used instead of the ICD. Call graph construction for OO programs as described by Grove [21] has been studied extensively, and many tools such as *doxygen,codeviz and valgrind* can be used to construct the call graph. Since a call graph is a directed graph, the

Table 3.7: Change impact methods for to-be-deleted change password use cases

Methods Impacted
LoginServlet::doPost(..)
LoginServlet::checkCnfPasswdMatch(..)
HTTPServlet::getParameter(..)
LoginUser::underpass(..)
LoginUser::checkStrongPasswd(..)
LoginUser::encryptPasswd(..)
DatabaseManager::updateUser(..)

notion of change impact and the purge algorithm can be applied similarly. The following methods are involved in our Change Password to-be-deleted use case as found in the ISD.

After applying the purge algorithm, the following resulting methods are found to be deleted from the code.

Table 3.8: Methods found to be deleted for to-be-deleted change password use cases

Methods to be deleted
LoginServlet::checkCnfPasswdMatch(..)
LoginUser::updateUserPasswd(..)
LoginUser::checkStrongPasswd(..)
LoginUser::EncryptPasswd(..)

3.2.3 **Iteration Validation**

The iteration validation step focuses on validating the implementation against the intended design and requirements by preparing and executing appropriate test cases. These include unit test cases, subsystem and system integration test cases, and acceptance test cases.

CHAPTER 4

CASE STUDIES

We have conducted two phases of case studies, a phase-1 preliminary qualitative and a phase-2 quantitative case study, to evaluate the N-model methodology to enhance several code bases of real-world web based applications.

Phase-1 consisted of two case studies that involved graduate students from two different semesters utilizing two different legacy code bases of the Academic Advising Scheduler Web (AASW) application. Section 4.1 describes the preliminary phase-1 case study in detail. The final quantitative case study using four categories for ten metrics is described in Section 4.2, where eight graduate students were asked to complete two assignments each within five weeks that involved enhancing two real-world, web-based legacy systems sequentially. The first assignment was to be completed with the students' existing knowledge, experience, and skills, while the second assignment was to be completed after being taught the N-model methodology. For these experiments, the following two hypotheses were designed:

Null hypothesis (H_0): The N-model methodology does not improve student performance in enhancement projects.

Alternative hypothesis (H_1): The N-model methodology does improve student performance in enhancement projects.

The following sections present the case study design, evaluation metrics, data collection, data analysis, evaluation results as well as result validity and threats in two separate sections titled "Preliminary Qualitative Case Study" and "Final Quantitative Case Study."

Table 4.1: Overview of Preliminary Case Studies

Case Study		1		2	
# of Participants		30 Graduate Students		31 Graduate Students	
Teams		2-4	1, 5-7	1,3,5,7	2,4,6,8
AAWS Project code base	Before learning the methodology	A	B	A	B
	After learning the methodology	B	A	B	A
	Duration	5 weeks	5 weeks	5 weeks	5 weeks

4.1 Preliminary Qualitative Case Study

4.1.1 Overview

In the preliminary phase, we conducted two qualitative case studies, as summarized in TABLE 4.1. These first two case studies involved 30-31 graduate students from two different semesters using two legacy code bases of the AASW system. For the first phase of the first case study, students were divided into seven teams and were asked to complete their enhancement assignments using their own chosen methods. Then, the N-methodology was taught to the class. In the second phase, the code bases were swapped, and the students were tasked with completing the second assignment following the learned methodology. In the second case study, eight teams were asked to learn and use the Rational Unified Process (RUP) [17, 11] and Agile Unified Methodology (AUM) [29] for the first assignment. Then, the teams used the N-methodology for the second assignment after swapping the code bases. For both phases of the case studies, data were collected from all the teams in the following two ways: (1) artifact submission that include enhanced code and (2) team demonstration of their working code to validate their claims.

4.1.2 **Data Analysis**

Analysis of the data collected from the first case study was performed by comparing different groups for the same legacy systems–that is, the quality of the enhanced code base A performed by the teams 2-4 in case study one using the methodology was compared to three out of the rest of four teams (randomly picked) using ad hoc ways. But in the second academic case study, the analysis was performed comparing student performance with RUP and AUM to that of the N-model methodology.

4.1.3 **Analysis and Result Interpretation**

The results of Case Study 1 indicate that for both the code bases of AASW, following any ad hoc way, only one out of three teams submitted and demonstrated the working code. All teams for both code bases simply submitted the requirements as they were given to them without any prioritization, categorization, effort estimation, or traceability. In contrast, the teams who followed the N-methodology for both the code bases performed much better, being able to complete 78-80% of enhancement requirements. All the teams submitted all the listed artifacts, which were evaluated to be of much better quality. The second case study results indicated that students following RUP or AUM could complete only 30 to 50% of the needed enhancements. The team that submitted the working code only submitted a few ISDs and ICDs but did not submit any other artifacts. In contrast, the teams following the N-model methodology performed much better, with 75-80% enhancement completion as shown in TABLE 4.2 cite. All the teams submitted all the listed artifacts after learning the N-model methodology; these artifacts were evaluated to be of much better quality. Thus, the hypothesis that the N-model methodology could be used to produce better results in regards to creating quality artifacts and enhanced software was confirmed.

Table 4.2: Preliminary Case Studies Results

Code Base	Submitted Artifacts	Case Study 1								Case Study 2									
		Teams using any Ad-hoc way				Teams using N-Methodology				Teams using process					Teams using the Methodology				
										RUP		AUM							
Base - A	*Teams →*	*2*	*3*	*4*	*Avg.*	*1*	*5*	*6*	*Avg.*	*1*	*3*	*2*	*4*	*Avg.*	*5*	*6*	*7*	*8*	*Avg.*
	Requirements	x	x	x	100%	x	x	x	100%	x	x	x	x	100%	x	x	x	x	100%
	ISD/DSD	-	-	-	0%	x	x	x	100%	-	-		x	25%	x	x	x	x	100%
	ICD	-	-	x	33%	x	x	x	100%	-	-	x	-	25%	x	x	x	x	100%
	Test cases	-	-	-	0%	x	x	x	100%	-	-	-	-	0%	x	x	x	x	100%
	Traceability	-	-	-	0%	x	x	x	100%	-	-	-	-	0%	x	x	x	x	100%
	Running code	-	-	x	33%	x	x	x	100%	-	x	x	x	75%	x	x	x	x	100%
	Completion %	0	0	10	3 %	60	95	90	82%	0	40	30	35	24%	70	60	80	90	80%
Base - B	*Teams →*	*5*	*6*	*7*	*Avg.*	*2*	*3*	*4*	*Avg.*	*5*	*7*	*6*	*8*	*Avg.*	*1*	*2*	*3*	*4*	*Avg.*
	Requirements	x	x	x	100%	x	x	x	100%	x	x	x	x	100%	x	x	x	x	100%
	ISD/DSD	-	-	-	0%	x	x	x	100%	-	-	x	x	50%	x	x	x	x	100%
	ICD	x	-	-	33%	x	x	x	100%	-	-	x	-	25%	x	x	x	x	100%
	Test cases	-	-	-	0%	x	x	x	100%	-	-	-	-	25%	x	x	x	x	100%
	Traceability	-	-	-	0%	x	x	x	100%	-	-	-	-	25%	x	x	x	x	100%
	Running code	x	-	-	0%	x	x	x	100%	-	x	x	x	25%	x	x	x	x	100%
	Completion %	33	0	0	11%	65	95	75	78%	0	0	25	50	19%	60	70	90	80	75%

" x " – Artifacts submitted and "-" – Artifacts not submitted .

4.1.4 **Limitations**

There are a few limitations to our first two case studies: first, only two AASW legacy code bases were used to compare and study the effectiveness of the methodology. Second, even though the students were inexperienced and the two projects had different code bases with different designs, the domain learning during the first assignment could become advantageous when approaching the second assignment. Finally, the small size of the project lead to results that were too limited to run any statistical analysis.

Despite these limitations, these two preliminary case studies show that the participants performed much better using the proposed N-methodology compared to using RUP, AUM or ad hoc ways.

4.2 Final Quantitative Case Study with Metrics

4.2.1 Design of Final Quantitative Case Study

The final case study aimed to evaluate quantitatively the effect of the N-model methodology on student performance in enhancement projects.

The subjects of this case study were eight graduate students from the Department of Computer Science and Engineering at The University of Texas at Arlington. They enrolled in the elective course CSE 6239 (Advanced Topics in Software Engineering). At the beginning of the course, the students were required to complete a survey of their academic and software development background relevant to the assignments. The purpose of the survey was to identify each student's proficiency in object-oriented programming, UML, Java, JSP, Tomcat and MySQL/relational DBMS. Based on the survey results, the students were divided into two teams with comparable academic and software development backgrounds. Each team comprised of four students. All of the students were asked not to contact students from other teams during both enhancement projects.

The student teams were required to enhance two real-world, web-based legacy systems as part of two team-based assignments. Specifically, in Assignment 1, they were tasked with enhancing MavAppoint, an online software developed for managing academic advising appointments. Students were asked to complete this assignment using their existing knowledge, experience and skills. After Assignment 1, the students were taught the N-model methodology. They were then required to apply the methodology to enhance a study abroad management system (SAMS), developed for the Office of International Education at the university. Information regarding these two legacy systems is shown in Table 4.3. The Figure 4.1 shows the histograms of the lines of code per class and the number of methods per class of both of these legacy systems. The second assignment SAMS is relatively three times bigger in size and complexity compared to MavAppoint. This increase in complexity

Table 4.3: Information about the legacy systems used in the final case study

	Assignment 1	Assignment 2
Legacy software	MavAppoint	SAMS
Programming language	Java, JSP, Javascript	Java, JSP, Javascript
Number of source lines of code	2240	7536
Total number of classes	55	80
Total number of methods	236	663
Database management system	MySQL	MySQL
Web server	Apache Tomcat	Apache Tomcat

and difficulty was intentionally arranged in order to nullify some of the domain learning experience and programming skills teams acquired from the first Mavappoint assignment. Further details of these two assignments is presented in the next two sections. The efficacy of the N-model methodology was evaluated by comparing the results of these two assignments using three categories of ten metrics, which are described in Section 4.2.2. The evaluation results were then used to assess the hypotheses presented above.

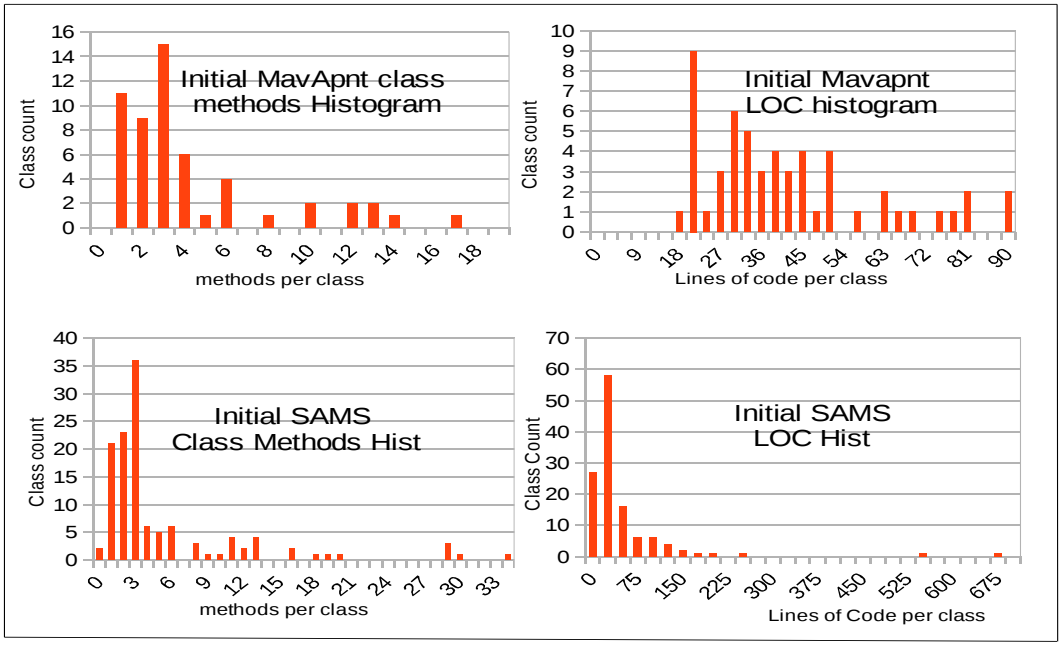


Figure 4.1: Legacy MavAppoint AND SAMS code base histograms

4.2.1.1 **Assignment 1 Description**

In Assignment 1, the two teams were given the source code of MavAppoint, an online academic advising appointment management system. The system allows academic advisors to specify their advising hours and students to make appointments with advisors and cancellations. The two teams were required to modify MavAppoint with the following 12 enhancement requirements using their existing software engineering knowledge, skills and experiences. The teams were given a period of five weeks to complete the assignment. The integer shown at the end of each enhancement requirement represents the task units. This number was the average of the Poker Game estimates [5] produced by the two teams. We used this estimate to measure the number of unit tasks needed to be completed per week to deliver the requirement. This measurement was used to evaluate the N-model methodology in terms of the time-based efficiency metric and overall relative efficiency metric, among other eight evaluation metrics, which are presented in Section 4.2.2.

1. MavAppoint shall prompt the user to change the system-generated temporary password when login is attempted for the first time. (1)
2. MavAppoint shall allow users to upload a picture to his/her profile. (1)
3. MavAppoint shall permit an admin user to set an expiration time for system-generated temporary passwords. (1)
4. MavAppoint shall allow admin users to assign advisors to different academic majors. (1)
5. MavAppoint shall allow an admin user to assign students to advisors according to the first letter of students' last names. (2)
6. MavAppoint shall notify students and advisors about the upcoming appointments one day in advance, and also on the date of the appointments. (1)

7. MavAppoint shall notify relevant advisors and students of any appointment cancellation. (1)
8. MavAppoint shall allow users to select either text message or email or both as the notification method. (2)
9. MavAppoint shall notify students on the waiting list whenever an appointment is canceled. (1)
10. MavAppoint shall let advisors to communicate with all students in a group by sending a broadcast email. (1)
11. MavAppoint shall allow each student to make only one appointment on a particular day and maximum two in a week. (1)
12. MavAppoint shall allow students to create online blogs and participate in them. (1)

4.2.1.2 **Assignment 2 Description**

For Assignment 2, the two teams were required to modify a legacy system called the Study Abroad Management System (SAMS). The system had been developed to support the study abroad exchange program at the university. Like MavAppoint, the system was implemented in Java and Java Server Pages (JSP), and ran on Apache Tomcat and MySQL. Unlike Assignment 1, the two teams of students were taught our N-model methodology and required to apply the methodology to enhance the system with the following 12 requirements. The students were given five weeks to complete this assignment.

1. SAMS shall have a link on the home page with all the upcoming events for the study abroad program and search capability. (1)
2. SAMS shall automatically fill in the contact information if the student has an account and has logged in. The student can change the contact information if needed. (1)
3. SAMS shall allow users to modify the travel abroad program they are currently enrolled in. (1)

4. SAMS shall allow users to enroll in one program per semester. (1)
5. SAMS shall allow students to write or create a blog about their travel abroad program and their experiences. (2)
6. SAMS shall have an emergency message link, which directly sends an email to the Office of International Education director when a student needs urgent help. (1)
7. SAMS shall remind users of upcoming events that they are registered for. (1)
8. SAMS shall notify users about any changes made to the study abroad program for which they are registered. (1)
9. SAMS shall provide users the feature to opt in/opt out of email notification. (1)
10. SAMS shall provide the capability for students to access programs' insurance requirements, facilitate enrollment in travel insurance, vaccination information, along with do's and don'ts for travel to the destination country. (2)
11. SAMS shall allow a user to chat online with an International Office customer service agent. (1)
12. SAMS shall have a link on the home page to search any person by name. (1)

4.2.2 **Evaluation Metrics**

The efficacy of the N-model methodology was evaluated by using three categories of ten metrics. That is, we compared the metrics in Assignment 1 and Assignment 2 to assess the two proposed hypotheses. This section presents the details of these metrics. Data collection and analysis to compute these metrics are described in the following sections.

1. *Process metrics.* These include: (1) time based efficiency, (2) overall relative efficiency taken from ISO/IEC 9126-4 [27], and (3) SUS scale taken from Bangor and colleagues [1]. They were calculated using the data accumulated during the entire enhancement process.

2. *Design and Code Metrics.* These include: (1) requirements coverage metric, (2) weighted methods per class (WMC), (3) depth of inheritance tree (DIT), (4) number of children (NOC), and (5) response for a class (RFC). Metrics (2)-(5) were taken from reference CK-metrics as described by Chidamber [12], who presented six metrics. We had selected only these four because they were supported by an empirical study performed by Bassili, Briand and Melo [3]. The two CK metrics not included in our case studies are coupling between object classes (CBO) and lack of cohesion on methods (LC0M) because the studies revolving around these two did not produce valuable results. The requirements coverage metric calculated the percentage of enhancement requirements that were fulfilled over the total number of enhancement requirements. The other four metrics measured the design and code quality of the software resulting from the enhancement effort. They were calculated using the code submitted by the teams.
3. *Test Metrics.* These include: (1) defect density, and (2) test case success ratio. These metrics were calculated by executing use-case based test cases on the software submitted by the teams and validated by teaching assistant during the project demos.

4.2.2.1 **Process Metrics**

1. *Time Based Efficiency Metric:*

The *time based efficiency* metric measures the number of tasks accomplished per unit time by the team of developers as a whole. This metric was used to evaluate the efficacy of the methodology in terms of work done per unit time. The formula for time based efficiency is:

$$Time\ Based\ Efficiency = (\sum_{j=1}^{R}\sum_{i=1}^{N}\frac{n_{ij}}{t_{ij}})/NR$$

where N is the number of tasks, R is the number of team members, and n_{ij} is the completion status of task i by team member j. If the task is completed successfully by the team member, then n_{ij} is one. If the task is not completed successfully, n_{ij} is zero. All of the tasks performed by all team members should be included. As an example, assume that there were two tasks and four team members. For task 1, the four members spent 30, 15, 25 and 10 time units, respectively, but member 1 did not complete his task successfully. For task 2, they spent 15, 45, 30 and 15 time units but members 2 and 3 did not complete their tasks successfully. The time based efficiency is calculated as follows. That is, the team was able to successfully complete 4.25% of a task per unit time.

$$\frac{(0/30 + 1/15) + (1/15 + 0/45) + (1/25 + 0/30) + (1/10 + 1/15)}{2 * 4} = 0.0425$$

.

2. *Overall Relative Efficiency Metric:*

The *overall relative efficiency* computes the ratio of productive time over the total time spent by the developers to produce the software. The metric was used to measure the efficiency of the methodology in terms of time spent on productive work. The formula for calculating the overall relative efficiency is shown below:

$$Overall\ Relative\ Efficiency = (\sum_{j=1}^{R} \frac{\sum_{i=1}^{N} n_{ij} t_{ij}}{\sum_{i=1}^{N} t_{ij}}) \times 100\%$$

where N and R are the total number of tasks and total number of team members, respectively. In this formula, n_{ij} is zero if member j did not complete task i successfully, or else it is a number in the Fibonacci series 1, 2, 3, 5, 8, 13, 21 with 21 being the cap. The number is determined by the complexity and difficulty for the given task. As an example, assume there were two tasks and four members. For task 1, the four members spent 30, 15, 25 and 10 time units, respectively, but member 1 did not complete his task successfully. For

task 2, they spent 15, 45, 30 and 15 time units but members 2 and 3 did not complete their tasks successfully. Assume that $n_{ij} = 1$ if a team member completes the task successfully. Using the above formula, the overall relative efficiency is

$$\frac{30 * 0 + 15 * 1 + 25 * 1 + 10 * 1}{30 + 15 + 25 + 10} + \frac{15 * 1 + 45 * 0 + 30 * 0 + 15 * 1}{15 + 45 + 30 + 15} = 91.07\%$$

.

3. *SUS Score Metric:*

The *SUS score* is calculated using the feedback from the developers about the methodology they used. There are ten questions, each of which can be scored from one to five, with one representing strongly disagree, two disagree, three neither agree nor disagree, four agree, and five strongly agree. The ten questions in their original form are specified in Figure 4.2. However, in this context the teams were asked to interpret the word "system" as the N-Model Methodology for their second assignment.

For each odd numbered question one is subtracted from the score, and for each even number question the value is subtracted from five. The resulting values are added together, and the sum is then multiplied by 2.5 to get a scale from 0 to 100. Figure 4.3 gives the range of values and what they represent in terms of acceptability ranges, grades and adjective ratings scale on the SUS scale.

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently.	1	2	3	4	5
2. I found the system unnecessarily complex.	1	2	3	4	5
3. I thought the system was easy to use.	1	2	3	4	5
4. I think I need the support of a technical person to be able to use this system.	1	2	3	4	5
5. I found the various functions in this system were well integrated.	1	2	3	4	5
6. I thought there was too much inconsistency in this system.	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly.	1	2	3	4	5
8. I found the system very cumbersome to use.	1	2	3	4	5
9. I felt very confident using the system.	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system.	1	2	3	4	5

Figure 4.2: SUS Questionare

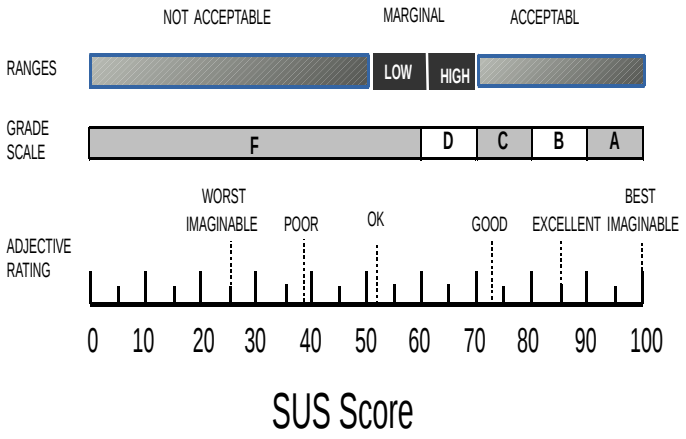


Figure 4.3: SUS scale and implication

4.2.2.2 Design and Code Metrics

The design and code metrics were used to assess the quality of the software produced.

1. *Requirements Coverage Metric:*

The *requirements coverage metric* measures the percentage of requirements covered in the software over the total number of requirements provided to the team. It is computed as follows:

$$Requirements\ Coverage = \frac{number\ of\ requirements\ covered}{total\ number\ of\ requirements}$$

The requirements coverage metric was calculated using the code submitted by the teams as well as during the software demonstration to the teaching assistant. The other four design and code metrics are defined as follows [12] and calculated based on the source code submitted by the student teams for assignments 1 and 2.

2. *Weighted Methods per Class (WMC) Metric:*

The *weighted methods per class (WMC)* metric is the summation of the cyclomatic complexity of all of the methods of a class. The cyclomatic complexity was originally proposed by McCabe [40]. The cyclomatic complexity of a function is the number of atomic binary conditions of the function plus 1. It represents the number of independent control flow paths of the function. For example, the cyclomatic complexity of a function with no conditional statement is 1; this means that there is only one control flow in the function. The higher the number of WMC, the more effort is needed to comprehend, test, maintain and reuse the class. The WMC metric has the following formula:

$$WMC = \sum_{i=1}^{n} cyclomatic\, complexity(m_i)$$

.

where n is the number of methods of the class and m_i denotes the ith method. The cyclomatic complexity of a method is the number of binary atomic conditions in the method plus 1. An atomic n-ary condition can be converted into n-1 binary atomic conditions. There are other ways to calculate the complexity by counting the number of nodes, edges and connected paths as specified in [60] explained below:

Mathmatically the cyclomatic complexity (M) of a structure program is defined as:
M = E - N + P

Where,

E = the number of edges of the graph

N = the number of nodes of the graph

P = the number of connected components

For a single program(or method) P always equals to 1.

So M = E - N + 2

In the small example program taken from cyclomatic wiki [15] as demonstrated in Figure 4.4, the cyclomatic complexity is calculated as M = 8-7 + 2 = 3.

3. *Depth of Inheritance Tree (DIT) Metric:*

The *depth of inheritance tree (DIT)* metric is defined as the height (or depth) of the inheritance tree (in other words, the length of the longest path(s) from the root to any of the leaf node of the inheritance tree). The higher the value of DIT, the more inheritance reuse of the methods among the classes. The deeper a class in the inheritance hierarchy, the greater the number of methods it will likely inherit, which makes it harder to understand

```
if( c1() )
    f1();
else
    f2();

if( c2() )
    f3();
else
    f4();
```

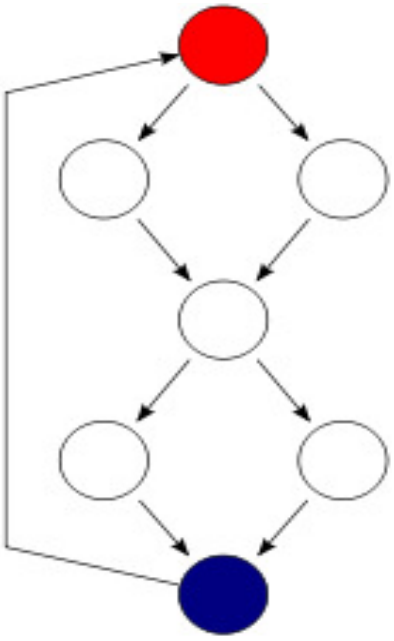


Figure 4.4: Control Flow Graph of a Simple Program for Cyclomatic complexity computation

and predict its behavior. Deeper trees involve greater design complexity since more classes and methods are involved. Deeper classes in the tree have a greater potential for reusing inherited methods.

4. *Number of Children (NOC) Metric:*

The *number of children (NOC)* metric is defined as the number of immediate subclasses subordinated to a class in the class inheritance hierarchy. This metric measures the immediate number of dependents the class has. The higher the value of NOC, the more reuse and more change impact of the class on other classes. If a class has a large number of children, it may be a case of misuse of subclassing. The NOC gives an idea of the potential influence a class has on other classes. If a class has a large NOC, it may require more testing of the methods in that class.

5. *Response for a Class (RFC) Metric:*

The *response for a class (RFC)* metric is the number of distinct methods and constructors invoked by a class as a result of a message sent to an object of the class. If a large number of methods are invoked in response to a message, the comprehension, testing, debugging and maintenance of the class becomes more difficult.

4.2.2.3 **Test Metrics**

1. *Defect Density Metric:*

The *defect density* metric measures the number of defects per one thousand lines of enhanced code. The enhanced lines of code includes all the newly added, modified and deleted source lines and excludes all blank and comment lines. A good methodology should have a low defect density. This metric is formulated as:

$$Defect\ Density = \frac{Total\ number\ of\ defects}{Total\ number\ of\ enhanced\ lines\ of\ codes(in\ KLOC)}$$

2. *Ratio of Success Tests metric:*

The *ratio of success tests* is defined as the ratio of success tests over the total number of tests. For this final case study, the test cases were prepared by the TA to ensure that the enhancement requirements were satisfied. The formula for this metric is:

$$Ratio\ of\ Success\ Tests = \frac{Number\ of\ success\ tests}{Total\ numberof\ tests\ executed}$$

4.2.3 Data Collection

This section presents the data collected for evaluating the ten metrics. Data collection was completed in two phases for each of the assignments — one before and the other after the demonstration of the software to the Teaching Assistant (TA). Data collection was accomplished by asking the students to fill out respective Excel sheets provided to the students by the TA.

Data collection for process metrics: In Assignment 1, the two student teams were asked to use their existing software engineering knowledge, skills and experiences to complete the assignment. In order to compare the time-based efficiency and overall relative efficiency metrics from the two assignments, we defined a task as a measure of a requirement with an estimated effort of one unit. These estimates were shown at the end of the enhancement requirements presented in Sections 4.2.1.1 and Section 4.2.1.2. The effort estimate for each of the enhancement requirements was collected from the teams as the Poker Game estimate [5] and then normalized by taking the average of the two teams. For the time-based efficiency and relative time efficiency metrics, the individual team members were asked to submit the time spent on each requirement and its completion status in an Excel sheet. If a requirement involving multiple members was not completed successfully, determined at the time of software demonstration to the TA, then the completion status for this requirement for all of the members involved were set to zero. In other words, the members had spent the time working on the requirement but the work was not done.

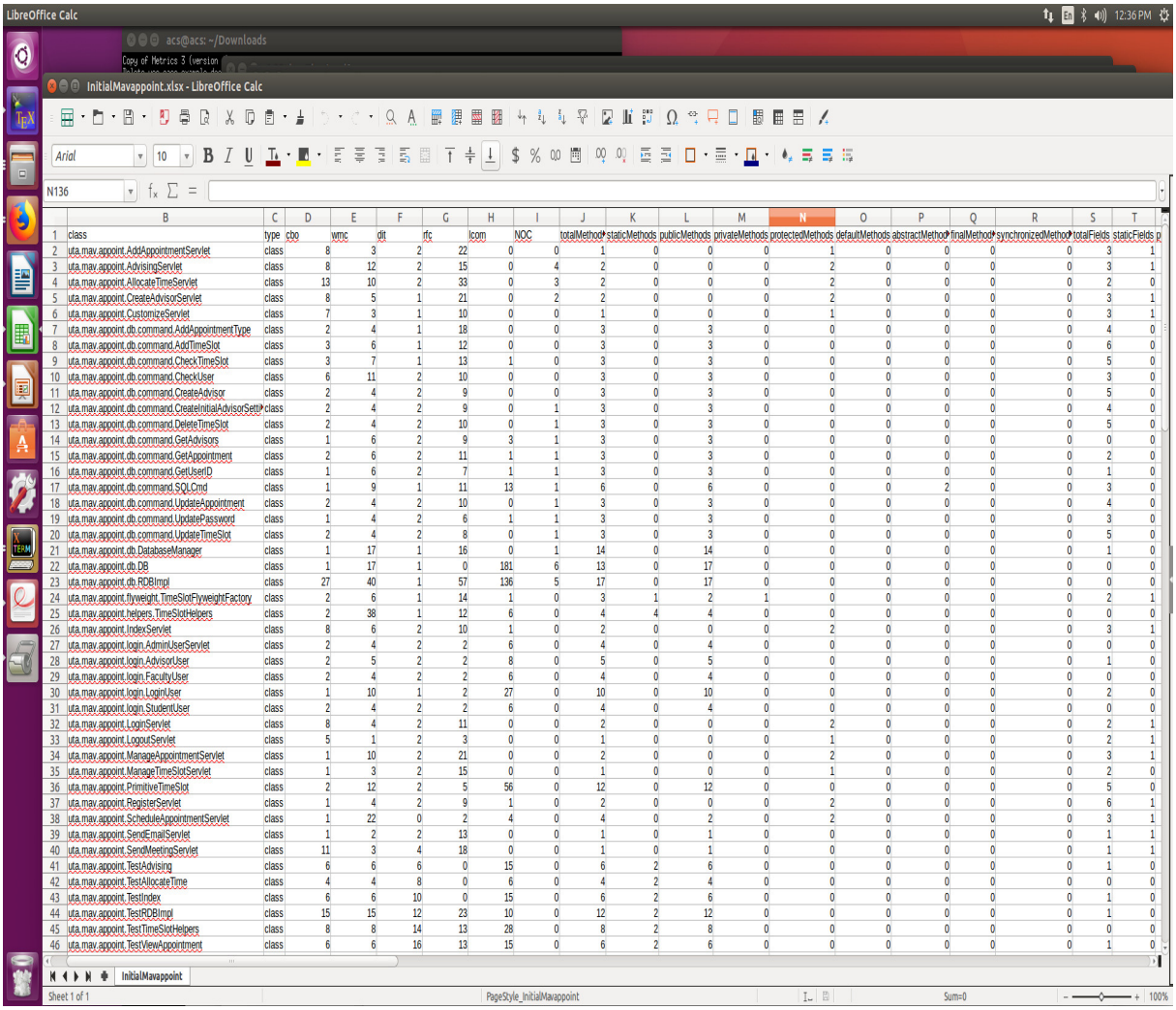
For the SUS scale part of the process metric, each team member was required to answer the ten SUS-scale questions presented in Section 4.2.2. Specifically, they were asked to answer these questions with a satisfaction level of 1 to 5 and submit the answers to the TA in an Excel sheet.

Data collection for design and code metrics: For both Assignment 1 and Assignment 2, each team was required to submit a list of requirements covered along with esti-

Table 4.4: Overall size statistics of legacy and enhanced systems for MavAppoint and SAMS

	MavAppoint			SAMS		
	Original	Team 1	Team 2	Original	Team 1	Team 2
Number of source lines of code	2240	3461	3413	7536	8846	8245
Total number of classes	55	70	70	80	135	115
Total number of methods	236	343	332	663	779	734

mated task units, which had been described previously. The list of requirements covered was verified by the TA during the software demonstration. If necessary, the submitted code was examined to confirm that a requirement was indeed implemented. Moreover, the teams were also mandated to submit the enhanced code to compute the four selected CK metrics using a ck-metrics tool [39]. The tool is a Java program which goes through all the files in a project from its root directory and find all the classes. It then computes all of the ck-metrics per each class in addition to computing the number of methods, lines of code, etc. per class basis and generates a comma separated (CSV) file. The CSV file can then be imported to Excel for additional computation needed for the metrics. An example of a generated comma separated file output from the CK-tool is shown in Figure 4.5. The overall size of the statistics of all the submitted code is specified and compared with original legacy code in the Table 4.4. The histograms of legacy MavAppoint, legacy SAMS, enhanced MavAppoint for team-1, enhanced MavAppoint for team-2, enhanced SAMS for team-1, and enhanced SAMS for team-2 code bases are given in Figure 4.6, Figure 4.7, Figure 4.8, Figure 4.9, Figure 4.10 and 4.11 respectively. Each of these figures depicts the histograms for WMC, RFC, per class methods, and per class lines of codes for their respective code base. In each of these histograms, the x-axis represents the value of metrics and the y-axis represents the number of classes.



	class	type	cbo	wmc	dit	rfc	lcom	NOC	totalMethod	staticMethods	publicMethods	privateMethods	protectedMethods	defaultMethods	abstractMethod	finalMethod	synchronizedMethod	totalFields	staticFields
2	uta.mav.appoint.AddAppointmentServlet	class	8	3	2	22	0	0	1	0	0	0	1	0	0	0	0	3	1
3	uta.mav.appoint.AdvisingServlet	class	8	12	2	15	0	4	2	0	0	0	2	0	0	0	0	3	1
4	uta.mav.appoint.AllocateTimeServlet	class	13	10	2	33	0	3	2	0	0	0	2	0	0	0	0	2	0
5	uta.mav.appoint.CreateAdvisorServlet	class	8	5	1	21	0	2	2	0	0	0	2	0	0	0	0	3	1
6	uta.mav.appoint.CustomizeServlet	class	7	3	1	10	0	0	1	0	0	0	1	0	0	0	0	3	1
7	uta.mav.appoint.db.command.AddAppointmentType	class	2	4	1	18	0	0	3	0	3	0	0	0	0	0	0	4	0
8	uta.mav.appoint.db.command.AddTimeSlot	class	3	6	1	12	0	0	3	0	3	0	0	0	0	0	0	6	0
9	uta.mav.appoint.db.command.CheckTimeSlot	class	3	7	1	13	1	0	3	0	3	0	0	0	0	0	0	5	0
10	uta.mav.appoint.db.command.CheckUser	class	6	11	2	10	0	0	3	0	3	0	0	0	0	0	0	3	0
11	uta.mav.appoint.db.command.CreateAdvisor	class	2	4	2	9	0	0	3	0	3	0	0	0	0	0	0	5	0
12	uta.mav.appoint.db.command.CreateInitialAdvisorSetti	class	2	4	2	9	0	1	3	0	3	0	0	0	0	0	0	4	0
13	uta.mav.appoint.db.command.DeleteTimeSlot	class	2	4	2	10	0	1	3	0	3	0	0	0	0	0	0	5	0
14	uta.mav.appoint.db.command.GetAdvisors	class	1	6	2	9	3	1	3	0	3	0	0	0	0	0	0	0	0
15	uta.mav.appoint.db.command.GetAppointment	class	2	6	2	11	1	1	3	0	3	0	0	0	0	0	0	2	0
16	uta.mav.appoint.db.command.GetUserID	class	1	6	2	7	1	1	3	0	3	0	0	0	0	0	0	1	0
17	uta.mav.appoint.db.command.SQLCmd	class	1	9	1	11	13	1	6	0	6	0	0	0	2	0	0	3	0
18	uta.mav.appoint.db.command.UpdateAppointment	class	2	4	2	10	0	1	3	0	3	0	0	0	0	0	0	4	0
19	uta.mav.appoint.db.command.UpdatePassword	class	1	4	2	6	1	1	3	0	3	0	0	0	0	0	0	3	0
20	uta.mav.appoint.db.command.UpdateTimeSlot	class	2	4	2	8	0	1	3	0	3	0	0	0	0	0	0	5	0
21	uta.mav.appoint.db.DatabaseManager	class	1	17	1	16	0	1	14	0	14	0	0	0	0	0	0	1	0
22	uta.mav.appoint.db.DB	class	1	17	1	0	181	6	13	0	17	0	0	0	0	0	0	0	0
23	uta.mav.appoint.db.RDBImpl	class	27	40	1	57	136	5	17	0	17	0	0	0	0	0	0	0	0
24	uta.mav.appoint.flyweight.TimeSlotFlyweightFactory	class	2	6	1	14	1	0	3	1	2	1	0	0	0	0	0	2	1
25	uta.mav.appoint.helpers.TimeSlotHelpers	class	2	38	1	12	6	0	4	4	4	0	0	0	0	0	0	0	0
26	uta.mav.appoint.IndexServlet	class	8	6	2	10	1	0	2	0	0	0	2	0	0	0	0	3	1
27	uta.mav.appoint.login.AdminUserServlet	class	2	4	2	2	6	0	4	0	4	0	0	0	0	0	0	0	0
28	uta.mav.appoint.login.AdvisorUser	class	2	5	2	2	8	0	5	0	5	0	0	0	0	0	0	1	0
29	uta.mav.appoint.login.FacultyUser	class	2	4	2	2	6	0	4	0	4	0	0	0	0	0	0	0	0
30	uta.mav.appoint.login.LoginUser	class	1	10	1	2	27	0	10	0	10	0	0	0	0	0	0	2	0
31	uta.mav.appoint.login.StudentUser	class	2	4	2	2	6	0	4	0	4	0	0	0	0	0	0	0	0
32	uta.mav.appoint.LoginServlet	class	8	4	2	11	0	0	2	0	0	0	2	0	0	0	0	2	1
33	uta.mav.appoint.LogoutServlet	class	5	1	2	3	0	0	1	0	0	0	1	0	0	0	0	2	1
34	uta.mav.appoint.ManageAppointmentServlet	class	1	10	2	21	0	0	2	0	0	0	2	0	0	0	0	3	1
35	uta.mav.appoint.ManageTimeSlotServlet	class	1	3	2	15	0	0	1	0	0	0	1	0	0	0	0	2	0
36	uta.mav.appoint.PrimitiveTimeSlot	class	2	12	2	5	56	0	12	0	12	0	0	0	0	0	0	5	0
37	uta.mav.appoint.RegisterServlet	class	1	4	2	9	1	0	2	0	0	0	2	0	0	0	0	6	1
38	uta.mav.appoint.ScheduleAppointmentServlet	class	1	22	0	2	4	0	4	0	2	0	2	0	0	0	0	3	1
39	uta.mav.appoint.SendEmailServlet	class	1	2	2	13	0	0	1	0	1	0	0	0	0	0	0	1	1
40	uta.mav.appoint.SendMeetingServlet	class	11	3	4	18	0	0	1	0	1	0	0	0	0	0	0	1	1
41	uta.mav.appoint.TestAdvising	class	6	6	6	0	15	0	6	2	6	0	0	0	0	0	0	1	0
42	uta.mav.appoint.TestAllocateTime	class	4	4	8	0	6	0	4	2	4	0	0	0	0	0	0	0	0
43	uta.mav.appoint.TestIndex	class	6	6	10	0	15	0	6	2	6	0	0	0	0	0	0	1	0
44	uta.mav.appoint.TestRDBImpl	class	15	15	12	23	10	0	12	2	12	0	0	0	0	0	0	1	0
45	uta.mav.appoint.TestTimeSlotHelpers	class	8	8	14	13	28	0	8	2	8	0	0	0	0	0	0	0	0
46	uta.mav.appoint.TestViewAppointment	class	6	6	16	13	15	0	6	2	6	0	0	0	0	0	0	1	0

Figure 4.5: CK Tool generated sample csv file opened in excel

Data collection for test metrics: The teams were asked to submit the number of defects found. The number of lines of enhanced source code was computed from the submitted code base. The *defect density metric* was then computed as the ratio of the number of defects over the number of thousand lines of enhanced code (KLOC). The enhanced lines of code includes all the newly added, modified and deleted source lines. The comments and blank lines are not included in the computation. The command line tool called "cloc" was used to calculate enhanced lines of codes by comparing the legacy code base

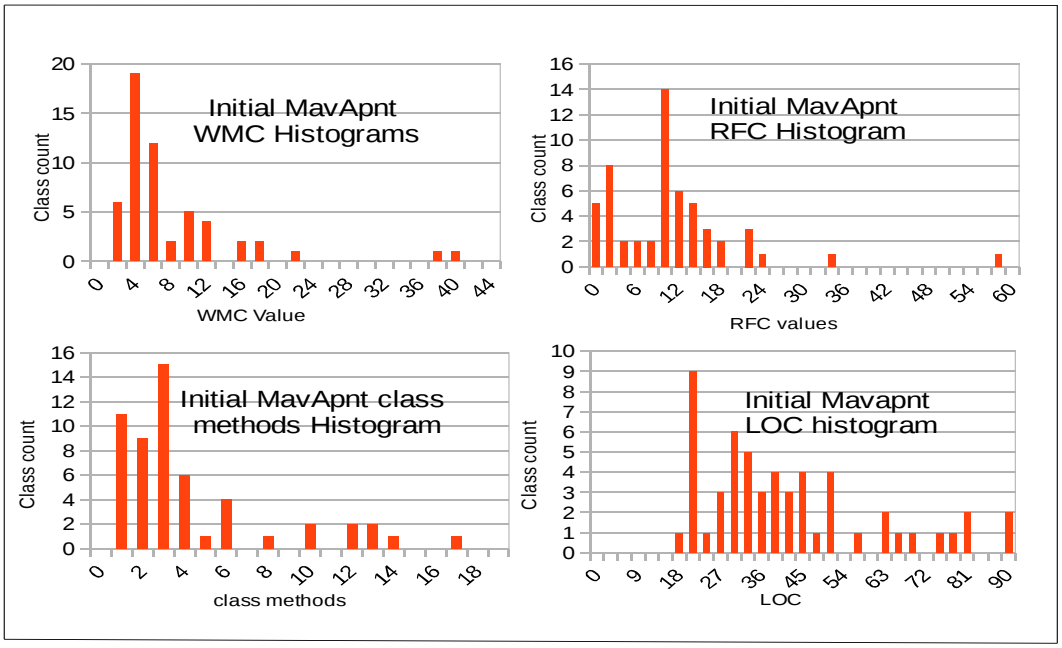


Figure 4.6: Histogram for the Legacy MavAppoint code base

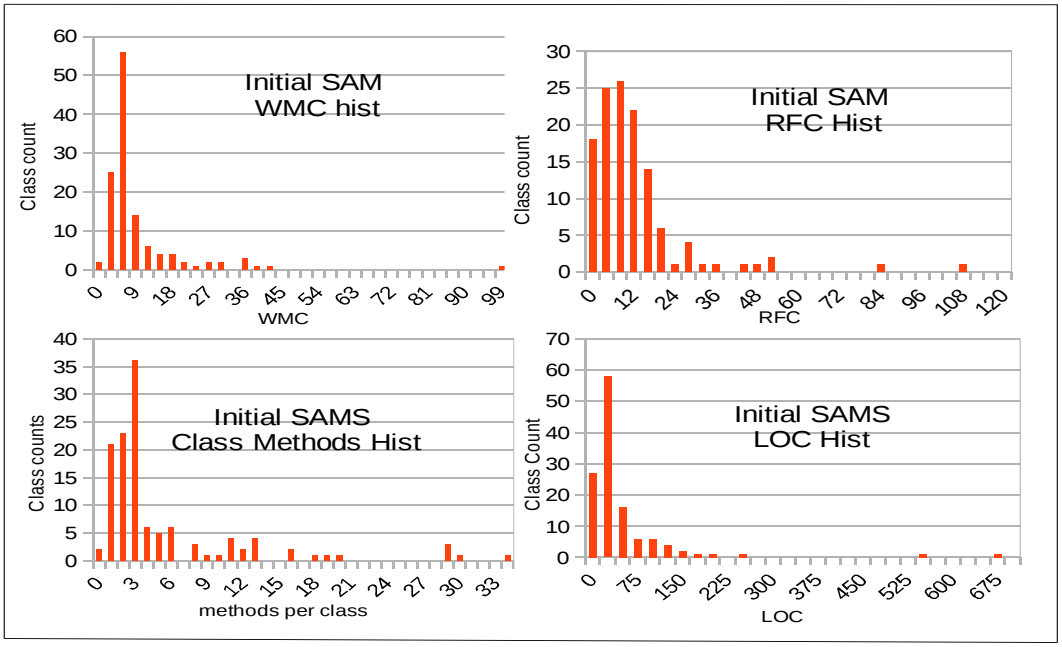


Figure 4.7: Histogram for the Legacy SAMS code base

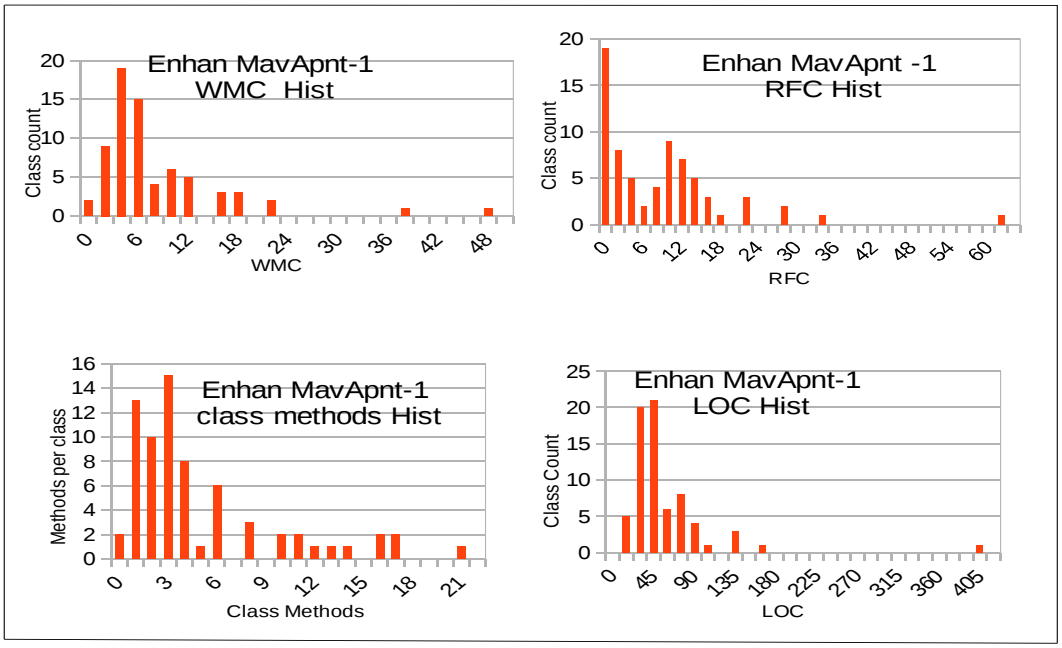


Figure 4.8: Histogram for the Enhanced MavAppoint code for Team 1

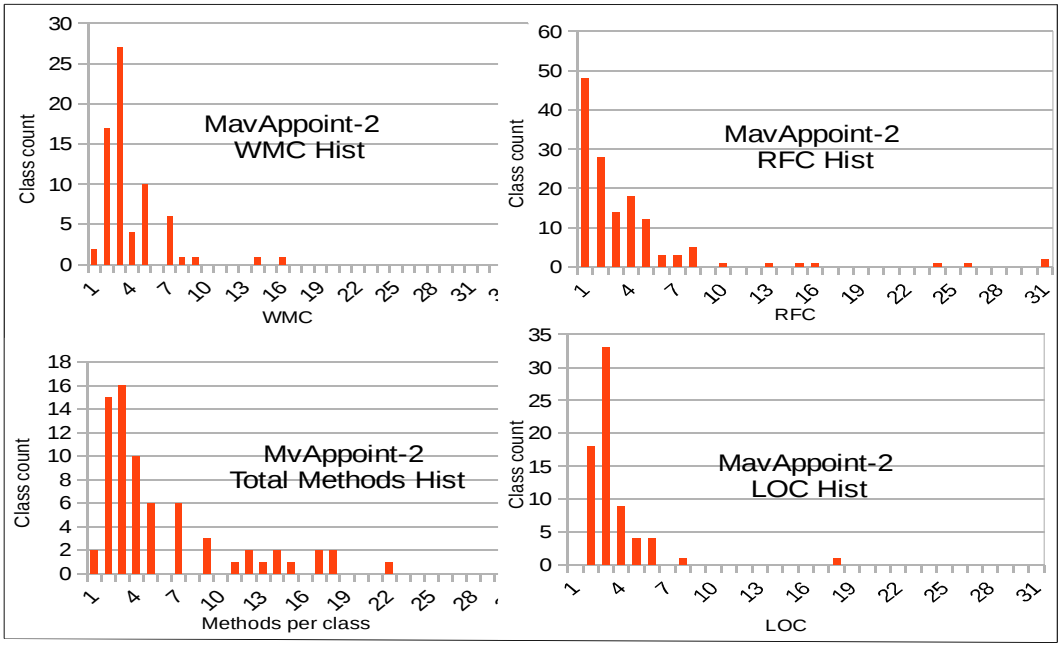


Figure 4.9: Histogram for the Enhanced MavAppoint Code for Team 2

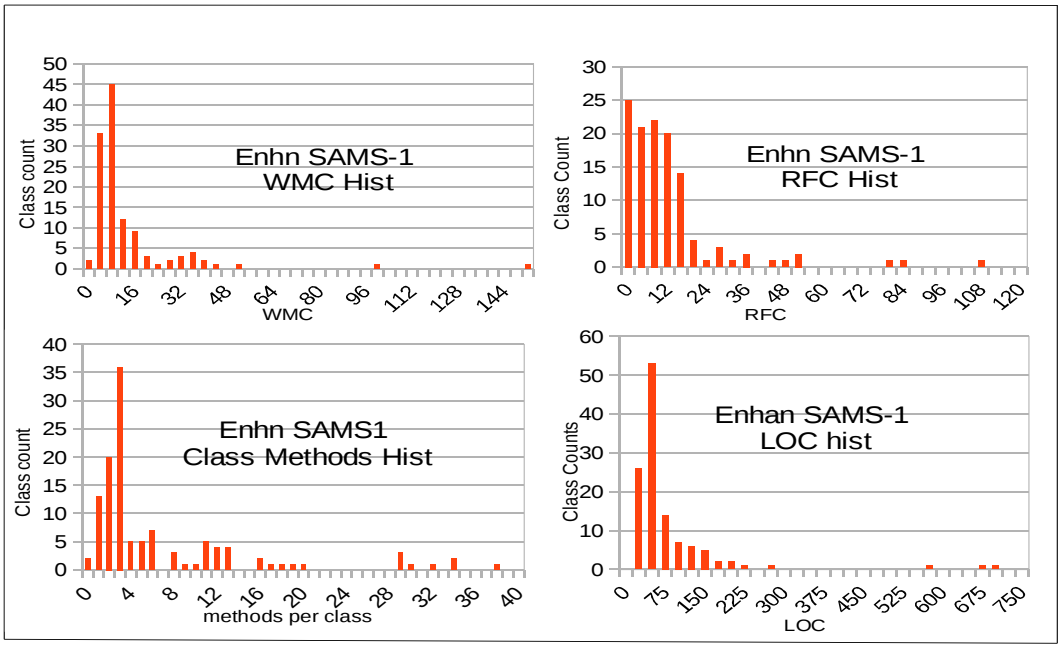


Figure 4.10: Histogram for the Enhanced SAMS code for Team 1

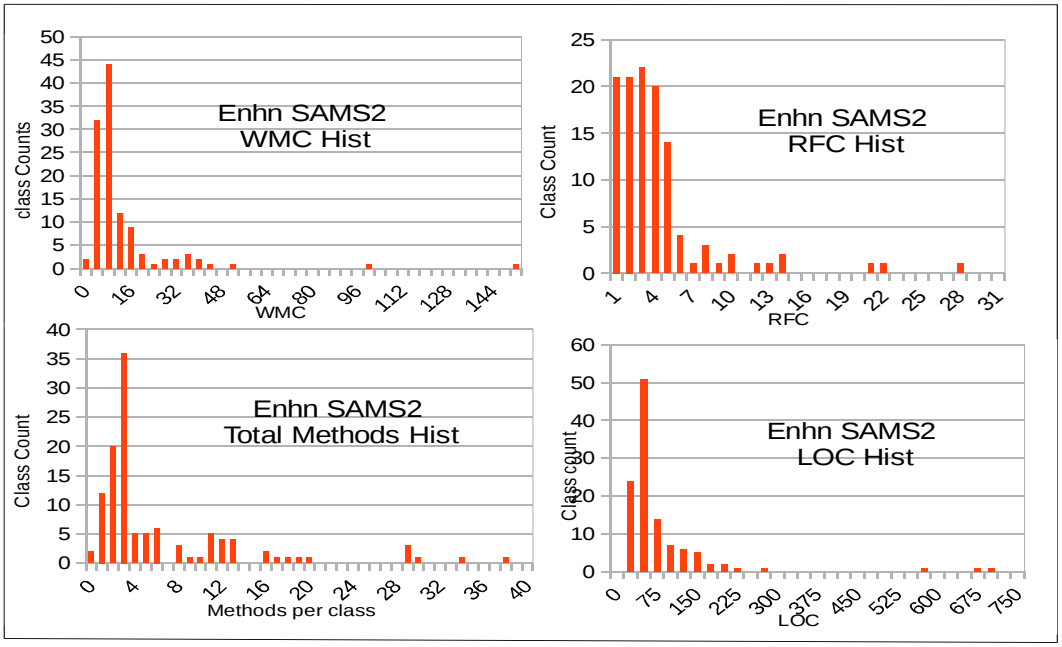


Figure 4.11: Histogram for the Enhanced SAMS Code for Team 2

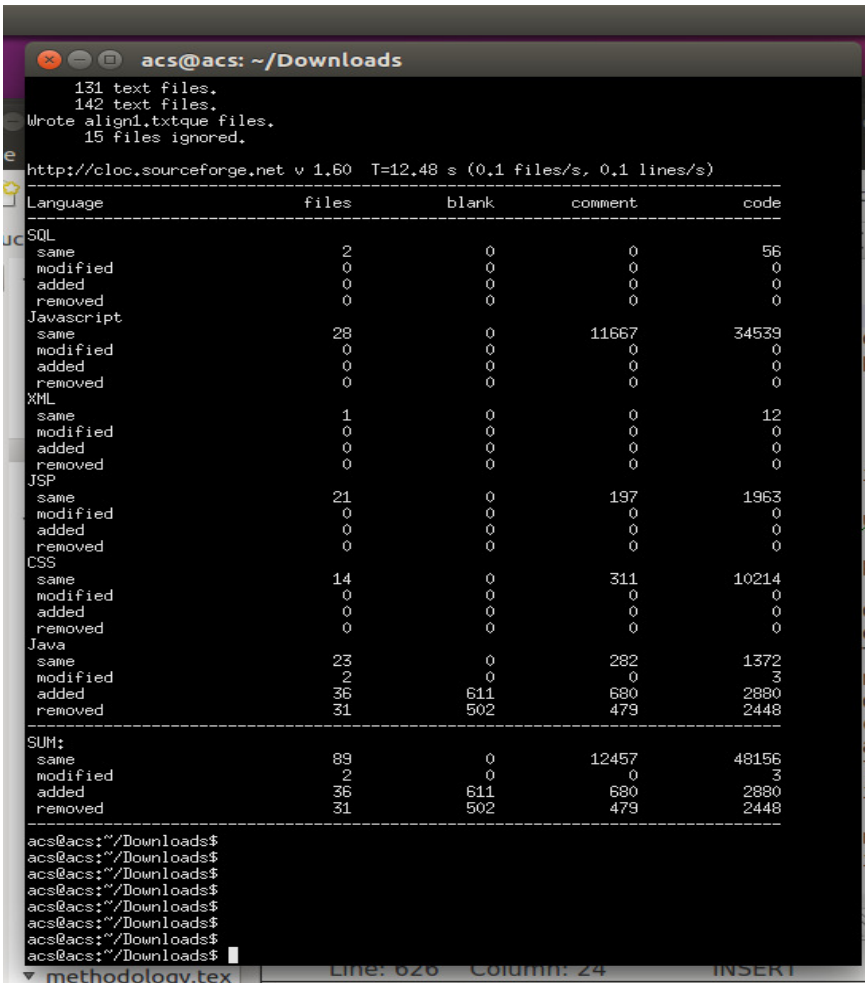


Figure 4.12: Sample output of cloc Tool to Calculate Enhanced Lines of Code

with that of the enhanced version. Cloc is a freeware available in almost all the platforms. We utilized Unix version to compute the enhanced lines of code. It requires two directories or zipfiles as inputs and needs to be run with a "-diff" option. An example screen dump is given in Figure 4.12. As shown, Cloc compares the two versions of code base before and after enhancements and calculates the lines of code categories as; "same", "modified", "added" and "deleted". The total lines of code for each category for the whole project is summarized at the end. The modified, added and deleted categories were summed up for the enhanced lines of code and then divided by 1000 for the defect density calculation.

For the *ratio of successful test cases*, the teams were asked to submit test cases and their pass/fail/not-run results. The list of test cases, being one of the mandatory artifacts, was validated against the TA's list of test cases to make sure that each team had 100% test case coverage that traced all the way to the given enhancement requirements. As an example, the following was one of the validation test cases used during data collection, where CAPTCHA stands for "completely automated public Turing test to tell computers and humans apart."

Precondition:	Run the application to get the initial home page.
Test step 1:	Enter correct user name in the user name field.
Test step 2:	Enter correct password in the password field.
Test step 3:	Read the system generated CAPTCHA* and enter it into the CAPTCHA field.
Test step 4:	Click on the Login button.
Expected result:	User should have been logged in and redirected to the Advising Scheduling page.
Actual Result:	Specify the execution result with respect to the expected result.
Test case status:	Specify "Passed," "Failed," or "not-run."

4.2.4 Evaluation Results

The N-model methodology was evaluated using the data collected from the two assignments. The data were then used to compute the ten metrics for each of the two assignments. Table 4.5 gives the evaluation results of all except the CK metrics, while Table 4.6 gives the evaluation results for the CK metrics. Table 4.5 and shows that time-based efficiency, overall relative efficiency, requirements coverage, defect density and test case

success ratio metrics were significantly improved after the students learned the N-model methodology. The time base team efficiency improvement was observed to be .39 task per week for team-1 and 1.1 tasks per week for Team-2 after learning the N-model methodoloy. Overall relative efficiency improved over 50% for both the teams due to the systematic approach of the N-model methodology. Moreover, the SUS scores were also much higher, reflecting students' satisfaction with using the N-model methodology compared to their own approaches. The SUS score increase of 70% for team-1 and 61% for team-2 is likely due to the fact that the students did not have anything to follow in assignment-1 in contrast to following the N-model methodology in assignment-2. The defect density per 1000 lines of enhanced code (KLOC) also dropped by ten and five points for team-1 and team-2, respectively. So was the case for test case success rate improvements of 16% for team-1 and 38% for team-2 by using N-model methodology. Thus, the alternative hypothesis(H_1), "that the N-Model methodology produces better result" is supported by these metrics results.

Table 4.5: Process and test metrics for assignments 1 and 2

	Team 1			Team 2		
	Asgmt 1	Asgmt 2		Asgmt 1	Asgmt 2	
	MavAppoint	SAMS	Improve	MavAppoint	SAMS	Improve
Time based efficiency (task units per week)	1.42	1.81	0.39	1.10	2.20	1.10
Overall relative efficiency	36%	89%	53%	30%	82%	52%
SUS Score	15.00	85.00	70.00	25.00	86.43	61.43
Requirements coverage	50%	69%	19%	42%	89%	47%
Defect density (Per 1000 lines of code)	47	37	-10	44	39	-5
Test case success ratio	50%	66%	16%	42%	80%	38%

For the CK metrics, all the data were captured using the CK Metrics tool [39]. The selected histograms for all the submitted versions of the code before and after enhancements are shown from Figures 4.6 to Figures 4.11. For the original MavAppoint legacy code, most of the class sizes are around 25 lines of codes with two outliers of maximum 98 lines. The most frequent (around 18 classes) WMC values were found to be around five, with a max of 40 for two classes. The RFC values peaked around 12, with one outlier of 60. In contrast, in the initial SAMS code , most of the class sizes were around 35 lines of codes with one 650 maximum 680 lines.The most frequent (around 18 classes) WMC values were found to be around eight, with a max of 99. The RFC values peaked around 12, with some values at 48, 84 and 108. The histograms show that the initial SAMS project was a bit larger and more complex than MavAppoint. Both enhancement projects for both the teams grew proportionately as shown in histograms from Figures 4.8 to Figures 4.11.

Table 4.6 shows the summary of CK metrics results in terms of computed average and standard deviation for the original MavAppoint and SAMS legacy software as well as their enhanced versions of the software applications submitted by the teams. We see that the CK metrics give us mixed results. The weighted methods per class (WMC) metrics for the enhanced SAMS versions are bit higher than their original counterparts when compared to MavAppoint. In particular, for the enhanced versions, the WMC metrics had increased from 8.85 to 10.41 and 11.38 for SAMS, as compared to the marginal increase for MavAppoint. These marginal increases could be justified by students focusing on implementing more logics and finishing more requirements using the same methods rather than focusing on reducing their cyclomatic complexities. The significant increases in requirements coverages, test case success ratios as well as the reduction of defect density as shown in Table 4.5 justifies this effect. The depth of inheritance remained more or less the same, with a marginal increase from 1.85 to 1.86 for team-2 for SAMS. This marginal increase likely indicates more code reuse. The number of children (NOC) metric significantly in-

Table 4.6: Evaluation results for the CK metrics

	MavAppoint			SAMS		
	Original	Team 1	Team 2	Original	Team 1	Team 2
Weighted Methods per Class (WMC)	7.76	7.81	7.59	8.85	10.41	11.38
StDev	7.50	7.76	7.70	11.42	16.86	17.93
Depth of Inheritance Tree (DIT)	1.49	1.60	1.50	1.85	1.78	1.86
StDev	0.50	0.49	0.67	1.07	1.05	1.10
Number of Children(NOC)	0.93	1.83	1.68	0.56	2.05	2.14
StDev	1.46	3.76	3.51	1.40	3.10	3.38
Response for Class (RFC)	10.44	8.39	8.30	10.80	11.00	11.59
StDev	9.62	10.06	9.75	14.97	15.75	16.68

creased by four fold for SAMS for both the teams. This increase indicates a good amount of code reuse, as well. The response for the class (RFC) marginally increased for enhanced SAMS, which again could be justified by the fact that the team focused on getting more requirements implemented within a limited amount time. Again, overall the alternative hypothesis(H_1), "that the N-Model methodology produces better result" is supported by these ten metrics results.

4.2.5 **Threats to Validity**

Several threats to the external validity of this final case study may limit the generalization of the experiment results:

1. This case study evaluation was performed on small programs i.e., 2240 lines of code, 55 classes and 236 methods for MavAppoint, and 7536 lines of code, 80 classes and 663 methods for SAMS. They are small when compared to large industry software systems. The evaluation results may not be valid for larger industry systems.
2. MavAppoint and SAMS were rather limited in conceptual complexity and functionality as compared to the large, complex industry software. Large, complex systems may not share the same experimental result.

3. The participants of this case study evaluation were eight graduate students, divided into two teams of four students each. Their software development training and experiences were very limited when compared to seasoned software developers. The small class size of eight students was also a drawback to the external validity of the study.
4. The time allocated to the two teams to complete each of the two assignments was five weeks. Moreover, we anticipated that each student would spend approximately ten hours per week working on the project. This timeline is very different from the software development environments in industry. The computed metrics could be very different if the case studies were performed in an industry setting.
5. The assignments were part of the academic course work. Even though the teams were repeatedly told that the data submitted to evaluate the N-model methodology by the teams would no way affect their individual grades, there may be some inaccuracies or biases which could not be accurately validated by the teaching assistant. For example, the reported individual time spent for the assigned task and answers to SUS questionnaires fall into these categories.
6. Both MavAppoint and SAMS applications use similar Java, JSP, MySQL and Tomcat based web technologies. There may be some technology learning experience from the first MavAppoint assignment benefiting the students when completing the second assignment SAMS. However, the size and complexity of the second assignment being larger, as described previously, would have compensated at least in part for this effect.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Enhancement is an important part of software maintenance in the software industry, in which the environment is constantly evolving and customer needs are ever-changing. Software maintenance consumes an average of 60% of software life costs, of which more than 60% is spent on enhancements. These costs present a challenge to the software community, who must deal with enhancing millions of lines of legacy code where documentation is inadequate or non-existent. Thus, software enhancement would sorely benefit from a process and methodology. Unfortunately systematic methodologies for object oriented software enhancement currently does not exist in the literature. This thesis presented an agile N-Model process and methodology to enhance object-oriented software, which focused on a front end quick planning phase, an iterative development phase, and a system validation phase. The application of the N-Model process and the methodology on several qualitative and a quantitative case studies resulted in several indications of improved quality and schedule over doing it in any ad-hoc manner or utilizing any forward engineering techniques. The future work that remains to be completed involves using automated tools to recover a higher level of abstraction such as the domain model from the recovered ICD and ISD iteratively. The agility of the methodology can also be improved by automating the manual steps and integrating them with existing reverse engineering and enhancement phases. With the advancement of machine learning and expert systems, the areas such as use case identification from enhance requirements, use case categorization, work effort estimation, change impact identification, automatic test case generation and test automation, metrics calculation for instant feedback etc. could be automated to help further improve the

utility and agility of the methodology. Finally, even though the case studies were conducted in classroom environments, the expectation is that the N-model methodology would also make a significant impact in software industry, where there is a dire need for a methodology to enhance object oriented legacy software.

REFERENCES

[1] Aaron Bangor , Philip Kortum , James Miller,"Determining what individual SUS scores mean: adding an adjective rating scale, Journal of Usability Studies," v.4 n.3, p.114-123, May 2009.

[2] Alkhalid, A., M. Alshayeb, and S. A. Mahmoud. "Software Refactoring at the Package Level using Clustering Techniques." IET Software, vol. 5, no. 3, 2011.

[3] Victor R. Bassili, Lionel C. Briand, and Walcelio L. Melo, "A validation of object-oriented design metrics as quality indicators," IEEE Transactions on Software Engineering, Vol. 22, No. 10, October 1996. pp. 751-761.

[4] Kent Beck, "Test Driven Development: By Example," Addison-Wesley Professional, 2002.

[5] Kent Beck, "Extreme Programming Explained: Embrace change," 2nd Edition, Addison-Wesley, 2004.

[6] Michael R. Blaha and James R Rumbaugh, "Object-Oriented Modeling and Design with UML (2nd Edition)," Prentice Hall, 2004.

[7] J. Borchers, "Invited Talk: Reengineering from a Practitioner's View – A Personal Lesson's Learned Assessment," 15th European Conference on Software Maintenance and Reengineering (CSMR), 2011. pp. 1-2.

[8] Briand, Basili, Yong-Mi Kim, et al. "A Change Analysis Process to Characterize Software Maintenance Projects", Proceedings 1994 International Conference on Software Maintenance, 1994.

[9] Bernd Bruegge and Allen H. Dutoit, "Object-Oriented Software Engineering: Using UML, Patterns, and Java," 3nd Edition, Prentice Hall, 2009.

[10] M.I. Cagnin, R. Penteado, R.T.V. Braga and P.C. Masiero, “Reengineering using design patterns, ” Reverse Engineering, 2000. Proceedings. Seventh Working Conference on, 2000., pp. 118-127.

[11] M.I. Cagnin and J.C. Maldonado, “PARFAIT: towards a framework-based agile reengineering process,” Proceedings of the Agile Development Conference (ADC), 2003., pp. 22-31.

[12] Chidamber, S. R., and C. F. Kemerer. “A Metrics Suite for Object Oriented Design,” IEEE Transactions on Software Engineering, vol. 20, no. 6, 1994. pp. 476-493.

[13] P. Claudia, M. Liliana and F. Liliana, “Recovering Use Case Diagrams from Object Oriented Code: An MDA-based Approach,” Eighth International Conference on Information Technology: New Generations (ITNG), 2011. pp. 737-742.

[14] Alistair Cockburn, “Agile Software Development: The Cooperative Game,” Addison-Wesley Professional, 2006.

[15] Wikipedia, “Cyclomatic complexity wiki page,” https://en.wikipedia.org/wiki/cyclomatic_complexity

[16] J.M. deBaud, S. Rugaber, “A software re-engineering method using domain models,” Software Maintenance, 1995. Proceedings., International Conference on, 1995., pp. 204-213.

[17] Philippe Dugerdil, “Using RUP to reverse-engineer a legacy system,” https://www.ibm.com/developerworks/rational/library/sep06/dugerdil/index.html.

[18] G. Ebner and H. Kaindl, “Tracing all around in reengineering,” IEEE Software, vol. 19, no. 3, 2002. pp. 70-77.

[19] T. Gane and C. Sarson, “Structured Systems Analysis: Tools and Techniques,” Prentice-Hall, 1978.

[20] R.L. Glass, “Frequently forgotten fundamental facts about software engineering,” IEEE Software, vol. 18, no. 3, 2001., pp. 112-111.

[21] David Grove , Greg DeFouw , Jeffrey Dean , Craig Chambers, “Call graph construction in object-oriented languages, ” Proceedings of the 12th ACM SIGPLAN conference on ObjecGrov97at-oriented programming, systems, languages, and applications, p.108-124, October 05-09, 1997, Atlanta, Georgia, USA [doi>10.1145/263698.264352]

[22] M. Hong, T. Xie and F. Yang, “JBOORET: an automated tool to recover OO design and source models,” 25th Annual International Computer Software and Applications Conference (COMPSAC), 2001., pp. 71-76.

[23] Hyland-Wood, D., D. Carrington, and S. Kaplan. “Towards a Software Maintenance Methodology using Semantic Web Techniques and Paradigmatic Documentation Modelling”, IET Software, vol. 2/no. 4, 2008.

[24] IBM tool, “Rational Rose Architectl,” Available:http://www.ibm.com/software/products/en/ratisoftarch.

[25] Christou, Ioannis, Stavros Ponis, and Eleni Palaiologou. “Using the Agile Unified Process in Banking”, IEEE Software, vol. 27/no. 3, 2010.

[26] Ivar Jacobson, James Rumbaugh and Grady Booch, “Unified Software Development Process,” Addison-Wesley, 1999.

[27] Jordan, Patrick W. “Usability Evaluation in Industry,” CRC Press, 1st edition, July 22, 2014.

[28] Kanchana, B., and V. V. S. Sarma. “Software Quality Enhancement through Software Process Optimization using Taguchi Methods”, Proceedings ECBS’99. IEEE Conference and Workshop on Engineering of Computer-Based Systems, 1999.

[29] David Kung, “Object-Oriented Software Engineering: An Agile Unified Methodology,” McGraw-Hill Higher Education, 2013. (24 chapters, 720 pages.)

[30] D. Kung, “The object-oriented paradigm,” Encyclopedia of Microcomputers, Vol. 12, pp. 287 - 305, Marcel Dekker Publishing Inc., 1993.

[31] D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima, and C. Chen, "Change impact identification in object oriented software maintenance," Proc. of IEEE International Conference on Software Maintenance, pp. 202 - 211, 1994.

[32] D. Kung, J. Gao, P. Hsia, Y. Toyoshima, C. Chen, Y.S. Kim, and Y. Song, "Developing an object-oriented software testing and maintenance environment", Communications of the ACM, Vol. 38, No. 10, pp. 75-87, October 1995.

[33] Y. Labiche, B. Kolbah and H. Mehrfard, "Combining Static and Dynamic Analyses to Reverse-Engineer Scenario Diagrams," 29th IEEE International Conference on Software Maintenance (ICSM), 2013., pp. 130-139.

[34] Craig Larman, "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development" (3rd Edition), Prentice Hall, 2004.

[35] E. Lee, B. Lee, W. Shin and C. Wu, "A reengineering process for migrating from an object-oriented legacy system to a component-based system," The 27th Annual International Conference on Computer Software and Applications(COMPSAC), 2003. pp. 336-341.

[36] M. Maier and E. Rechtin, "The Art of Systems Architecting (Systems Engineering)," 2nd Ed., CRC Press, Boca Raton, Florida, 2000.

[37] Mathur, Bhawana, and Manju Kaushik." In Object-Oriented Software Framework Improving Maintenance Exercises through K-Means Clustering Approach," IEEE, 2018.

[38] L. Martinez, C. Pereira and L. Favre, "Recovering sequence diagrams from object-oriented code: An ADM approach," International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 2014., pp. 1-8.

[39] The CK Metrics tool, https://github.com/mauricioaniche/ck#ck.

[40] McCabe, T. J., "A software complexity measure," IEEE Trans. on Software Engineering, Vol. 2, No. 6, pp. 308 - 320, Dec. 1976.

[41] Object-Aid tool, "ObjectAid," Available: http://objectaid.com/.

[42] T. Parsons, A. Mos, M. Trofin, T. Gschwind and J. Murphy, "Extracting Interactions in Component-Based Systems," IEEE Transactions on Software Engineering, vol. 34, no. 6, 2008., pp. 783-799.

[43] I. Pashov and M. Riebisch, "Using feature modeling for program comprehension and software architecture recovery," The 11th IEEE International Conference and Workshop on Engineering of Computer-Based Systems, 2004. pp. 406-417.

[44] Pighin, M. "A New Methodology for Component Reuse and Maintenance", Proceedings Fifth European Conference on Software Maintenance and Reengineering, 2001.

[45] Polo, M., M. Piattini, F. Ruiz, et al. "MANTEMA: A Software Maintenance Methodology Based on the ISO/IEC 12207 Standard", Proceedings 4th IEEE International Software Engineering Standards Symposium and Forum (ISESS'99). 'Best Software Practices for the Internet Age', 1999.

[46] Roger S. Pressman, "Software Engineering: A Practitioner's Approach," 8th Ed., McGraw-Hill, 2014.

[47] D. M. Rickman, "A process for combining object oriented and structured analysis and design, " 20th DASC. 20th Digital Avionics Systems Conference 2001.

[48] Anam Sahoo, David Kung, and Sanika Gupta, "An agile methodology for reengineering object-oriented software," Proc. of 28th International Conference on Software Engineering and Knowledge Engineering, Redwood City, San Francisco Bay, California, USA, July 1 - July 3, 2016.

[49] A. Serebrenik, S. Roubtsov, E. Roubtsova and M. van den Brand, "Reverse Engineering Sequence Diagrams for Enterprise JavaBeans with Business Method Interceptors," The 16th Working Conference on Reverse Engineering, WCRE, 2009. pp. 269-273.

[50] Siddik, Saeed, Alim U. Gias, and Shah M. Khaled. "Optimizing Software Design Migration from Structured Programming to Object Oriented Paradigm, " IEEE, 2014.

[51] Ian Sommerville, “Software Engineering,” 10th Ed., Pearson, 2015.

[52] Suenobu, H., et al.“Stepwise Approach for Introducing Object-Oriented Technique at Software Maintenance Stages,” vol. 5, IEEE, 1999.

[53] Sparxsystems tool, “Enterprise Architect,” Available:http://www.sparxsystems.com/.

[54] Tahvildari, L., and K. Kontogiannis. “A Software Transformation Framework for Quality-Driven Object-Oriented Re-Engineering”, International Conference on Software Maintenance, 2002. Proceedings, 2002.

[55] Tigris tool, “AgroUML,” Available: http://argouml.tigris.org.

[56] Tan, H. B. K., Y. Yang, and L. Bian. “Systematic Transformation of Functional Analysis Model into OO Design and Implementation. “ IEEE Transactions on Software Engineering, vol. 32, no. 2, 2006, pp. 111-135.

[57] P. Tonella and A. Potrich, “Static and dynamic C++ code analysis for the recovery of the object diagram,” International Conference on Software Maintenance, 2002. pp. 54-63.

[58] Vora, Urjaswala, and N. L. Sarda. “Framework for evolving systems. ” In Proceedings of the 5th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems, Madrid, Spain. 2006.

[59] Vora, U. “Architectural Design Methodologies for Complex Evolving Systems”, 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007), 2007.

[60] AH Watson, DR Wallace, TJ McCabe, “books.google.com”- 1996.

[61] Wise, Richard, Lindsey Sheppard, John Huggins, et al. “A Methodology and Heuristics for Re-Architecting a Legacy System”, 2015 Annual IEEE Systems Conference (SysCon) Proceedings, 2015.

[62] Wong, W. E., and J. J. Li. “Redesigning Legacy Systems into the Object-Oriented Paradigm, ” IEEE, 2003.

[63] Xing, Zhenchang, and E. Stroulia. “Understanding Phases and Styles of Object-Oriented Systems’ Evolution”, 20th IEEE International Conference on Software Maintenance, 2004. Proceedings, 2004.

[64] Yongchang, Ren, Liu Zhongjing, Xing Tao, et al. “Software Maintenance Process Model and Contrastive Analysis”, 2011 International Conference on Information Management, Innovation Management and Industrial Engineering, vol. 3. 2011.

[65] E. Yourdon, “Modern Structured Analysis,” Prentice- Hall, Englewood Cliffs, N.J., 1989.

[66] Zaki, M. Z. M., and D. N. A. Jawawi. “Model-Based Methodology for Implementing MARTE in Embedded Real-Time Software, ” IEEE, 2011.

[67] Zhou, Jia, et al.“ A Software Enhancement System for Embedded Software Development,” IEEE, 2006.

[68] H. Zhou, H. Yang and A. Hugill, “An Ontology-Based Approach to Reengineering Enterprise Software for Cloud Computing,” IEEE 34th Annual Computer Software and Applications Conference (COMPSAC), 2010. pp. 383-388.

[69] Ziadi, T., et al. “A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams, ” IEEE, 2011.

[70] Zou, Ying, and K. Kontogiannis. “A Framework for Migrating Procedural Code to Object-Oriented Platforms”, Proceedings Eighth Asia-Pacific Software Engineering Conference, 2001.

[71] Zou, Ying.“Incremental Quality Driven Software Migration to Object Oriented Systems,“IEEE, 2004.