EXPERIMENTAL EVALUATION OF N-MODEL MOETHODOLOGY

by

MEHRAB IRANI

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

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

April 2019

ACKNOWLEDGEMENTS

I would like to thank my supervising professor Dr. David Kung for his continuous motivation and support. Dr. Kung’s guidance helped me to complete the thesis work on time, also his insight on various problems helped me to set my goals. Without his guidance and support, this thesis would not have been possible.

I like to thank my committee members Dr. Ramez Elmasri and David Levine for their support and for being a part of my thesis supervising committee.

I would like to thank Anam Sahoo for his help in collaborative study in various area of my thesis work and for his guidance and motivation throughout our research which we competed together.

Finally, I would like to thank my late Grandmother Roda Irani who inspired me to pursue my Masters.

May 10, 2019

ABSTRACT

EXPERIMENTAL EVALUATION OF N- MODEL METHODOLOGY

MEHRAB IRANI, M.S.

The University of Texas at Arlington, 2019

Supervising Professor: Dr. David Kung

Software maintenance is an essential part of the software development life cycle. Usually software engineers use ad hoc approaches to enhance legacy systems in the absence of a systematic methodology. However, there exists a methodology named "N- Model methodology" to enhance object-oriented legacy code. In this thesis, an experimental procedure is designed and applied to the N-Model methodology for enhancement of object-oriented software. A set of four categories of metrics; Process Metrics, Requirement Metrics, Design and Code Metrics and Test Metrics (total of 10 metrics) has been identified and applied. Additionally, a controlled experiment has been designed to compare the performance of the N-Model methodology with that of ad hoc approaches by using two separate legacy code bases. Although the experiment is limited in scope, using this experimental procedure and metrics, it has been validated that the N-model methodology significantly outperforms the ad hoc approaches.

Keywords: Software maintenance, object-oriented software, legacy system, software reengineering, reverse engineering, software process and methodology, agile method.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Software maintenance is an essential part of the software development life cycle. maintenance typically consumes an average of 60% of software life costs, of which more than 75% are spent on enhancements [5,6]. Usually software engineers use ad hoc approaches to enhance legacy systems in the absence of a systematic methodology. Software enhancement needs a process and a methodology. Process specifies *when* to do *what* but not how to do them. For example, the waterfall process states that first do 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 or how to carry out the activities of a process. There are several forward engineering process that exist to work on freshly initiated software projects, but hardly any literature is available on software enhancement methodology. There exists a process and methodology named "N- Model process and methodology[11]" to enhance object-oriented legacy code. The N-Process Model and Methodology is briefly described in the section 2. The methodology has three distinct phases: a release planning phase, iterative reengineering phase, and a system validation phase as shown in Fig. 1. The methodology has multiple releases and each release begins with a quick a agile planning phase, followed by an iterative reengineering phase consisting of a series of iterations and finally a system validation phase is performed to validate the release before its release[11].

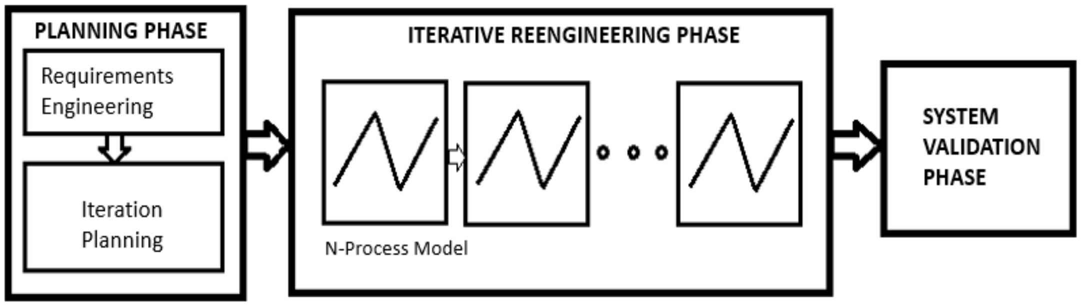

Fig 1: N- Process Model and Methodology Overview

In order to evaluate the efficacy of the N- Process model and methodology, an experiment was designed with a group of eight graduate students to enhance two individual projects. A set of three categories of metrics; Process Metrics, Design and Code Metrics and Test Metrics (total of 10 metrics) has been identified and applied to evaluate the results of the experiment. Although the experiment is limited in scope, using this experimental procedure and metrics, it has been validated that the N-model methodology significantly outperforms the ad hoc approaches.

The layout of the paper is as follows. Chapter 2 presents a brief description of N-Process Model and Methodology for enhancing a legacy OO software. Chapter 3 evaluates the effectiveness of the N-model methodology via an experiment. In Chapter 4, we present discussion and Conclusion.

CHAPTER 2

N- PROCESS MODEL AND METHODLOGY

This chapter explains the N- Process Model and Methodology with the steps to be followed for the said methodology briefly.

N- Process Model and Methodology has three distinct phases: a release planning phase, iterative reengineering phase, and a system validation phase as shown in Fig.1. For each release a quick agile planning phase is performed, followed by an iterative reengineering phase consisting of a series of iterations and finally a system validation phase is performed. The planning phase has two activities. First, new requirements are identified and prioritized by applying information collection techniques and are based on a statement of work (SOW) from the customer. 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 reengineering activities. The iterative reengineering phase consists of a series of iterations. Each iteration has three legs: the reverse engineering leg, the reincarnation leg, and the validation leg. This is referred to as the N-process model.

As described in “An agile methodology for reengineering object-oriented software[33]” the reverse engineering leg recovers design artifacts and helps to understand the existing system. It starts from a legacy code and has two major outputs: recovered Implemented Sequence Diagram(ISD) and recovered Implemented Class Diagram(ICD). There are several tools and techniques for recovering these design artifacts. The middle leg is the re-incarnation leg, which transforms the legacy system to a new working system. The third leg of the iterative reengineering phase focuses on validating the implementation against the intended design and requirements by preparing appropriate test cases. These include component level unit test cases, subsystem/system integration test cases, and system test cases. The system validation phase is meant to perform a formal release testing when a release candidate build is ready.

The N- Model Process and Methodology utilizes the above process and provides detailed steps on how to carry out the iterative enhancement activities. The Panning phase, Iterative Enhancement Phase and Validation phase is described as below:

a. Planning Phase:

As the first part of the planning phase, the enhancement requirements are identified prioritized and allocated for iterative development. Second, planning for release iterations is performed to assign the new, to-be-modified and to-be-deleted use cases to the release iterations. It is carried out by examining each of the enhancement requirements and existing use cases. The categories of use cases derived from enhanced requirements are classified into new use case, to-be-modified use and to-be-deleted use cases. These enhanced use cases are then assigned to iterative Enhancement Phase. How-ever, it is beyond the scope of this paper to detail this step.

b. 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 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. Finally, the validation leg generates and runs new test cases, and perform regression testing to ensure that the new, to-be-modified and to-be-deleted use cases are implemented correctly.

c. 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 3

EXPERIMENTAL EVALUATION

We have conducted an experimental evaluation of the N-model methodology using two real- world applications. Eight graduate students from an academic class participated in the experiment. This section presents the experiment design, evaluation metrics, data collection, data analysis, evaluation result as well as result validity and threats.

3.1 Design of Experiment

The experiment was aimed to evaluate the effect of the N-model methodology on student performance in enhancement projects. To accomplish this, we designed the experiment to evaluate the following hypotheses:

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

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

The subjects of the experiment were eight graduate students of Department of Computer Science and Engineering at The University of Texas at Arlington. They enrolled in an 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 back- ground relevant to the assignments. The purpose of the survey was to identify each student's proficiency of object-oriented programming, UML, Java, JSP, Tomcat and MySQL/relational DBMS. Based on the survey result, the students were divided into two teams with comparable academic and software development backgrounds. Each team had four students. All of the students were asked not to contact students of the other team during both enhancement projects.

Table 1: Information about the legacy systems used in the experiment

	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

The student teams were required to enhance two real-world, web-based legacy systems as two teamwork assignments. In particular, in Assignment 1, they were required to enhance MavAp- point, 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 our N-model methodology. They were then asked to apply the methodology to enhance a study abroad management system (SAMS), developed for the Office of International Education of our university. Information about these two legacy systems is shown in Table 1. Further detail of these two assignments is presented in the next two sections. The efficacy of the N-model methodology were evaluated by comparing the results of these two assignments using three categories of 10 metrics, to be described in Section 3.2. The evaluation results were then used to assess the hypotheses presented above.

3.1.1 Assignment 1 Description

In Assignment 1, the two teams were given the source code of MavAppoint, an online academic advising appointment management system. It allowed academic advisors specify their advising hours, and students make appointments with advisors as well as cancel appointments by students and advisors. 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 this assignment. The integer shown at the end of each enhancement requirement represents the task units. It was the average of the Poker Game estimates [2] produced by the two teams. We used this to measure the number of unit tasks needed to be completed per week to delivery 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 to be presented in Section 3.2.

1. MavAppoint shall prompt the user to change the system-generated temporary password when login is attempted 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 it. (1)

3.1.2 Assignment 2 Description

In Assignment 2, the two teams were required to modify a legacy system called Study Abroad Management System (SAMS). The system had been developed to support study abroad ex- change program of our university. It had 26 use cases. 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 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 relative 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 an urgent help. (1)

7. SAMS shall remind users of upcoming events they are registered for. (1)

8. SAMS shall notify users about any changes made to the study abroad program to which they are registered for. (1)

9. SAMS shall provide users the feature to opt for/opt out of email notification. (1)

10. SAMS shall provide the capability for students to access programs insurance requirements, facilitate enrollment to the travel insurance, vaccination information, along with do’s and don’ts to 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)

3.2 Evaluation Metrics

The efficacy of the N-model methodology was evaluated by using three categories of 10 metrics. That is, we compared the metrics in Assignment 1 and Assignment 2 to assess the two hypotheses stated in Section 3.1. This section presents the details of these metrics. Data collection and analysis to compute these metrics will be described in the following sections.

1.Process metrics. This includes: (1) time based efficiency, (2) overall relative efficiency, and (3) SUS scale[4]. Metrics one and two were taken from ISO/IEC 9126-4 [10]. They were calculated using the data accumulated during the entire enhancement process.

2. Design and Code Metrics. This includes: (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 [3], which presented six metrics. We had selected only these four because they were supported by an empirical study performed by Bassili, Briand and Melo [1]. The requirements coverage metric calculated the percentage of enhancement requirements that were fulfilled over the total number of enhancement requirements. The other four metrics measure 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. This includes: (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.

3.2.1 Process Metrics

The time based efficiency metric measures the number of tasks accomplished per unit time by the team of developers as a whole. It 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 nij is the completion status of task i by team member j. If the task is completed successfully by the team member, then nij is one, else it 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.

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

The overall relative efficiency computes the ratio of productive time over the total time spent by the developers to produce the software. The metrics 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 the formula, nij is zero if member j did not complete task i successfully, else it is 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 nij = 1 if a team member completes the task successfully. Using the above formula, the overall relative efficiency is

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

The *SUS scale* 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 are as follows:

1. I think that I would like to use this methodology frequently.

2. I found the methodology unnecessarily complex.

3. I thought the methodology was easy to use.

4. I think that I would need the support of a technical person to be able to use this methodology.

5. I found the various functions in this methodology were well integrated.

6. I thought there was too much inconsistency in this methodology.

7. I would imagine that most people would learn to use this methodology very quickly.

8. I found the methodology very cumbersome to use.

9. I felt very confident using the methodology.

10. I needed to learn a lot of things before I could get going with this methodology.

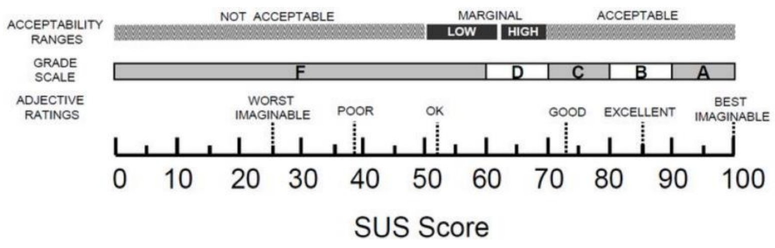

Fig. 2. Range of values - SUS scale

For each odd numbered question one is subtracted from the score, 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 2 gives the range of values and what they represent on the SUS scale.

3.2.2 Design and Code Metrics

The design and code metrics were used to assess the quality of the software produced. In particular, 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:

$$\mathit{Requirements\,Coverage} = \frac{\text{Number of requirements covered}}{\text{Total number of requirements}}$$

The requirements coverage metric was calculated using the code submitted by the teams as well as during the software demonstration to the TA. The other four design and code metrics are defined as follows [3] and calculated based on the source code submitted by the student teams for assignments 1 and 2.

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 Mc- Cabe [9]. 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 func- tion. For example, the cyclomatic complexity of a function with no conditional statement is 1; it 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.

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 probably inherit, which makes it harder to understand 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.

The *number of children (NOC)* metric is defined as the number of immediate subclasses sub- ordinated to a class in the class inheritance hierarchy. This 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 sub classing. 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.

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.

3.2.3 Test Metrics

The defect density metric measures the number of defects per thousand lines of code. A good methodology should have a low defect density. This metric is formulated as:

$$Defect\,Density = \frac{Total\,number\,of\,defects}{Number\,of\,Lines\,of\,Code}$$

The *ratio of success tests* is defined as the ratio of success tests over the total number of tests. In our experiment, the tests were prepared by the TA to ensure that the enhancement requirements were satisfied. The formula for this metric is:

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

3.3 Data Collection

This section presents the data collected for evaluating the 10 metrics. Data collection were done 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 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 estimated effort of one unit. These estimates were shown at the end of the enhancement requirements presented in Sections 3.1.1 and Section 3.1.2. The effort estimate for each of the enhancement requirements was collected from the teams as the Poker Game estimate [2] 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 answered the ten SUS-scale questions presented in Section 3.2. In particular, 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 estimated task units, which had been described previously. Moreover, the team must also submit the enhanced code, which was used to compute the four selected CK metrics. 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.

Data collection for test metrics. The teams were asked to submit the number of defects found, the number of lines of source code. The defect density metric was then computed as the ratio of the number of defects over the number of thousand lines of code. For the ratio of successful test cases, the teams were asked to submit test cases and their pass/fail/not-run results. [7] 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 CAPTHA 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 CAPTHA and enter it into the CAPTHA 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."

Table 2: 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%

3.4 Evaluation Results

The N-model methodology was evaluated using the data collected from the two assignments. The data were then used to compute the 10 metrics for each of the two assignments. Table 2 gives the evaluation results except the CK metrics, while Table 3 gives the evaluation results for the CK metrics. Table 2 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. Moreover, the SUS scores were also much higher, reflecting students' satisfaction in using the N-model methodology compared to their own approaches.

Table 3 shows the CK metrics for the original MavAppoint and SAMS legacy software as well as their enhanced versions. We obtained these using the CK Metrics tool [8] to obtain the metrics for all of the classes for each version and then computed the average of each of the metrics over all classes of each version. We see that compare to the original software the CK metrics do not change much. The weighted methods per class (WMC) metrics for the enhanced SAMS versions are noticeably higher than the original counterpart, compared to Assignment 1 results. In particular, for the enhanced versions, the WMC metrics had increased 17.63% and 28.59% for SAMS, compared to 0.64% and -2.19% for MavAppoint. We believe that these could be justified by the significant increases in requirements coverages and test case success ratios as well as significant reduction in defect density as shown in Table 2. For example, requirements coverages increased from 50% to 69% and 42% to 89%, while test success ratios increased from 50% to 66% and 42% to 80%, respectively. Similarly, defect densities reduced from 47 to 37 or 21.18%, and 44 to 39 or 11.36%, respectively. It is likely that the bigger increases in the

WMC metric are due to increased use of conditional statements to achieve better requirements coverage, defect density and test case success ratio.

Table 3: 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	7.5	7.76	7.7	11.42	16.86	17.93
Depth of Inheritance Tree (DIT)	1.49	1.6	1.5	1.85	1.78	1.86
StDev	0.5	0.49	0.67	1.07	1.05	1.1
Number of Children (NOC)	0.93	1.83	1.68	0.56	2.05	2.14
StDev	1.46	3.76	3.51	1.4	3.1	3.38
Response for Class (RFC)	10.44	8.39	8.3	10.8	11	11.59
StDev	9.62	10.06	9.75	14.97	15.75	16.68

3.5 Threats to Validity

Several threats to the external validity of our experiment may limit the generalization of the experiment results:

1. This experimental 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 as compared to large industry software systems. The evaluation results may not be valid for large industry systems.

2. MavAppoint and SAMS were rather limited in conceptual complexity and functionality as compared to large, complex industry software. Large, complex systems may not share the same experimental result.

3. The participants of this experimental evaluation were eight graduate students, divided into two teams of four students each. Their software development training and experiences were very limited as 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 10 hours per week working on the project. These were very different from the software development environments in industry. The computed metrics could be very different if the experiment were performed in an industry setting.

References

[1] 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.

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

[3] 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.

[4] Jordan, Patrick W. "Usability Evaluation in Industry," CRC Press, 1st edition, July 22, 2014.

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

[6] 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.

[7] 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.

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

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

[10] 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.

[11] 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,

BIOGRAPHICAL STATEMENT

Mehrab Irani received his bachelor’s degree in Information Technology from The University of Pune, India. He joined the University of Texas at Arlington in 2016 Fall to pursue his Master’s degree in Computer Science. He has worked in various roles in software industry for 3.5 years. His area of interest includes Software development and Project Management.