Investigating the Inspection Effectiveness of Software Requirements Specification with UML Diagrams: A Concept Paper

1Balsam A. Mustafa, 2Hamayoon Ghafory

1Senior Lecturer, Faculty of Computer System and Software Engineering, University Malaysia Pahang, Malaysia
2Master student, Faculty of Computer System and Software Engineering, University Malaysia Pahang, Malaysia

E-mail: 1balsam@ump.edu.my, 2hamayoon.ghafory@yahoo.com

ABSTRACT

Software inspections are a proven concept that is much used in industry. An inspection is a visual examination of a software product to detect and identify software defects, including errors and deviations from standards and specifications. Software requirements inspection is important as defects found late are expensive to correct and detecting defects in the requirements specification will considerably reduce the costs on rework in later stages. The unified modeling language (UML) aids developers to model different static and dynamic aspects of the software system and it is common to see UML diagrams included in software requirements specification (SRS) as a visual representation of the system. This research proposes the design of controlled experiment in an academic context in Malaysia to investigate the effect of including UML diagrams in (SRS) on the rate of defects detected in (SRS) and the effectiveness of the inspection process as well as the efficiency of inspection in terms of the time taken to complete the process.

Keywords- software requirements specification, requirements inspection, UML diagrams, controlled experiment

1. INTRODUCTION

Software systems requirements engineering (RE) is the process of discovering the purpose for which a software system is produced, as the measure of success of any system is the degree to which it meets that purpose [1]. Researchers have defined RE as a structured set of activities which are followed to derive, analyze, validate, maintain and check requirements usually stored in system requirements document. The product of the RE activities is the software requirement specification (SRS) document which is usually rich in knowledge and provides readers with a great amount of information in various sections which smooth the progress of reading [2].

Software inspection is a method initially designed by Fagan [3] for use in the coding stage to detect defects but it has been practiced in the industry in early stages of system development as researchers indicated that many defects are found in software requirements specification. Studies that researched the causes of software defects indicated that large percentage of defects is caused by incorrect specifications and translation of requirements, or incomplete ones. Moreover, it is found that half of the problems rooted in requirements are resulted from ambiguous, poorly written, unclear and incorrect requirements [4]. It is well known that some defects cannot be found by testing, and defects found late are expensive to correct. Several empirical studies have reported great savings or improved effectiveness when using inspections during the development process and inspections are found to be a very good mechanism for highlighting and prioritizing candidate areas for enhancement in the software development activities [5].

Typically, inspections require individuals to review a particular artifact, and then meet as teams to discuss and record defects, which are then, send to the document’s author to be corrected. The checklist-based reading technique (CBR) is the dominant inspection method used in industry and it will be used in this study. In this method each reviewer receives a checklist which poses questions that help reviewers discover faults, each questions implies a “yes” or “No” answer which suggests ways for reviewer to identify faults [6].

A typical practice that software requirements are specified in natural language [7]. However, the natural languages are intrinsically ambiguous and some researchers have studied the reasons of ambiguity in natural language and suggested various approaches to identify and measure it in natural language based software requirement specification [8, 9].

To the best knowledge of the authors, there is no empirical study has used the three diagrams (use case, sequence diagram, and activity diagram) in an experiment to investigate the effect of including these diagrams in the requirements specification on the inspection effectiveness of software requirements document. Also, it is the first experiment that aims to achieve this objective in an academic context in Malaysia.

The remainder of this paper is organized as follows. Section 2 presents an overview of UML diagrams. Section 3 summarizes related work. Section 4 sketches an experiment for evaluating the effectiveness and efficiency of inspecting software requirements specification that includes UML diagrams and Section 5 discusses the potential threats to the validity of the experiment results.
2. UML DIAGRAMS

The Unified Modeling Language (UML) has become a standard modeling language in OO software development. UML is not a methodology but a set of tools that provides visualization and enables developers to document and describe projects in a standardized way [10]. It offers two types of diagrams: structure diagrams and behavior diagrams. Use cases and use case diagrams are one of the key concepts in UML, where they are intended to help engineers in modeling user requirements as in Figure 1. In this research we are going to use the use case diagram and other UML diagrams that could usefully be compared against use cases, for example, the sequence diagram which helps to detect whether the functionality described by the use case was captured and expected behaviors regarding this functionality were represented as in Figure 2. Another UML diagram that will be used in this study is the Activity diagram which is basically a flow chart to show the flow from one activity to another activity. The activity can be described as an operation of the system. So, the control flow is drawn from one operation to another. In practice it is common to see requirements documents that include these UML diagrams.

![Figure1- Simple Use Case diagram for ATM system](image)

![Figure2 – Simple Sequence diagram showing a money withdrawal](image)

Very few researches investigated the effect of including UML diagrams in the SRS on the effectiveness of software requirements inspection. The main goal of this study is to examine whether including UML diagrams in the software requirements specification would help the SRS inspectors to detect more faults and increase the inspection effectiveness. We intend to undertake an experiment in an academic environment to investigate the effect of including UML diagrams in the software requirements specification on the rate of detected defects by individual reviewers.

3. RELATED WORK

In what follows, we discuss a number of empirical studies that have investigated different inspection techniques and the use of UML diagrams in software inspection. Sabaliauskaite et al [11] have reported an experiment to compare the effectiveness of two inspection techniques, checklist based reading (CBR) and perspective based reading (PBR) for object oriented (OO) design inspection. The results indicated that a) defect detection effectiveness using both inspection techniques is similar (PBR:69%, CBR:70%); b) reviewers who use PBR spend less time on inspection than reviewers who use CBR; c) cost per defect of reviewers who use CBR is smaller.

In the experimental study of Porter et al [12], a comparison and evaluation was made between Defect-based reading, Ad-hoc reading and Checklist based reading. They tried to evaluate and compare the effect of these techniques on fault detection effectiveness. They found out that the performance of defect-based readers was much better than ad-hoc and checklist readers. The defect based readers were not only able to focus on specific fault classes but also they were effective in detecting other faults, and finally the effectiveness of both checklist reading and ad hoc reading were almost the same.

Lange and Chaudron [13] performed an experiment with a group of (111) students to investigate to what extent implementers detect defects in UML diagrams (class and sequence diagrams) and to what extent defects cause different interpretations of the diagrams by different readers. The experiments’ results showed that defects often remain undetected and cause misinterpretation. In their attempt to assess the end product qualities of maintainability and correctness in the context of an inspection of the software design, an experiment is conducted by Ackerman et al [14] to investigate whether design inspection checklists can be tailored as to effectively target certain defect types without impairing the overall defect detection rate. The results showed that the design inspection approach used in the study uncovered useful design quality issues and that the checklists can be effectively tailored for some types of defects. Albayrak [15] investigated the effectiveness of inspection of SRSs that utilize UML diagrams and the number of defects reported. She conducted an experiment in academic environment and used the use case and the class diagram in the experiment. The results of analysis indicted that inspectors significantly reported more defects when they inspected documents with UML diagrams.

4. Experiment

We plan to carry out an experiment with the following goals:

- To examine the effect of inclusion of UML diagrams in the software requirements specification (SRS) on the rate of defect detection by inspectors and the effectiveness of inspection.
• To find out whether inspecting SRS documents with UML diagrams will take more time than documents without UML diagrams.

This section describes the first version of an experimental design. The design is inspired by the previous experiments referred to in section 3.

• Research questions

RQ1: What is the effect of including UML diagrams in the software requirements specification on the rate of detected defects by individual reviewers?

RQ2: What is the effect of including UML diagrams in the software requirements specification on the effectiveness of inspection?

RQ3: Does the inspection of SRS document with UML diagrams takes more time than that without UML diagrams?

Hypotheses

The Null hypotheses are assumed in this stage. The following hypotheses are constructed to test the effect of using UML diagrams in SRS on the defects detected, inspection effectiveness, and the time taken for inspection (Q1, Q2, and Q3).

H₀₁: No difference in the rate of detected defects between SRS documents with and without UML diagrams.

H₀₂: No difference in the effectiveness of inspection between SRS documents with and without UML diagrams.

H₀₃: No difference in time taken to inspect SRS documents with and without UML diagrams.

Hypothesis Variables:

Two types of variables are defined for the purpose of the experiment: independent variables and dependent variables.

The main dependent variables of the experiment are number of defects found in SRS, inspection effectiveness in terms of the number of correct defects found from a list of seeded defects, and the time spent by individual inspector on inspection. The independent variables include the SRS documents of software systems with and without UML diagrams.

Study Setup

The study will be conducted in class room settings at University Malaysia Pahang (UMP). For this study three types of UML diagrams will be used: use case diagram, sequence diagram, and activity diagram.

Subjects

There will be approximately 40-50 subjects who will participate in this experiment and do the inspector role. Participants will be senior students of software engineering in the faculty of computer systems and software engineering in (UMP). The subjects are expected to have varying experience and knowledge about the application domain, and varying experience and training in inspection techniques. It is assumed that they have had previous classroom experience with the programming languages, Object-Oriented development, UML, and software design activities. Students will be randomly assigned into two groups, one group will inspect the SRS document with UML diagrams and the other will inspect the same document without UML diagrams. Each group will focus on inspecting two software systems to increase the generalization of the experiment results.

Artifacts

The subjects will be presented with two software systems to increase the validity of the results. For each system, there will be two requirements specification documents: one with UML diagrams and another without UML diagrams. The artifacts with UML diagrams will have the same set of requirements written in natural language in the artifact without UML diagrams. The same requirements defects (inconsistency, omission, ambiguity, redundancy) will be seeded into the UML and non-UML versions of the requirements document. The subjects will be given a checklist to refer to while inspecting the artifacts.

5. Threats to Validity

1. The subjects in our experiment are students who may not represent software professionals which may limit the ability to generalize the results of our experiment.

2. The specifications documents may not represent a large real world applications, i.e. the inspection process may not represent a real software development practice.

3. In this experiment, the student’s subjects will study the artifacts individually. This situation may differ from how they usually read requirements documents. The limited number of subjects may also make it difficult to obtain statistical validity.

4. Each subject will be required to investigate several software requirements documents. There may therefore be learning during the experiment, which may affect the results.

References


AUTHORS PROFILES

1. Balsam A. Mustafa is a senior lecturer at the Software Engineering Department, Faculty of Computer Systems and Software Engineering, University Malaysia Pahang. She received her PhD in computer science in 2009. Her interests include software requirements engineering and empirical software engineering. She is a member of the IEEE and the IEEE Computer Society.

2. Hamayoon Ghafory received his degree in computer science from Kabul Education University, Kabul Afghanistan in 2007. He has worked as lecturer in computer Science department at Kabul Education University in Afghanistan since 2008. Currently he is a master student at Faculty of Computer Systems and Software Engineering, University Malaysia Pahang.