Early Warning Signs in Software Projects

Mihir Kothari

A thesis submitted to Auckland University of Technology in partial fulfilment of the requirements for the degree of Master of Computer and Information Sciences (MCIS)

2010

School of Computing and Mathematical Sciences

Auckland University of Technology

Primary Supervisor: Andy Connor

Secondary Supervisor: Stephen MacDonell

Table of Contents

Attestation of Authorship 4

Acknowledgements 5

List of Abbreviations 6

Abstract 7

Chapter I: Introduction 8

1.1 Introduction 8

1.2 Research Objectives and Methodology 10

1.3 Thesis Structure 14

Chapter II: Literature Review 15

2.1 Introduction 15

2.2 Software Project Failure Problem 16

2.3 Dealing with Uncertainty 20

2.4 Theory of Weak Signals 41

2.5 Early Warnings 48

2.6 Gaps in the literature 58

2.7 Summary 60

Chapter III: Research Objectives and Methodology 61

3.1 Introduction 61

3.2 Research Objectives 61

3.3 Research Methodology 62

3.4 Research Design 64

Chapter IV: Results and Analysis 69

4.1 Introduction 69

4.2 Early Warning Signs (EWS) 69

4.3 SimSE 72

4.4 Evaluation parameters 82
4.5 Experimental Design 83
4.6 Experimental Results 87
4.7 Summary 154

Chapter V: Discussion and Conclusion 155
5.1 Introduction 155
5.2 Answers to the Research Questions 155
5.3 Limitations and Recommendations 157
5.4 Conclusion 160

References 164

Appendices 174
Appendix A 175
Appendix B 177
Appendix C 180

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Yours sincerely,

Mihir Kothari

Acknowledgements

This research is one of the biggest academic achievements in my life. I am very grateful for the opportunity to conduct this research and finish it successfully.

First and foremost, I would like to thank my primary supervisor, Andy Connor whose contributions to the research effort are simply too many to mention here. I also wish to thank my secondary supervisor, Stephen MacDonell for his ideas, suggestions, and encouragements.

The participants in this research deserve special thanks. Without their time and efforts this research would not have been possible.

I would like to thank my parents and my younger brother, for their continuous support, love, care and countless prayers during my studies.

Last but not least, I would like to thank GOD for His love, His patience, His guidance, His providence, and ultimately who makes all of these things possible. Amen.

List of Abbreviations

CMM:	Capability Maturity Model
CMMi:	Capability Maturity Model Integration
EF:	Experience Factory
EWS:	Early Warning Signs
FDD:	Feature Driven Development
GQM:	Goal Question Metric
IDEAL:	Initiating, Diagnosing, Establishing, Acting and Learning
IPD-CMM:	Integrated Product Development Capability Maturity Model
ISO 9000:	International Organisation for Standardization 9000
ISO 9001:	International Organisation for Standardization 9001
P-CMM:	People Capability Maturity Model
PMBOK:	Project Management Body Of Knowledge
PMI:	Project Management Institute
PSP:	Personal Software Process
QIP:	Quality Improvement Paradigm
SA-CMM:	Software Acquisition Capability Maturity Model
SDLC:	Software Development Life Cycle
SDRM:	System Development Research Methodology
SE-CMM:	Systems Engineering Capability Maturity Model
SIM:	Strategic Issue Management
SPI:	Software Process Improvement
SPICE:	Software Process Improvement and Capability dEtermination
SPIQ:	Software Process Improvement for better Quality
SW-CMM:	Software Capability Maturity Model
TQM:	Total Quality Management
TSP:	Team Software Process
WFMS:	Workflow Management Systems
WSSIM:	Weak Signals Strategic Issue Management
XP:	eXtreme Programming

Abstract

The software industry has been plagued by the staggering failure rate of projects, which have resulted in the loss of billions of dollars. The well known Chaos Report by the Standish Group declared that software projects are in chaos with only 16.2% of software projects actually being successful in the year 1994 and a more recent study by them suggest that 32% of the projects were successful in the year 2009 (Eveleens and Verhoef, 2010; Dominguez, 2009; Bishop, 2009).

The post-mortem examination of failed software development projects reveals that failures do not happen overnight and that long before the failure, the projects render significant symptoms or “early warning signs” of trouble (Kappelman, McKeeman and Zhang, 2006). A warning sign is an indication or an event that predicts or alerts impending problems. Early warning sign provide an indication of manifesting risks. This research mainly focuses on a new and innovative concept known as early warning signs which could be incorporated into ongoing project risk management to ameliorate the project success rates by addressing early warning signs encountered during the project. The project risk management theories are not closely integrated with the early warning phenomenon but this can apparently be utilised as a tool in project risk management (Nikander, 2002).

The study utilises the System Development Research Methodology. The models simulating a typical project environment were designed using a simulation tool known as SimSE. For the evaluation of the models two experimental techniques namely “Individual EWS Testing” and “Controlled Experimental Study” were used. Findings of the research suggest that the implementation of early warning phenomenon has positive effects on the project outcomes. Also, there is a positive impact on the project outcomes if the corrective actions are taken early. The concept of early warning signs looks promising and this study is just one step in this direction and has introduced this new concept to the research arena.

Chapter I
Introduction

1.1 Introduction

In the 21st century, software has become pervasive and is an instrumental factor not only in almost every business but also in day to day living. Software development as a discipline has been into existence for more than 5 decades and during this time there have been numerous changes and improvements in software development techniques (Linberg, 1999). The industry has witnessed at least four generations of programming languages and three major development paradigms (Reel, 1999). In spite of this tremendous improvement and maturity of software development techniques, software development projects still fail and at an astounding rate. The well known Chaos Report by the Standish Group declared that software projects are in chaos with only 16.2% of software projects actually being successful in the year 1994 (Eveleens and Verhoef, 2010; Dominguez, 2009; Bishop, 2009). The term successful means software projects were able to achieve important and basic software project parameters like quality, schedule, cost and requirements objectives of the project. However, since that time it appears that IT project's success rate is improving, albeit slowly. According to a more recent study by the Standish group, 32% of the projects were successful in the year 2009 (Eveleens and Verhoef, 2010). Whilst the Chaos report has faced many critiques by several authors regarding the credibility, validity of figures, relevance and integrity (Eveleens and Verhoef, 2010; Robin and Goldsmith, 2007; Molokken, 2006, Glass, 2006), it provides some degree of evidence that developing software is still challenging today as it was at the time of writing of the Mythical Man Month (Brooks, 1995), though the reasons and their relative impact may have changed in the intervening period.

Software development projects are difficult to manage and extremely challenging in nature. There are many reasons why software development projects are more challenging than projects in other domains, in part because software projects have inherent characteristics like human interaction, abstraction, complexity and uncertainty. Out of all these, uncertainty is considered as one of the most important reasons for causing software project failures and is possibly caused by the intangible nature of the end product.

This research is addressing this challenge and identifying approaches to deal with uncertainty so that the project success rates can be ameliorated. This research mainly focuses on a new and innovative concept known as early warning signs which could be incorporated into ongoing project risk management to ameliorate the project success rates by addressing early warning signs encountered during the project.

The post-mortem examination of failed software development projects reveals that failures do not happen overnight and that long before the failure, the projects render significant symptoms or “early warning signs” of trouble (Kappelman, McKeeman and Zhang, 2006). A warning sign is an indication or an event that predicts or alerts impending problem(s) (Kappelman et al., 2006). Early warning sign provide an indication of manifesting risks. According to Nikander (2002, p. 49), early warning is defined as:

> *An early warning is an observation, a signal, a message or some other item that is or can be seen as an expression, an indication, a proof, or a sign of the existence of some future or incipient positive or negative issue. It is a signal, omen, or indication of future developments.*

Although there is no silver bullet to ameliorate the software project success rates but if these early warning signs are caught and acted upon early then the projects could be saved. Early warnings concept is a proactive management style to deal with uncertainties. The project risk management theories are not closely integrated with the early warning phenomenon but this can apparently be utilised as a tool in project risk management (Nikander, 2002).

1.2 Research Objectives and Methodology

It is suggested that implementing early warnings phenomenon may have positive effects on the project outcomes. But as yet there is no empirical evidence that demonstrates this. The aim of this research work is to provide some empirical evidence that there is a value in following and implementing early warnings phenomenon as indicators used to control project activities. A further aim of this research work is to provide some empirical evidence that there is a value in terms of undertaking corrective actions early by demonstrating that if we delay corrective actions then it has a negative impact on the project outcomes. These research objectives lead to the following two research questions which will be answered within this research:

- Does the implementation of early warning phenomenon have positive effects on the project outcomes?
- Is there positive impact on the project outcomes if the corrective actions are taken early?

In light of the research questions, let's explain the early warnings concept and what the research questions mean by using simple state diagrams. Let's consider a 'project' as a series of tasks or activities that transform the project from one state to another and assume that the project goes perfectly to the plan. When each task is successfully completed it takes the project to the next idealised state and the project ultimately reaches the desired end state as shown in figure 1.

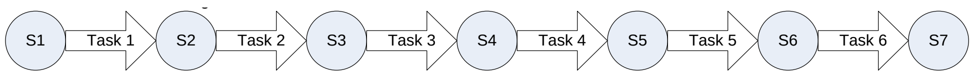


Figure 1: Ideal Project

This is fine, but as we know, projects rarely go exactly to the plan. So, let's assume that a task "fails" (i.e. it either takes longer time, or produces an unexpected result). In this case as shown in figure 2, we are assuming that task 3 fails and becomes task 3z.

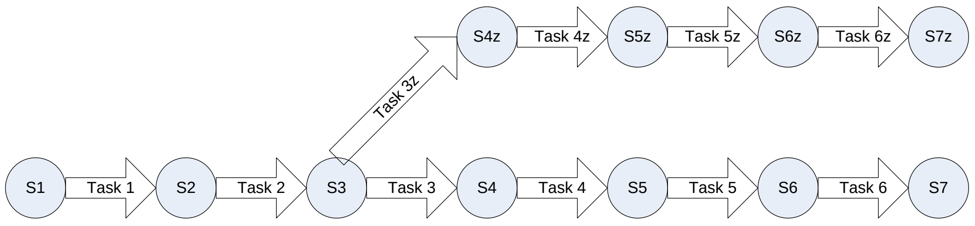


Figure 2: Task 3 fails

Depending on exactly how the task "fails", it is likely to have an impact on the downstream plan. So we might have thought that task 4 would take 3 days, but in fact if we carry on following the plan it may turn out that task 4 (and 5, 6, etc) takes a different duration and hence becomes task 4z.

The end state S7z, is not same as S7 and might represent an undesirable state i.e. a failed project.

The concept of early warning signs (EWS) suggests that before the task 3 fails (in this case) it shows some early warning signs - let's say in state 2. If we listen to the early warning sign and are pro-active and act (take corrective actions) then in an ideal situation the project goes back to the normal state where task 3 won't fail and the project still follows the ideal path.

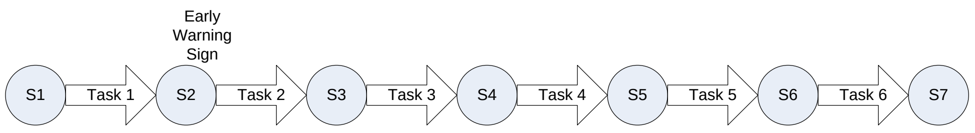


Figure 3: Ideal EWS situation

But no impact on timeline is very naïve to say and perhaps in non-ideal situation, the states may look like as shown in figure 4:

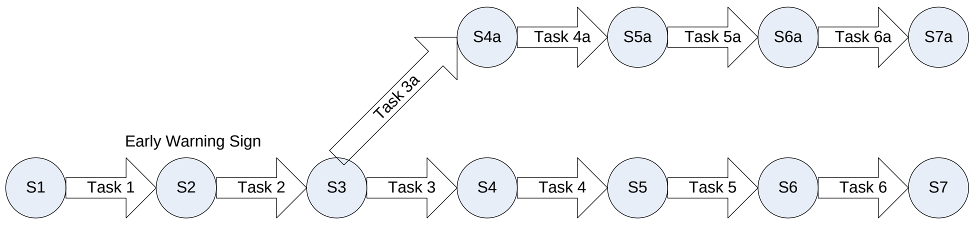


Figure 4: Non-ideal EWS situation

Through research question one this study is attempting to investigate whether the end state S7a is much closer to ideal state S7 if compared to failed state S7z. Or in other words is state S7z worse than state S7a if compared to ideal state S7? (i.e. does the implementation of early warning phenomenon have positive effects on the project outcomes?)

Further as shown in figure 5, depending on the time taken to apply corrective actions once we have found an early warning sign, project may take different branches and hence have different end states.

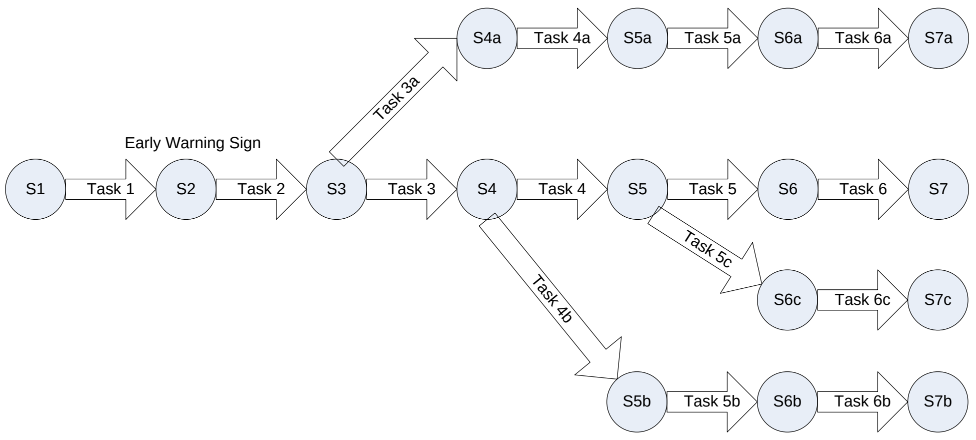


Figure 5: Different end states depending on the timing of the application of corrective actions

For example, branch 1 is where corrective action is taken after state 3 and hence we reach to end state S7a. Branch 2 is where the corrective action is slightly delayed (compared to branch 1) and it is applied after state 4 and thus we reach to end state S7b. Branch 3 further delays the application of corrective action (compared to branch 2) and hence reaches to state S7c. If we don't apply any corrective actions then we might reach to state 7z (which is an undesirable state). Through research question two this study is attempting to investigate whether state 7a is better than state 7b; state 7b is better than state 7c; etc, i.e. is there a positive impact on the project outcomes if the corrective actions are applied early?

This research is both exploratory and constructivist in nature. Based on the research objectives and the identified research questions, the System Development Research Methodology (SDRM) as suggested by Nunamaker and Chen (1990) has been utilised by the research. SDRM methodology has been used extensively in software engineering and information systems development research domain and it can accommodate dynamic evolution of the research in order to create innovations, define new ideas, and develop new technical capabilities (Limbu, 2008). SDRM has five stages which have been explicated in chapter 3.

1.3 Thesis Structure

This thesis has been divided into five chapters followed by references and appendices.

Chapter 1 - Introduction provides the overview of the software failure problem and lays the foundation for this thesis. Software development projects are difficult to manage and extremely challenging in nature. Human interaction, abstraction, complexity and uncertainty are four main perceived reasons for software projects failures. Furthermore, a new concept of early warning signs was briefly introduced which can be embedded into project risk management to improve project success rates.

Chapter 2 - Literature Review illustrates the essential background and the context central to this thesis. It discusses different focal areas to deal with uncertainties. Further, it also elaborates the theory of weak signals and early warning phenomenon.

Chapter 3 - Research Objectives and Methodology outlines the research objectives and the research questions addressed by this thesis. It also describes the research methodology and the research design employed by the research.

Chapter 4 - Results and Analysis presents the experiment design and the experiment results, driven by the research objectives and the research methodology. It describes the early warning signs used by the research. Also, the simulation tool SimSE which has been used by the study has been explicated.

Chapter 5 - Discussion and Conclusion presents the outcome of the research and answers the research questions of the study. It also forwards the limitations of the research. Further, it discusses several different directions in which future work can be undertaken. Finally, it concludes the thesis and highlights the contribution of this study to the research field.

Chapter II
Literature Review

2.1 Introduction

This chapter summarises the literature review of the domain under consideration (software project failure) and then discusses existing research in different areas that all have the goal of improving the success of software projects. It provides a broader context and the motivation of the study. Section 2.2 provides the facts and figures related to software project failure along with a review of the main perceived reasons for software project failure. In section 2.3, different potential areas of focus are discussed which have the potential to deal with uncertainties encountered by the project environment and thereby helping to ameliorate the project success rate. These areas of focus have the potential to inform an organisations ability to thrive in a dynamic work environment that is fraught with uncertainty. Section 2.4 describes the concept of “Theory of Weak Signals”. It was invented by researcher Dr. Ansoff, in an endeavour to improve strategic planning methods. This concept if applied in to project environment is known as “early warnings” phenomenon, which is explained in detail in section 2.5. Further, the interrelation of early warnings phenomenon and project risk management is also explicated. This section also provides some insights of how to integrate and apply early warnings in a typical problem environment, though little research has been done in this area. Section 2.6 provides information about the gaps in the literature and the motivation of the study. Section 2.7 provides the summary of the chapter in the form of the synthesis of the entire literature review.

2.2 Software Project Failure Problem

The nature of the software development discipline has already been discussed in Chapter 1. From a business perspective, one key to success lies on the use of contemporary software for running the businesses and thus providing a competitive edge. The software development discipline has witnessed at least four generations of programming languages and three major development paradigms (Reel, 1999). The first software development paradigm was invented in 1970s, after the article “Managing the Development of Large Software Systems” by Winston Royce was published (Larman and Basili, 2003). In that article, Winston Royce shared his ideas and his model which is widely known as the “waterfall model”. The waterfall model was the first attempt at formalising software development paradigms and lifecycle. Following on from the success of the waterfall model were the Incremental and Spiral models in the 1980s; Rapid application development (RAD) in 1990s; and Agile software development methodology in 2001 (Larman and Basili, 2003). In spite of this tremendous improvement and maturity of software development techniques, software development projects still fail. In fact, the failure rate of projects has been astounding. The term failure means software projects fail to fulfil all the important and basic software project parameters like quality, schedule, cost and requirements objectives. The well known Chaos Report by the Standish Group declared that software projects are in chaos with only 16.2% of software projects actually being successful in the year 1994 (Eveleens and Verhoef, 2010; Dominguez, 2009; Bishop, 2009; Standish Group International, 1995). Standish Group classified projects in to following three resolution types:

Successful:

The project is completed on time and on budget, with all features and functions originally specified.

Challenged:

The project is completed and operational, but over-budget, over the time estimate, and with fewer features and functions than initially specified.

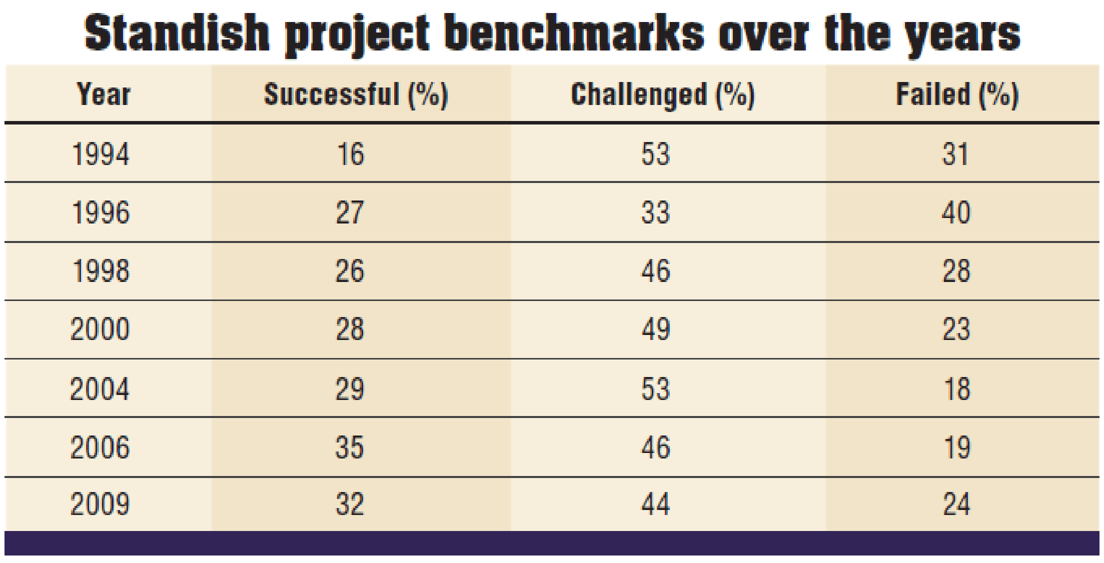
Failed:

The project is cancelled before completion, or never implemented.

In the 1995 Chaos report, the Standish Group reported that United States spent more than $250 billion each year for development of IT applications comprising of approximately 175,000 projects. Their research also suggested that in the year 1994, 31.1% of projects were cancelled before they ever got completed; 52.7% of projects had cost 189% of their initial estimates; and 16.2% projects were successful. The cost of these project overruns, lost opportunities and failures are really not measurable as it is very complicated task and have so many different dimensions to consider. According to Tiwana and Keil (2004), there were around $2.5 trillion spent on IT projects during 1997-2001 with nearly $1 trillion invested on underperforming projects. Most of the underperforming projects eventually failed costing companies more than $75 billion each year. According to research findings of Taylor (2000) which addressed 1027 projects, only 130 projects were successful. The success rate is just 12.7% which is not much different from the findings of Chaos Report. Heeks (2003) also reports similar project success rates (15%) as does Shein (1996), who stated that only 16% of software projects were successful (as cited in Martin and Raffo, 2000). Efforts by Jones (2004) have similar results. His research included 250 projects out of which only 25 projects were deemed as successful which is merely 10% success rate. Authors like Weill and Broadbent (1998); Barki, Rivard and Talbot (1993); Nidumolu (1995); Nidumolu (1996); and Lyytinen (1998) have also suggested that software development projects have been characterised by issues like over time, over budget and not fulfilling all user requirements (as cited in Jiang and Klein, 2000). Gorden (1999) and Johnson (1999) also have the same story and demonstrated that most software projects fail to function as intended and many projects never even got delivered (as cited in Wallace, Keil and Rai, 2004). Studies by other institutions like Gartner, GAO report, Carnegie Mellon University, Project Management Institute

(PMI) all point out the same reality that "projects very often fail". Such gloom and doom is not at all uncommon in the literature. Because software projects often spiral out of control, Glass (1998) describes these projects as "runaway projects" whereas Yourdon (1997) describes them as "death-march projects" (as cited in Linberg, 1999).

However, since Chaos report in 1994 it appears that IT project's success rate is improving, albeit slowly. According to a more recent study by the Standish group, 32% of the projects were successful in the year 2009 (Eveleens and Verhoef, 2010). The following figure 6 compares the results of the studies done by Standish Group in the years 1994, 1996, 1998, 2000, 2004, 2006 and 2009. It clearly outlines the steady progress and improvements achieved by the industry in terms of improving project success rates. However, it is important to understand that the number of projects in to the consideration by the survey has been different every year. Overall, it looks good with project success rate up, failure rate down and over runs down as well.


Standish project benchmarks over the years

Year	Successful (%)	Challenged (%)	Failed (%)
1994	16	53	31
1996	27	33	40
1998	26	46	28
2000	28	49	23
2004	29	53	18
2006	35	46	19
2009	32	44	24

Figure 6: Project success rates as per Standish Group (Eveleens and Verhoef, 2010)

However, the Chaos report has faced many critiques by several authors regarding the credibility, validity of figures, relevance and integrity. Jorgensen and Molokken (2006) have questioned the research findings of Standish Group's Chaos report. Glass (2006) has also doubted the relevance and integrity of Chaos report. Robin and Goldsmith (2007) also showed their concern on Standish report. In fact, Eveleens and Verhoef (2010) have stated that Standish's report results are meaningless for benchmarking.

Considering the use of software in the 21st century and the wealth of software development insights, 32% success rate of software projects in the year 2009 is considered low and a probable issue to address. And hence it provides some degree of evidence that developing software is still challenging today as it was at the time of writing of the *Mythical Man Month* (Brooks, 1995), though the reasons and their relative impact may have changed in the intervening period. This is supported not just by the Chaos report, but a much wider range of empirical studies.

2.2.1 Why software development projects fail?

Software development projects are difficult to manage and extremely challenging in nature. There are many reasons why software development projects are inherently more challenging than projects in other domains, in part because software projects have inherently complex characteristics like:

1. *Human Interaction:* Software development process is basically a human endeavour. This human element causes issues that relate to the qualities and nature of human beings especially the way they interact. Baines (1998) suggested that in a typical software development project environment 90 to 100% of the resources are people instead of material resources and therefore they carry higher risks (as cited in Crosby, 2007).

2. *Abstraction:* Unlike hardware, software is intangible and an abstract entity. Incomplete software is described in an abstract way and hence it becomes difficult to manage as it contains no clear visible milestone and physical manifestation to measure progress and quality (Jurison, 1999).

3. *Complexity:* The inevitable complex nature of contemporary software makes it challenging for the people to comprehend it effectively. Complexity not only causes technical problems but also causes management problems (Garvey, 1997; Jurison, 1999).

4. *Uncertainty:* Software is volatile and can be easily changed. Change in user requirements is not uncommon in a typical software development environment. Software projects face a lot of uncertainties throughout its life cycle causing an intense pressure to reach the project success criteria. Some of the uncertainties are due to external risks and beyond the control of project manager sometimes also perceived as “acts of God” like earthquakes and hurricanes (Schmidt, Lyytinen, Keil and Cule, 2001).

All the above characteristics make the software development process extremely challenging. However, uncertainty is considered as one of the most important reasons for causing software project failures. Project uncertainties are varied in their nature and can impact a project along all quality dimensions. The agile software movement has suggested that one solution is to embrace uncertainty (Anderson, 2004), though that concept is not restricted to agile projects alone. The idea of embracing uncertainty can be adopted by fostering a more dynamic management approach.

2.3 Dealing with Uncertainty

Project management has to plan and control the flow of project activities (Mauerkirchner, 2000). Typically, when management plan projects they create dependencies between tasks that define an order or sequence. Besides logical dependencies between activities, they also have to take into consideration aspects such as time and availability of specialised resources (Mauerkirchner, 2000). Further, each estimated plan depends upon several limiting constraints, which are influenced by real time decisions based on the uncertainties or disturbances faced by the project environment. Some of the examples of disturbances faced by a project are:

1) time deviations of project activities
2) resource changes
3) quality objectives not met

Therefore project management strategies should be dynamic and non-deterministic in nature, and needs the ability of adapting itself due to external decisions or factors (Mauerkirchner, 2000). Non-deterministic planning of projects is still not a common approach, though has some grounding in the academic community (Connor, 2007). The more traditional approach to dealing with uncertainty is to introduce fixed contingencies in terms of both budget and schedule to provide a buffer should unforeseen complexity arise (Connor, 2007). Often, these contingencies are proposed at the project outset and not revised throughout the project progression (Connor, 2007). Similarly, project plans are not often revised dynamically through the project.

There are a range of areas that can be considered in order to incorporate dynamic management and foster an attitude of continual refinement. If these areas are dealt appropriately by the project then there are chances to mitigate the risks caused due to uncertainty in software projects. These are dealt with in the next sections.

2.3.1 Requirements Engineering (RE)

Requirements engineering (RE) refers to the process of determining requirements by analysing customer needs and then representing it systematically in the form of specifications. A specification is a concise document consisting of the requirements which software must satisfy it. RE process is considered to be a bridge between stakeholders and developers.RE process consists of four distinct activities: elicitation, modelling, validation and verification (Hofmann and Lehner, 2001). Elicitation is basically understanding and perceiving the customer's requirements. There are different methods but often used are interviewing users or conducting group sessions or workshops or analyzing documents. Modelling includes converting expert’s requirements perceptions in to formal, semi-formal or informal notations (in application domain’s context). Different models (e.g. UML models like Use Case, State, Activity, Sequence, Component diagrams) are used to represent the information. Validation and verification stage allows receiving feedback from customers and developers and thereby helping to maintain internal and external consistency.

Requirements engineering is one of the initial and crucial phases of system development life cycle, yet recent research has shown that few organisations in New Zealand adopt any degree of formal requirements activities (Talbot, 2010). The management of requirements engineering process is inevitably an essential issue for successful project management (Nyugen and Swatman, 2003). Ferreira, Collofello, Shunk, Mackulak and Wolfe (2003) demonstrated the importance of requirements engineering process for a successful project outcome. They expressed that inability of RE process to deal with volatility directly effects project management key indicators like cost, schedule and quality. A very classic example of it is cost of correction of a defect as shown in figure 7. McConnell (1998) suggested that an error/defect created in early phases of a project, for example during RE phase costs 50 to 200 times more if corrected at a later stage compared to a point close to where it was originally created. The reason is one requirement can easily turn into many design diagrams which in turn gets converted into hundreds of lines of source code, heaps of test cases, user manuals, help screens and others. Hence poor RE can become the cause of project failure.

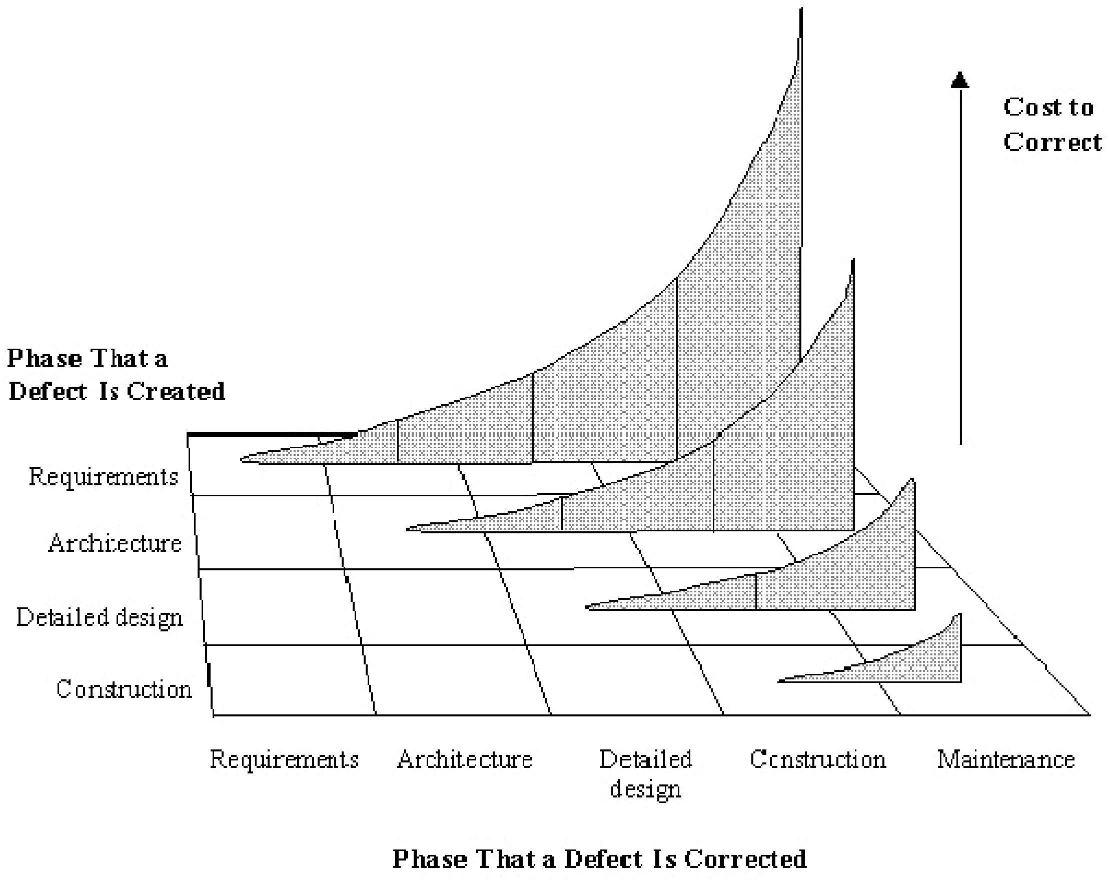


Figure 7: Correcting cost depending on the phase that a defect is corrected (McConnell, 1998)

In theory, the requirements engineering process is considered to be systematic, incremental and smooth in nature but according to Nyugen and Swatman (2003), the requirements engineering process should differ from the classical traditional model in order to incorporate the complexities of current project environment. They suggested that the RE process is not a smooth, systematic and incremental evolution of the requirements model but requires periodic restructuring and reorganising of it. By its very nature, adopting the dynamic nature of the project requirements should encourage project managers to revisit and revise their project estimates more frequently. Nyugen and Swatman (2003) also advised on adding creativity factor in RE process for adding more flexibility in the process. However they didn't provide much information about what creativity means in RE domain, how it should be implemented and how it helps to increase flexibility.

Hofmann and Lehner (2001) have similar ideas and suggested that RE process should be able to combat with communication breakdowns, fluctuating requirements and other uncertainties which emerge during a typical project life cycle. Getting requirements right is an important and difficult part of software project. They believed that RE process should be continuous and progressive till the end of the project to specify a project successfully. They motivated the idea of performing multiple RE cycles.

To summarise, refining, restructuring, simplifying and reorganising the requirements specification/model can be one of the effective management strategies to deal and incorporate dynamic elements in the software development life cycle.

2.3.2 Planning and Replanning

Strong upfront planning is the cornerstone of successful software development projects (Jurison, 1999). Project planning consists of identifying the activities, determining the resource needed and its management to ensure successful project completion. Planning includes project definition (defining objectives and requirements; choosing development methodology and defining the work - WBS), estimation (cost and scheduling) and risk assessment (Jurison, 1999). It involves selecting the appropriate processes and management related activities and making important decisions of parameters associated with those activities like tools, resources and others. Project planning culminates in a software project plan which is like a roadmap for guiding team members (Jurison, 1999). Planning is an important phase of software development life cycle and poor planning has been root cause for many project failures (Jurison, 1999). Because planning is one of the initial stages of software development life cycle, it’s not uncommon that during this phase the known information is incomplete. Also, in typical project environment the uncertainty factor makes planning process more vulnerable. Kirk and MacDonell (2009) proposed that as project progresses, (re)planning should occur in order to eliminate the risks introduced by uncertainty and inadequate information during planning phase. In their work they expressed that there are two key reasons for (re)planning effort:

1) As all process and management related activities aim to achieve project objectives/goals, replanning helps to keep a track of any change which increases the likelihood of the project goals not being met.
2) Replanning effort helps to keep a check on predictions of uncertainty. For instance, a project manager might predict that a particular activity will take a certain amount of time for a developer but this time is purely approximate and its accuracy depends on many factors like experience of the developer or project manager, domain expertise, tools available, and others. Such an uncertainty factor is also present if the prediction is based on company's historical data for similar projects.

Kirk and MacDonell (2009) also designed a framework which supports (re)planning for software projects. In addition, Rainer and Shepperd (1999) suggested the following reasons for replanning effort:

1) Events and issues which were unplanned and unexpected. For example, important resource leaving the organisation.
2) Events and issues which were expected but can't be planned precisely. For example, working overtime: managers anticipate that this will be essential but explicitly "when" is not sure.
3) Events and issues which were expected and planned but the original plans were imprecise. For example, plans prepared on the basis of incomplete information.

Emran, Pfahl and Ruhe (2008); Nyugen (2006); Srinivasan (2002) also motivated the idea of replanning and expressed that planning phase should be iterated in order to support uncertainties in the project. The contemporary and widely adopted agile software development methodologies employ this idea as well.

2.3.3 Decision Making System

Decision making is a significant aspect of software development life cycle but there are very few studies uncovering the decision making process in comparison to other activities like planning, estimating, designing, coding, testing and others. This is possibly because decision making is embedded in all other activities; it is not really an activity in its own. The two basic principles of effective decision making are; a clear apprehension of the decision itself and the accessibility of proper required information for supporting the decision (Ncube and Dean, 2002). Both these facets are dealt with the help of proper decision making system. The traditional static decision making systems were based on a manager's experience (Ruiz, Riquelme, Rodriguez and Ramos, 2002). But now, decision making rules are derived by using dynamic models which allows the identification of optimal solutions and helps in making good decisions even with incomplete and scarce data (Ruiz et al., 2002). Nyugen (2006) believed that "*unmatched structures between software development models and organisations driving them"* and "*little attention paid to the building of practical, automated, relatively simple and effective decision models"* are important reasons for delayed decision making process. He proposed a new decision making model which mainly included four concepts namely interoperability, mappability, controllability and accountability. Similarly Mauerkirchner (2000) also presented his model known as Decision Making System (DMS) which is non-deterministic and dynamic in nature.

A sophisticated project management system should contain a proper decision making model (Nyugen, 2006). Such systems should allow project managers to continuously monitor the status of the project with the objective of detecting deviation between the project plan and reality as soon as possible. In case of noted difference, a decision should be made for the correction of the project plan immediately. If the deviation is not addressed immediately and appropriately it may cause unnecessary delays and thereby leading to project failure. In situations where uncertainties arise, quick and effective decision making system could make a difference to the project's end result.

2.3.4 Software Process Management

A software process is a framework/model for the activities/tasks needed to achieve high standard software (Pressman, 2001). In order to develop a software product or a system, one approach is to follow a series of prescribed steps - a road map which aids to produce a timely and high standard end product. That road map is known as software process (Pressman, 2001). A software process determines the approach taken to engineer a given piece of software. It is a procedural structure that is imposed to manage software development activities. According to Lonchamp (1993) a software process is a coherent set of related products, ordered steps, policies, procedures, resources (human and technological), organisation structures, artefacts, constraints; all combined together to develop and maintain the requested software product. There are many similar terms associated to software process, like software development process and software life cycle.

Many advocates of software engineering suggest that there is an explicit causal relationship between the quality of the process and the quality of the product (Halvorsen and Conradi, 2001) .i.e.

$$\text{Quality } (\textbf{Process}) \Rightarrow \text{Quality } (\textbf{Product})$$

And therefore we should focus on software process in order to improve software’s (end product) quality. However, there are counter examples that show that quality in the process doesn’t lead to quality in the product but discussion of this side of the argument is out-of-scope of the thesis.

More and more people realise the importance of software process (Halvorsen and Conradi, 2001) in the era where software success rate is as low as 20%. In spite of wealth of research insights and domain knowledge the software process stream is lagging to cope up with rapidly changing business requirements. Anecdotal evidence and research manifest that there is high rate of discontentment with these software processes (MacCormack and Verganti, 2003). Software processes are surrounded by some serious inadequacies which cannot deal with uncertainties and dynamic environment in which a software project lives. Lubelczynk and Parra (2000) stressed on the need to focus on process and that these processes should be improved continuously in order to support uncertainties (as cited in Nguyen, 2006). Henderson, Howard and Walters (2001) voiced the idea that software process is a crucial element and that it should be improved and reviewed periodically to achieve better speed and quality of software development. A similar view was expressed by MacCormack and Verganti (2003), depicting that it is not that processes are poorly designed or implemented but the problem arises in the inflexibility of their specification which therefore does not allow organisations to improve the particulars of the process in order to reflect the unique context of a particular project.

Further, MacCormack and Verganti (2003) present an innovative way to deal this problem by recommending that we should adopt a “contingent” approach towards design of product development software process. Contingent view concept emerges from the fact that different kinds of project operated in different environments are potentially to involve different software development processes if they want to be successful (MacCormack and Verganti, 2003). Or in other words contingent views entails that the performance of different software development processes directly relate to the context in which those processes are operated. Such an idea is also supported by Rai and Al-Hindi (2000). MacCormack and Verganti (2003) believed that a superior performance is achieved if there is a certain match in the context between projects and software development process. In their work they have discussed about two important concepts, first the need to incorporate flexibility in the process and second about the sources/types of uncertainties. They believed that the traditional stage-gate sequential model no longer works effectively in a dynamic and uncertain environment. Rather they emphasised on more iterative process which were capable of learning and adapting. They called this process a flexible one, which refers to the ability to address new, unknown and uncertain information. In order to realise a flexible process higher investments in architectural/framework design is required as the design needs alterations i.e. facilitation of process flexibility. A flexible architectural design should be able to support early integration of the product design (helping to receive performance feedback at the system level) and also should have the ability to accept functional changes even at later phases of development in response to newly encountered information (MacCormack and Verganti, 2003). Getting early feedback on product performance at system level is an important activity in order to develop flexible process as it allows acquiring critical insights about the product’s design both from technical and market perspective (MacCormack and Verganti, 2003). Here it is possible to see the emergence of congruent ideas, as client involvement is promoted by the majority of formal requirements engineering approaches.

Normally loosely coupled or more modular architecture helps to achieve these objectives. But such techniques very often cause conflict with the design’s primary goal of optimising system level performance of the product. Selection of product architecture therefore becomes very complex and important task where projects face higher level of uncertainties compared to projects facing fewer uncertainties (MacCormack and Verganti, 2003). Their research focussed on projects which mainly faced two types of uncertainties namely:

1) Platform Uncertainties
2) Market Uncertainties

Platform uncertainties imply the uncertainties involved due to new work that must be carried out during the project (e.g. change in design). Whereas, market uncertainties imply the uncertainties involved in facing, understanding, analysing and identifying customer requirements for the product under development (e.g. confused customer or customer having less idea/impact of its product). They used the concept of uncertainties to define the context of the project.

Their findings suggest that for projects with higher uncertainties, early technical and early market feedback corresponds to higher performance. Early technical feedback helps to bridge platform uncertainties whereas early market feedback bridges market uncertainties. Managers should carefully select most appropriate process with some meaningful investments to optimise them such that they match the context leading to successful management strategy.

Coming back to software process management, there are two different dimensions through which software processes can be looked at. The first dimension is through software process modelling and second one is software process improvement (SPI).

The first dimension, software process model is defined as a description of a software process using appropriate modelling languages (Finkelstein, Kramer, Nuseibeh, 1994). Software process modelling is used to describe software process abstractly using techniques like state-charts, flowcharts, graphs, matrixes, Petri-nets, Unified Modelling Language (UML) diagrams (e.g. class, use case, activity, sequence and collaboration) and others. A study done by Rai and Al-Hindi (2000) demonstrates that process modelling of software development projects is positively related to process and product quality whereas task uncertainty is negatively related to them. They also suggested that projects facing high level of uncertainties should consider defining process models forming project management framework by defining tasks, their sequences, their relationships and logical dependencies.

Normally software process models are used for communication and execution purposes. Because process models are mainly represented in diagrammatic form they are perhaps inherently easy to understand and therefore facilitate effective communication between project stakeholders. Models like class diagrams are used for execution purposes as they can be easily converted in to an executable form. Different software process models help to define different perspectives or views of the software. For example one model may provide the sequence of the activities while other can provide the information about the actors involved in those activities. However, it is important to know that even though process models try to describe/represent the reality of the software it is very difficult to describe/represent all the aspects or views or perspectives of the software.

From the literature review it appears that there is some confusion over what is considered a software process model. In many cases, these process models are actually used to model a software product and not a software process i.e. process models are used to model software behaviour itself as opposed to defining the processes used to develop it. This leads to an interesting question of whether software projects don't concern themselves too much with the "how" of producing something, and focus on the "what" of the product itself? The “what” issue focuses on ‘product’ and “how” issue focuses on ‘process’. From research it appears that this has been a central point of discussion for a while now and there has been a pendulum effect to answer this question. In the early days of contemporary Software Engineering (SE) there was a major focus on the product. Then people realised that ‘the product is an outcome of the process’ and therefore there was a big push to turn the effort towards process which hence led to the dominance of frameworks like ISO9000, CMM, and others. But later people realised that it was possible to have a ‘good’ process (i.e. defined, documented, repeatable, managed, optimised, etc) in place and still produce bad software and hence the pendulum swung back to the product again. Attention to both, the 'product' and the 'process' is needed and is important for the project success. The dominance of one over the other depends on where the most important problems are. It is possible that the failure of many software projects could be attributed to getting the wrong balance between the “what” and the “how".

The second dimension is software process improvement (SPI). Soon after realising that software processes should be continuously monitored, evaluated, changed and improved to cope with the requirements of the market, researchers and practitioners came up with the idea of quality models which focussed on software process improvement (SPI). SPI focuses on systematic and structured ways for improving software processes of an organisation. SPI emerged as an effective solution for process related difficulties and is now gaining momentum in many organisations to achieve their IT goals

(Balandisand Laurinskaite, 2005). There are many SPI models like CMM, CMMi, SW-CMM, P-CMM, SA-CMM, SE-CMM, IPD-CMM, PSP, TSP, TQM, ISO 9000, ISO 9001, SPIQ, QIP/EF/GQM, SPICE (ISO/IEC 15504), Six Sigma, IDEAL and others. Henderson, Howard and Walters (2001) have proposed a new tool called RolEnactand Schackmann, Jansen, Lischkowitz and Lichter (2009) in their work have proposed a new tool called QMetric. These tools help software process analysis and their evaluation and thereby helping to understand any discrepancies in the process and leading to its improvement.

2.3.5 Workflow Management Systems

With an increasing demand to deal with dynamic work environment evolved the concept of workflow management systems (WFMS) which provides a new solution to deal with old problem of monitoring, controlling, optimising and supporting business processes (Aalst, 1998). According to Workflow Management Coalition (1996), workflow management system is defined as follows:

> *A system that completely defines, manages, and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic.*

Other terms used to refer workflow management systems are workflow manager, business operating system, logistic control system and case manager (Aalst, 1998). The workload management system concentrates on the logistics of business processes by assigning tasks to the users according to predefined workflow plan (Aalst, 1998). But there is a trade-off between the desire to control the processes and the desire to achieve flexible processes (Pesic, Schonenberg, Sidorova and Aalst, 2007). It appears that this paradox has limited the application of workflow management systems (Pesic et al., 2007). The traditional workflow management systems support the concept of having computer support for explicitly representing business process logic. Contemporary workflow management systems provide a wide range of functionalities. The tasks which the users are allowed to perform are offered to them in the form of work items through specific work lists by workflow management systems (Mans, Russell, Aalst, Moleman and Bakker, 2009). WFMS provide flexible environments and the infrastructure to manage business processes effectively. WFMS helps automating well defined repetitive business processes and thereby reducing the execution time significantly. It provides project managers better control over monitoring the process, allocating the resources and getting feedback. WFMS provide a means for allowing automated coordination of tasks that may be a part of many critical projects of an organisation (Sadiq and Orlowska, 2000). The order in which tasks are executed is very important in WFMS, as tasks are generally interrelated such that initiation of one set of tasks depends on the successful completion of the previous set (Sadiq and Orlowska, 2000).

Workflows are normally case based i.e. every set of activities is executed for a specific scenario or a case (Aalst, 1998). Typical examples of specific cases are a loan application, a tax declaration, an insurance claim, a request for information, an order. The main goal of WFMS is to handle these cases in the best possible manner by executing tasks in a specific order. The information about the task order is specified by workflow process definition. There are pre-conditions and post-conditions associated to each task (Aalst, 1998). Each task is executed by a resource which can either be a machine (e.g. fax, scanner, printer, etc) or a person (e.g. customer, employee, participant, etc).

Workflow management systems have the capability to work in dynamic environment. There are many different WFMS models namely DYNAMITE, DynaFlow, OpenPM, ADEPT, WIDE, WISE and others. Petri nets are widely used to realise these models. A dynamic workflow management system is capable of modifying its process model at run time in order to conform to its dynamic business conditions and unexpected situations (Meing, Su, Lam, Helal, Xian, Liu and Yang, 2006). One particularly interesting approach to dynamic workflow management is the Signposting (Clarkson & Hamilton, 2000) method that is in part based upon the Design Structure Matrix (Steward, 1981) approach, which in itself has been adapted to dynamic process management (Eppinger, Whitney, Smith, &Gebala, 1994). The basis of the Signposting approach is that any given process can be broken down into a series of tasks, and the relationship between tasks and the parameters related to the end product or process is defined. Dependencies between those tasks define an order or sequence. A dynamic process is created by recommending the next task on the basis of “what is known” about the project. Essentially, what happens is that as "what you know" changes, the approach recommends the best next task to do. This approach has successfully been applied to a number of engineering design processes (Clarkson & Hamilton, 2000; Clarkson, Melo, & Connor, 2000) as well as process visualisation (Clarkson, Melo, & Eckert, 2000).

2.3.6 Cost and Schedule Estimation

Cost and schedule estimation for software projects has been an active area for researchers over many years. An indication of the vibrancy of this topic can be gauged from the proliferation of review articles that exist (Boehm, Abts, & Chulani, 2000; Heemstra, 1990; Jørgensen, 2004; Jörgensen & Shepperd, 2007; MacDonell, 1994; Moløkken & Jörgensen, 2003; Niazi, Dai, Balabani, & Seneviratne, 2006). Estimation of cost and schedule is one of the most challenging tasks for a project manager. Generally, they are supposed to estimate at very initial stages (as early as biding phase) of SDLC where the information available is highly incomplete and inaccurate. A high bid could result in losing the bid whereas a low bid could result in a major loss (Connor and MacDonell, 2006). From this estimate, top level management decides whether to proceed further with the bid of the project. There is a need to have accurate estimates and a way to identify and compensate uncertainties. Cost and schedule estimates are in themselves dynamic, and indeed contain a great deal of uncertainty. It has been clearly identified that any cost and schedule estimates needs to be periodically revised to accommodate change as the degree of uncertainty is discovered or altered (Lederer & Prasad, 1993), however it is not uncommon for software development projects to simply estimate once at the beginning of a project and not make periodic revisions. Poor project management practices such as this have been identified as a failure factor in software development projects (Charette, 2005).

Some approaches have been developed that attempt to model and capture uncertainty in software project estimations to provide richer schedule information to project managers (Connor, 2007; Connor &MacDonell, 2006). In their work, Connor and MacDonell (2007) have proposed a novel way of tracking uncertainties present in cost and schedule estimates using Monte-Carlo simulation. Further, they have also linked the estimates with the historical database of organisation’s real project data which thereby gives realistic or practical touch to their approach. Long and Ohsato (2008) have done considerable work to develop a fuzzy critical chain method for dealing with project scheduling under uncertainties and resource constraints. Their idea is to develop appropriate deterministic schedule under resource constraints and then add a project buffer (PB) to the end of that deterministic schedule to address uncertainty factor. Computations are done using fuzzy numbers to determine the size of the project buffer. Once the project execution is started the penetration level (usage level) of the project buffer is determined and then dynamically schedule is updated to reflect more accurate schedule and actual status/progress of the project. Chang, Jiang, Di, Zhu and Ge (2008) used genetic algorithms to create more accurate schedules and task assignments.

However, still the challenge remains to merge together the rich information available from such approaches, with progress information available to project managers in a way that enables the project to be managed in a more dynamic manner to ensure the project success.

2.3.7 Change Management

Because of the nature of current business world, organisations often have to change their mode of operations in order to attain sustainable competitive advantage (Bloodgood and Salisbury, 2001). Change management helps to face these challenges and realise the ever changing needs of the customers (and business). Change management is a structured and a systematic approach of starting and managing the change process. During the change process there is a transition from current state (of project(s), system(s), process(es)) to a desired future state (defined by the required change)

(Creasey, 2007). It is crucial at both organisation and at strategic level. The ultimate goal of change management is to efficiently improve the organisation's practice of 'how work is done' to incorporate the required change (Creasey, 2007). A change normally impacts any of the following four parts of an organisation (Creasey, 2007):

1) Processes
2) Systems
3) Organisation structure
4) Job roles

Project management and change management share a close relationship and often go hand in hand. Project management emphasizes on applying techniques and skills to project activities to meet requirements whereas change management emphasizes on techniques to manage the 'resource (people) side' of the change to achieve business objectives (i.e. meet requirements in a typical project environment) (Creasey, 2007). Both have separate and independent approach but are integrated in practice. Resource and knowledge management are other two disciplines related to change management. Figure 8 below demonstrates how both project and change management collectively focuses on successful transition from current state to the future state as defined by the 'change'.

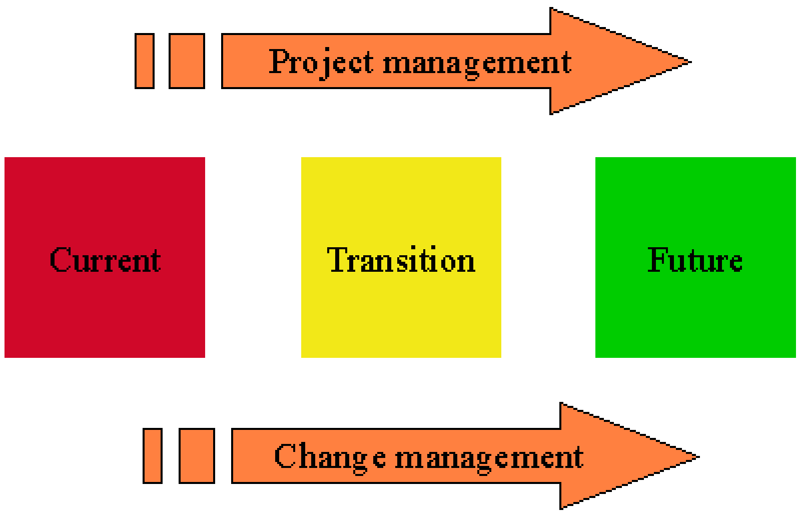


Figure 8: Project and Change Management Collaboration (Creasey, 2007)

Advocates of software engineering suggest that ‘change’ contain two sides one a ‘technical’ side and second a ‘people’ side. Project management focuses on the technical side whereas change management focuses on the people side. Change management helps personnel and individual transitions for the realization and adoption of change. Typically in a project management environment, change management acts as a project management process aiming to introduce changes to a project formally (Creasey, 2007). Sometimes project manager acts as a change manager.

Since the need for a change is often unknown and unpredictable, change management is often ad hoc, discontinuous and reactive (By, 2005). Change should be initiated and controlled using proper change management techniques and framework. Effective change management is considered as an essential element for the survival in today’s highly evolving and competitive environment (By, 2005). Lack of proper techniques and framework may result in unsuccessful management of change. There are some widely known change management frameworks like ADKAR model, Kubler Ross change model, Prosci’s methodology, Lewin Change model and others. However, selecting a framework and optimizing it to fit with the organisation’s practices, policies and structure may be challenging.

2.3.8 Agile Methodologies

In light of conventional software development life cycles and software project failures emerged agile methodologies. Agile methodologies are also known as lightweight or lean methodologies. Its built on four values i.e. adapting to changing requirements, higher customer satisfaction, iteratively delivering working software model, and close collaboration of customers (business people, clients and end users) and developers (Paetsch, Eberlein and Maurer, 2003). Agile methodologies like Extreme Programming (XP), Feature driven development (FDD), Scrum and others are being touted as the next generation software development methods (Paulk, 2002; Bose, 2008). Anecdotal evidence shows that agile methodologies are certainly effective for dealing with frequently changing requirements where as ineffective for stable requirements and mission or life critical projects, however there is an ongoing discussion regarding the suitability of agile methods for large scale projects.

Paulk (2002) believes that agile methodologies are the best choice for the current volatile and high speed world of software development. Highsmith (2000) also believes that agile methodologies provide iteration over software design and requirements by promoting more communication/interaction between the team members. Beznosov and Kruchten (2004) claim that agile methodologies help to reduce software project failures. This claim was also supported by Kuppuswami, Vivekanandan, Ramaswamy & Rodrigues (2003) and Abrahamsson, Warsta, Siponen & Ronkainen (2003) suggesting that by mitigating project risks agile methodologies helps in reducing software project failures. However the supporting researchers do not provide empirical evidence to back their claim. Agile methodologies also have been widely criticized. A very contrasting view was forwarded by Sharp, Robinson & Segal (2004) and Kirk & Tempero (2006). They argued that agile methodologies may introduce new complexities and thereby leading to the introduction of new risks.

Williams (2005) suggested that software projects characterised by complexities, time constraints, uncertainties should opt for agile methodologies. Whereas Boehm and Turner (2003) claimed that software projects characterised by highly predictable, requiring few changes, stable and mission or life critical projects should opt for conventional (plan driven) methodologies. The summary is that agile methodologies employ many good software engineering practices. However some of the practices are at an extreme end of the spectrum which causes controversial arguments (Paulk, 2002). Probably, a hybrid model combining both plan driven methods and agile methods could be more effective and might be an effective approach that bridges both extremes of viewpoint (Boehm, 2002).

2.3.9 Risk Management

Risk is defined as the likelihood or probability of failing to achieve objectives and the consequence of not achieving those goals (Conrow and Shishido, 1997). Risks are the factors which if present can adversely affect a project (Wallace and Keil, 2004). Anything which can endanger project's success can be referred as a risk. Project Management Body of Knowledge (PMBOK) defines risk as a condition or uncertain event if occurs, has positive or negative effects on project objectives (PMI, 2000). Project failures are the outcome of unattended or unaddressed or unmanaged risks (Alter & Ginzberg, 1978; McFarlan, 1981; Charette, 1989; Ginzberg, 1981; Boehm, 1991; Barki, Rivard, Talbot, 1993; as cited in Keil, Tiwana, Bush, 2002). Risks are present at every levels of project. Table 1 below shows the glimpse of some of the key risk issues at different levels.

Risk Grouping	Software Risks
Project level	• Excessive, immature, unrealistic, or unstable requirements • Lack of user involvement • Underestimation of project complexity or dynamic nature
Project attributes	• Performance shortfalls (includes errors and quality) • Unrealistic cost or schedule (estimates and/or allocated amounts)
Management	• Ineffective project management (multiple levels possible)
Engineering	• Ineffective integration, assembly and test, quality control, specialty engineering, or systems engineering (multiple levels possible) • Unanticipated difficulties associated with the user interface
Work environment	• Immature or untried design, process, or technologies selected • Inadequate work plans or configuration control • Inappropriate methods or tool selection or inaccurate metrics • Poor training
Other	• Inadequate or excessive documentation or review process • Legal or contractual issues (such as litigation, malpractice, ownership) • Obsolescence (includes excessive schedule length) • Unanticipated difficulties with subcontracted items • Unanticipated maintenance and/or support costs

Table 1: A Summary of Key Risk Issues (Conrow and Shishido, 1997)

Apparently, the importance of project risk management cannot be overemphasized in this unhealthy software project environment. Project risk management is an integral part of project management. PMBOK definition of project risk management is (PMI, 2000):

> *Risk management is the systematic process of identifying, analysing and responding to project risk. It includes maximising the probability and consequences of positive events and minimising the probability and consequences of adverse events to project objectives.*

It contains following four main processes (PMI, 2000):

1) **Risk Identification** - determining probable risks affecting the project and documenting it.
2) **Risk Quantification** - evaluating risks for assessing possible project outcomes.
3) **Risk Response Development** - techniques to address risks.
4) **Risk Response Control** - responding and controlling risk effects continuously throughout the project life cycle.

Advocates of software risk management claim that by identifying and treating risks, we can reduce the chances of project failure (Schmidt, Lyytinen, Keil and Cule, 2001). Implementing good risk management strategies on intensive software projects is an effective solution to keep a check on the risk levels of the project (Conrow and Shishido, 1997). However, the challenge still remains in identifying the risks which jeopardize project's success (Wallace and Keil, 2004).

2.3.10 Summary of different focal areas

In the above sections, we have identified and briefly explained different focal areas to deal with uncertainties. Dealing with all the areas would have been ideal but due to the time constraints it's not possible. But based on the level of impact and the interest of the author, the priority is Risk Management. Having said earlier that identifying the risks in a project is a challenging task, in the following sections we will learn about a new risk identification technique in project risk management known as "Early Warnings". Before getting into early warning signs let's explore the origin of its concept from "Theory of Weak Signals".

2.4 Theory of Weak Signals

The central aspect of this research is related to the concept of 'theory of weak signals'. The concept of 'theory of weak signals' was invented by Dr. Igor Ansoff, in an endeavour to improve strategic planning methods (as it didn't work satisfactorily when unforeseen or sudden changes occurred in business environment). He firmly believed that in this fast changing business environment strategic surprise(s) provide advance information of themselves (as cited in Nikander, 2002). However that information is initially ambiguous, inexact, fuzzy, unclear, vague and difficult to analyse (Nikander, 2002). The information then gradually becomes clearer, stronger, more evident and easier to interpret. Ansoff calls such information or signals as weak signals. He claimed that there are many weak signals stimulated due to change in environment. He defines weak signals in conjunction with strong signals. His definition for strong signals is as follows (Ansoff, 1984a):

> *Issues identified through environmental surveillance will differ in the amount of information they contain. Some issues will be sufficiently visible and concrete to permit the firm to compute their impact and to devise specific plans for Response. We shall call these strong signal issues.*

And weak signals as (Ansoff, 1984a):

> *"Other issues will contain weak signals, imprecise early indications about impending impactful events......all that is known (of them) is that some Threats and Opportunities will undoubtedly arise, but their shape and nature and source are not yet known."*

Weak signals mature over a certain period of time to become strong signals.

Many researchers have made an effort to define "weak signals" since then. Michelle Codet describes it as (as cited in Uskali, 2005):

"A weak signal is a factor for change hardly perceptible at present, but which will constitute a strong trend in the future."

Pierre Masse defines it as (as cited in Uskali, 2005):

"A sign which is slight in present dimensions but huge in terms of its virtual consequences"

Ansoff believed that if we were ready to deal with first issue/threat appearing on the horizon then the uncertainties in the environment wouldn't hurt us much.

However, the concept of 'theory of weak signals' by Ansoff has always been a central point of debate. Ansoff had provided no evidence to support and prove his theory. He had only provided information about his discussions with various people. Ansoff suggested that some actions should be taken when weak signals are identified (Ansoff, 1975). However Heiskanen (1988) admonishes this concept (as cited in Nikander, 2002) and Madridakis and Heau (1987) argued that 'theory of weak signals' has remained as just an academic idea. Webb (1987) in his work reported that it's not possible to get or obtain information or messages related to future events. Further, he stated that there is no grounding for Ansoff's research. Webb (1987) critically questioned the presence of weak signals and suggested that there needs more investigation to prove this concept. On the same note, Ashley (1989a) claims that such warning phenomenon doesn't exist at all. He believed that once the detection of an issue/event is made then there is hindsight bias which often causes interpretation of past events. Mintzberg (1994a) even questions the importance or necessity of weak signals in strategic planning and management. In his work, Betss (1982) acknowledges the bias of hindsight but also asserted the existence of warnings phenomenon. A similar view was shared by Morris (1997). Aberg (1993) suggested that weak signals are too vague and can be easily missed completely (as cited in Nikander, 2002).

As said earlier, Ansoff's theory is controversial. The main reason behind the argument is whether weak signals can be practically detected. There are studies which support Ansoff’s views. In his doctoral studies, Nikander (2002) demonstrated the presence of weak signals (he termed it as “early warnings” in typical project environment context). Similarly Weschke (1994) forwarded group of early warning signs associated to economically risky enterprise. After critically analysing Ansoff’s work, Webb in his doctoral studies investigated and identified the presence of weak signals (the lack of evidence of Ansoff’s theory was his prime motivation for his doctoral work) (Webb, 1987). The work of Leidecker and Brono (1987) manifested the importance of observing early warnings or weak signals when they identified the critical success factors in company’s activities. Similarly, project implementation profile by Pinto and Slevin (1992) can be considered as a research in support of the existence of weak signals. Ilmola and Kotsalo-Mustonen (2004) also believe in Ansoff’s theory (as their work of weak signal filters is based on Ansoff’s theory). Nikander and Eloranta (1997) based their work of preliminary signals and early warnings on Ansoff’s theory. Uskali (2005) has extended Ansoff’s theory to innovation journalism. Makela (1999) links weak signals to megatrends and implicit anticipatory knowledge whereas Mannermaa (1999a) in his book links it to wild cards (as cited in Nikander, 2002). In a quantitative approach Metsamuronen (1999) uses mathematical methods (Markov chain theory) for detecting weak signals (as cited in Nikander, 2002). Schoemaker and Day (2009) discussed different techniques to identify and interpret weak signals effectively.

“Weak signals” is becoming increasingly popular among researchers (Hiltunen, 2008). According to Miller and Ward (2003), Ansoff’s weak signal theory is witnessing a renaissance in context of strategic flexibility of an organisation. Mannermaa (1999a) in his book expresses that; weak signals would be one of the most fascinating research areas in future (as cited in Nikander, 2002). Not only strategic management but also other disciplines like communications research, journalism, international security, business economy (predicting bankruptcies), international politics and even military science are interested in weak signals (Nikander, 2002; Uskali, 2005).

2.4.1 Weak Signals Strategic Issue Management Systems

The real motivation behind inventing the theory of weak signals was to improve Strategic Issue Management (SIM). Management method which uses weak signals in SIM is called as Weak Signals Strategic Issue Management (WSSIM). Detection of weak signals is a challenging task and requires close attention to all the information. Advocates suggest that detection can be carried out at two levels namely, on an individual level as well as a more isolated environment scanning method (Nikander, 2002). Researchers like Ansoff (1975; 1984), Aberg (1993), Webb (1987), Juran (1995) and Taylor (1987) discuss at individual level detection whereas King (1987), David (1991) and Aberg (1993) suggest different models for environment scanning method (as cited in Nikander, 2002). Individual level detection requires sensitivity from everyone in the organisation and therefore shouldn't be left to a few key people of the organisation. Juran (1995) suggests that human beings are the best sensors to detect weak signals. These procedures are very similar to risk identification procedures of risk management (Nikander, 2002).

Ansoff (1984) suggests that a piece of information (or weak signal) has to go through different types of filters before it can be apprehended and influence the strategy process. These filters could either facilitate or confine the important information contained in those signals (Nikander, 2002). As shown in figure 9, Ansoff classifies these filters into three different types:

1) Surveillance filter
2) Mentality filter
3) Power/ Political filter

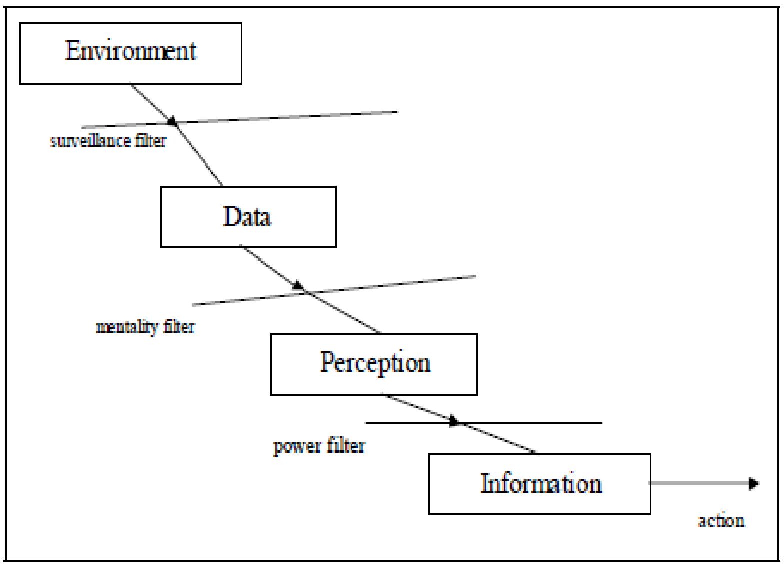


Figure 9: Ansoff's filters (Nikander, 2002)

The surveillance filter needs the organisation to identify the type of information required and the techniques which should be employed to do so. It defines the field or domain where observation needs to be done. It's the first of the obstructions the raw information encounters (Ilmola and Kotsalo-Mustonen, 2003). The information available after it passes through surveillance filter is usually inoperable and little of use. The information needs immediate reduction to make sense. This is done using mentality filter. The reduction criteria are based on the expertise knowledge of the domain. The final, power filter is used by the decision makers. The power filter determines the information which would influence the decision making of the organisation (Nikander, 2002). Figure 10 below depicts the decision making model of weak signals in SIM. If the impact (effect) of the concerned issue is minor then it can be dropped from the response analysis process. If the impact (effect) of the issue under consideration is major then the estimation of the signal strength needs to be done. If the signal is strong then three alternatives are available depending on the urgency levels of the response (Nikander, 2002):

(1) Delay-able response and needs continuous monitoring of the situation
(2) Postponable response and needs to be actioned in next cycle
(3) Urgent response and start taking immediate actions to address the response

If the signal is weak, then the options are (Nikander, 2002):

(1) Delay-able response and needs continuous monitoring of the situation

(2) Postponable response and needs to be actioned in next cycle

(3) Urgent response and start taking gradual actions to address the response

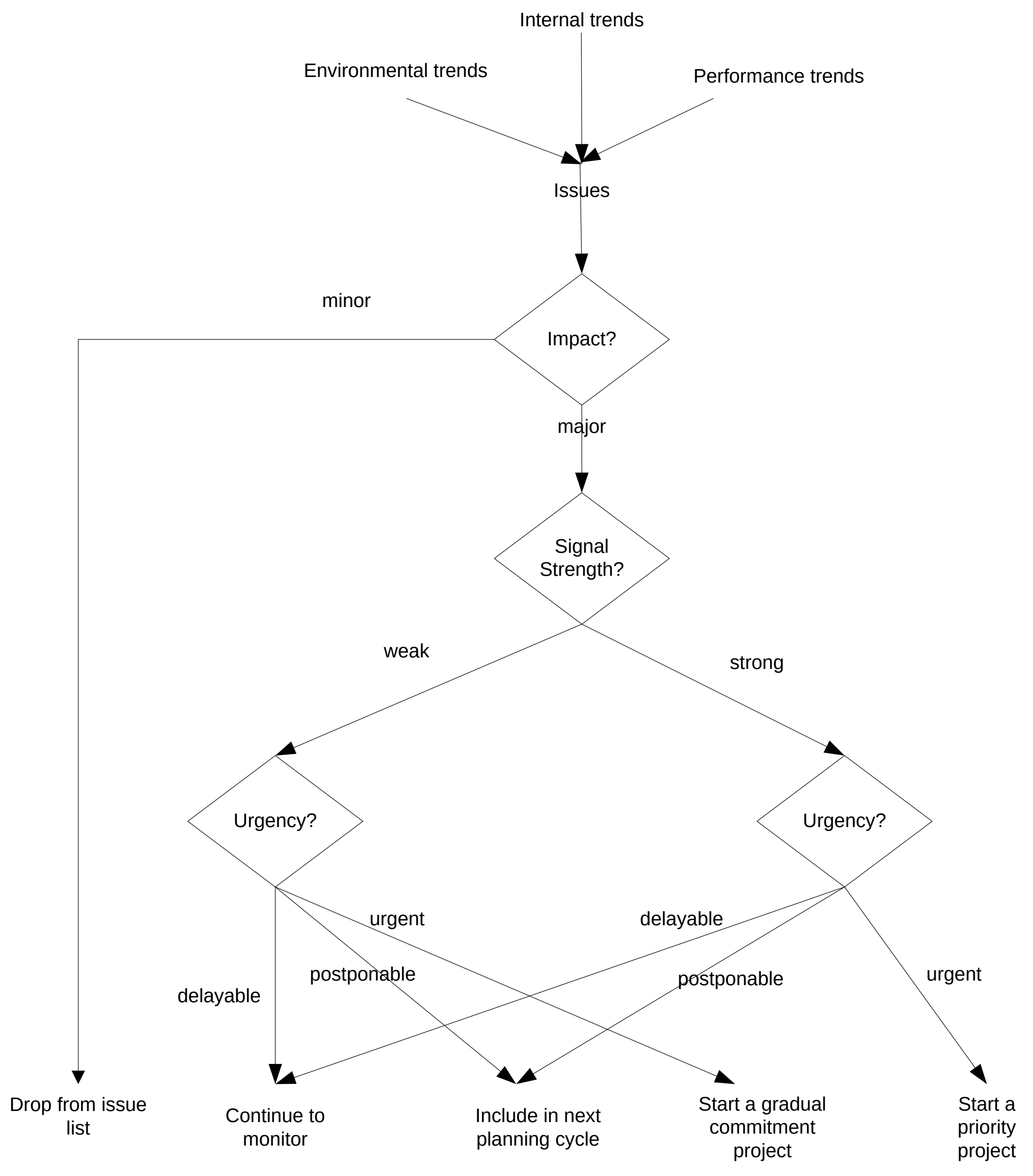


Figure 10: Decision making model of Weak Signal SIM (Ansoff, 1984)

Aberg (1989) also forwarded a model for incorporating weak signals (as shown in figure 11). He suggested that scanning for weak signals is a continuous process. And scanning should be continuously done internally as well as externally.

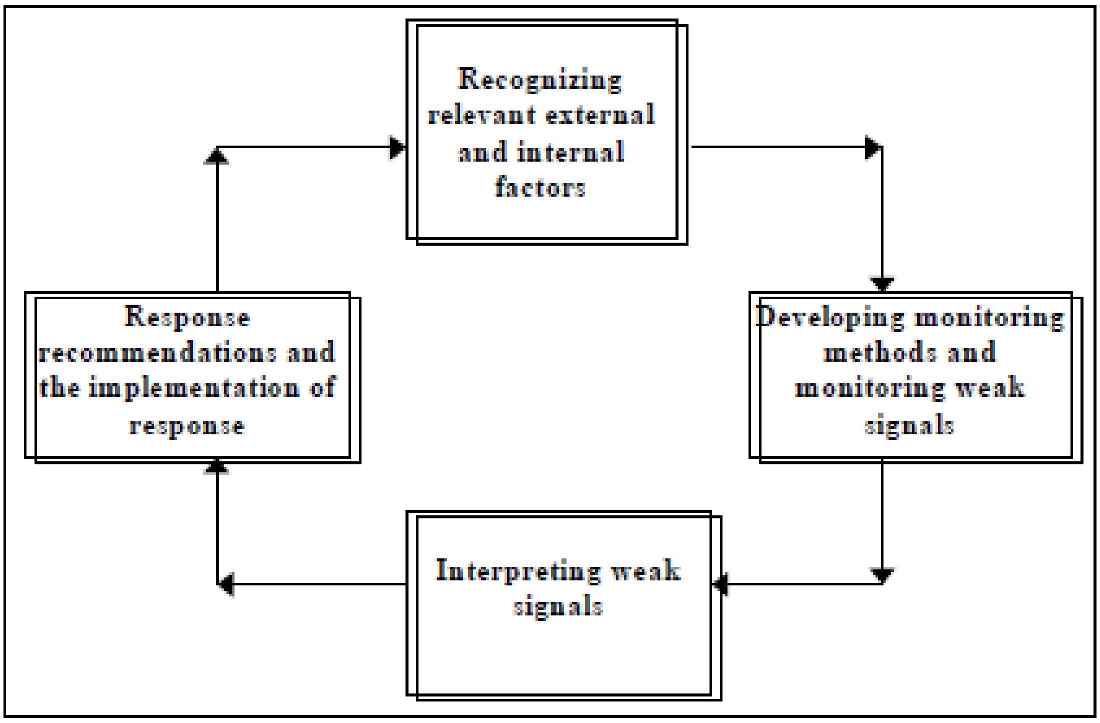
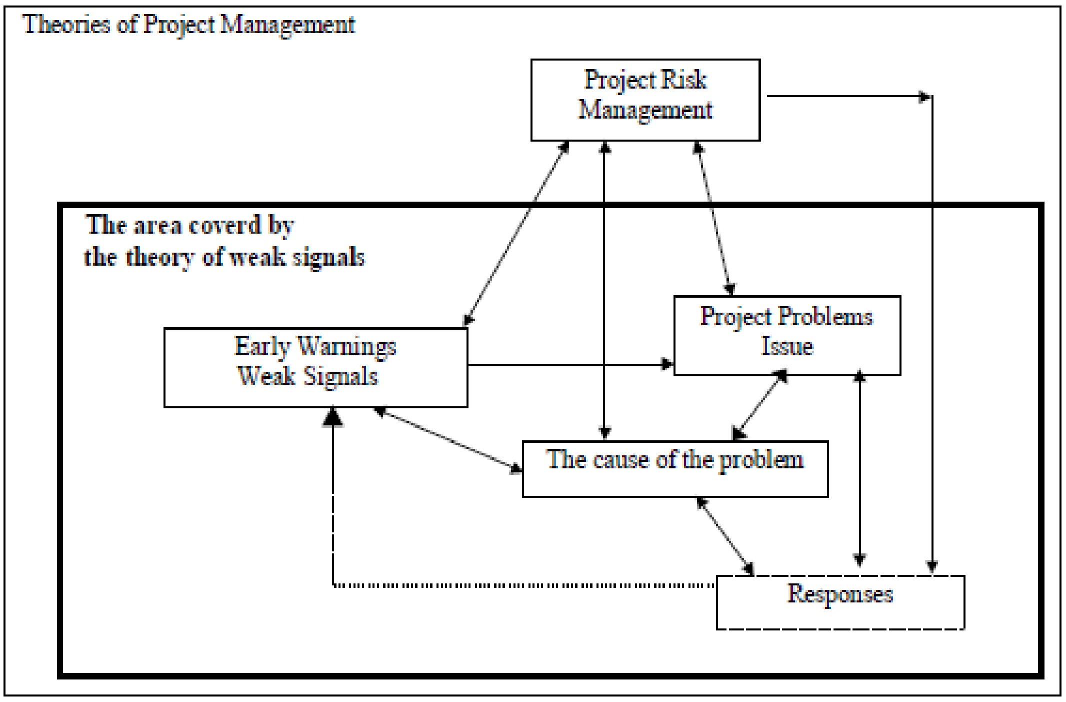

Recognizing relevant external and internal factors
Developing monitoring methods and monitoring weak signals
Interpreting weak signals
Response recommendations and the implementation of response

Figure 11: Aberg's weak signals scanning model (Aberg, 1989)

As shown in figure 12, Ansoff's theory of weak signals can be placed in a broader sense with theories of project management. Further, Ansoff's weak signals phenomenon provides insights of a probable impending issue; which coincides with the understanding of the concept of early warnings, which will be discussed in the next section.



Theories of Project Management
Project Risk Management
The area coverd by the theory of weak signals
Project Problems Issue
Early Warnings Weak Signals
The cause of the problem
Responses

Figure 12: Theory of Weak Signals in broader sense (Nikander, 2002)

2.5 Early Warnings

Ansoff's theory of weak signals has strong emphasis on strategic management. Some of the writers, even though they address Ansoff's theory by different names, have applied similar idea to different streams of business but only one (Nikander) has applied it in the field of project management. Table 2 below compares the range of Ansoff's theory to projects.

Ansoff's theory	Projects
Both look for additional information about future events	
Both include the need to anticipate the future	
The external operating environment of a company	The internal and external environment of projects
Strategic management	Daily management and project control
Unexpected significant changes	Unpredictable events, even small changes
Continuous activity over several years	The short implementation time of projects
Continuity in activity	One-time event, no repetition

Table 2: Comparison between operational ranges of Ansoff's theory to Projects (Nikander, 2002)

It's hard to apply Ansoff's theory to project work due to the difference in the scale (Nikander, 2002). However, the rational of the theory could be applied to project work environment. Dr. Nikander had made a successful attempt to link/replicate the phenomenon of weak signals to a typical project environment. He applied Ansoff's theory to project work. He called it as "early warnings". He believed the concept of "weak signals" and "early warnings" are parallel concepts (Nikander, 2002). Early warnings concept is same as of weak signals but is focussed on project environment (instead of strategic management). Nikander (2002, p. 49) formally defines the concept of early warnings as follows:

> *An early warning is an observation, a signal, a message or some other item that is or can be seen as an expression, an indication, a proof, or a sign of the existence of some future or incipient positive or negative issue. It is a signal, omen, or indication of future developments.*

Early warnings typically give information. The negative information received through the early warnings should not be ignored. The treatment of early warnings is important else an ignored minor issue may lead to a cascading effect of causes, problems, early warnings and responses chains in project environment (Problems in this context refer to any issues which hinders the completion of project. Causes are factors or elements which have led to a problem. Responses are actions taken by personnel to mitigate/minimize the effect of a problem). A typical such chain is illustrated in the figure 13below. Consider problem A, which was detected at time instance "T-n" yesterday, has its causes. This problem A can possibly raise another problem (say Problem B). Hence, Problem A can become the cause of Problem B which means there would be a causal relationship between Problem A and Problem B. If this relation is immediately noticed, then the observation of problem A can be considered as an early warning for impending problem B.

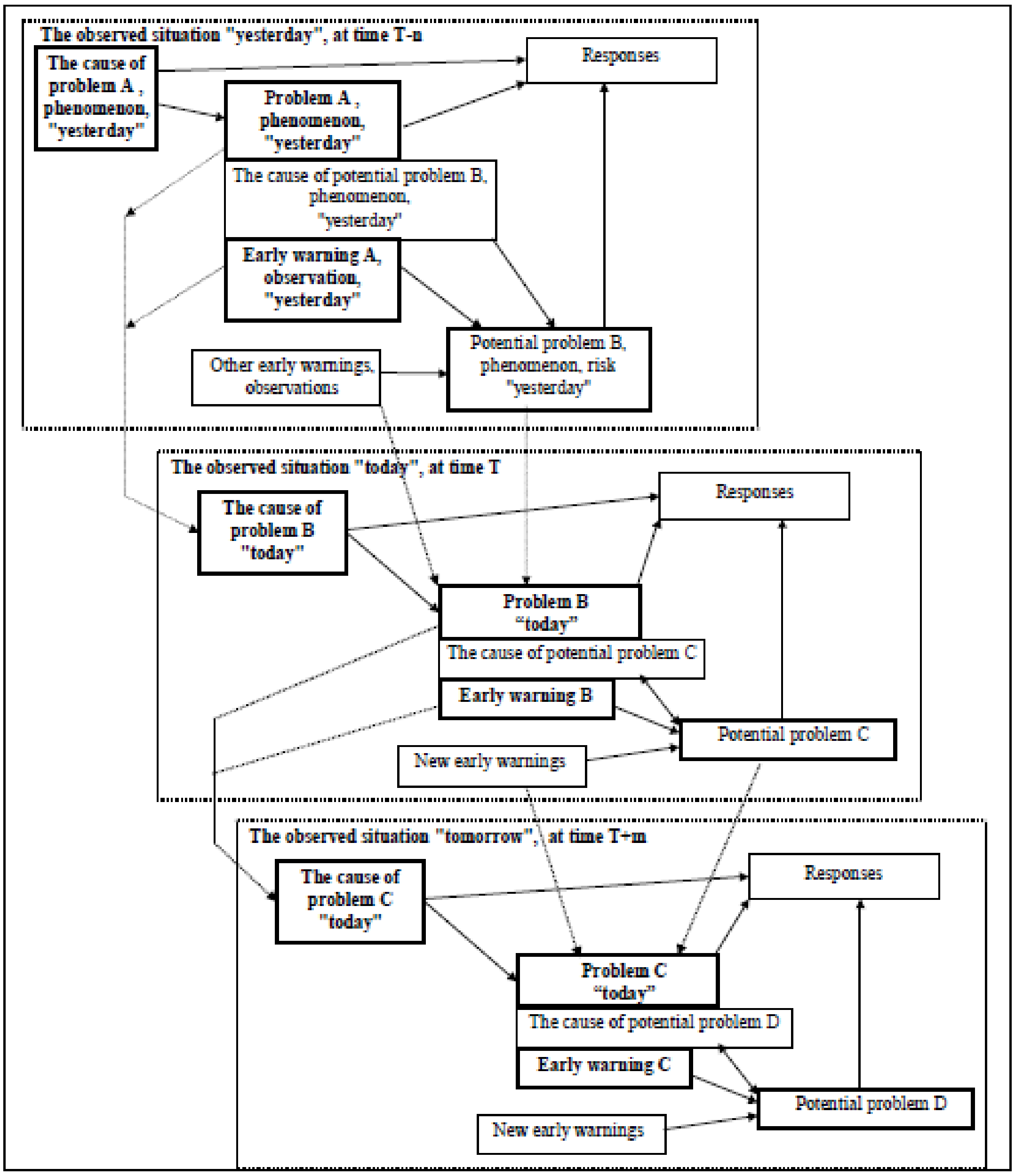

The observed situation "yesterday", at time T-n
The cause of problem A , phenomenon, "yesterday"
Responses
Problem A , phenomenon, "yesterday"
The cause of potential problem B, phenomenon, "yesterday"
Early warning A, observation, "yesterday"
Other early warnings, observations
Potential problem B, phenomenon, risk "yesterday"
The observed situation "today", at time T
The cause of problem B "today"
Responses
Problem B "today"
The cause of potential problem C
Early warning B
Potential problem C
New early warnings
The observed situation "tomorrow", at time T+m
The cause of problem C "today"
Responses
Problem C "today"
The cause of potential problem D
Early warning C
Potential problem D
New early warnings

Figure 13: Chain of Causes, Problems, Early Warnings and Responses (Nikander, 2002)

If appropriate responses are not taken to minimize the effects of problem A then problem B materializes at a time instance "T" say today and problem A becomes the cause of Problem B. Similar cycle could start with Problem B and probably raising another problem C at time instance "T+m" say tomorrow. Such a chain can cause cascading effect and end up being extremely long. Figure 14 below shows the dependencies between early warnings, causes, problems and responses.

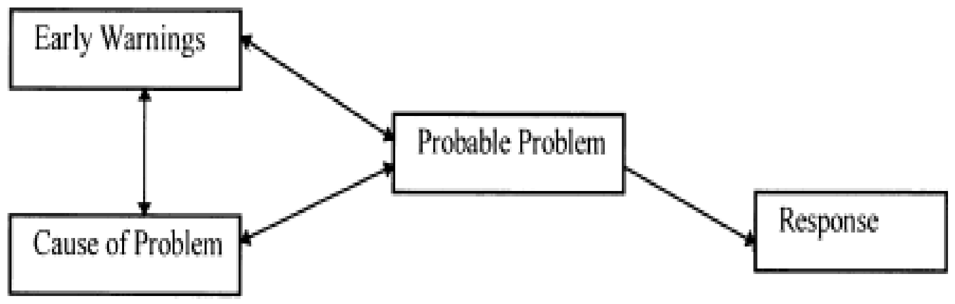


Figure 14: Dependencies between early warnings, causes, problems and responses (Nikander and Eloranta, 2001)

Hence, it implies that an early warning can easily become a cause of a problem if not observed. It is also important to note the significance of time (i.e. the application of the treatment of early warning before it becomes the cause of another problem). The treatment/response should be actioned in the time available range. Figure 15 below shows the detailed information about the sub-periods of time available.

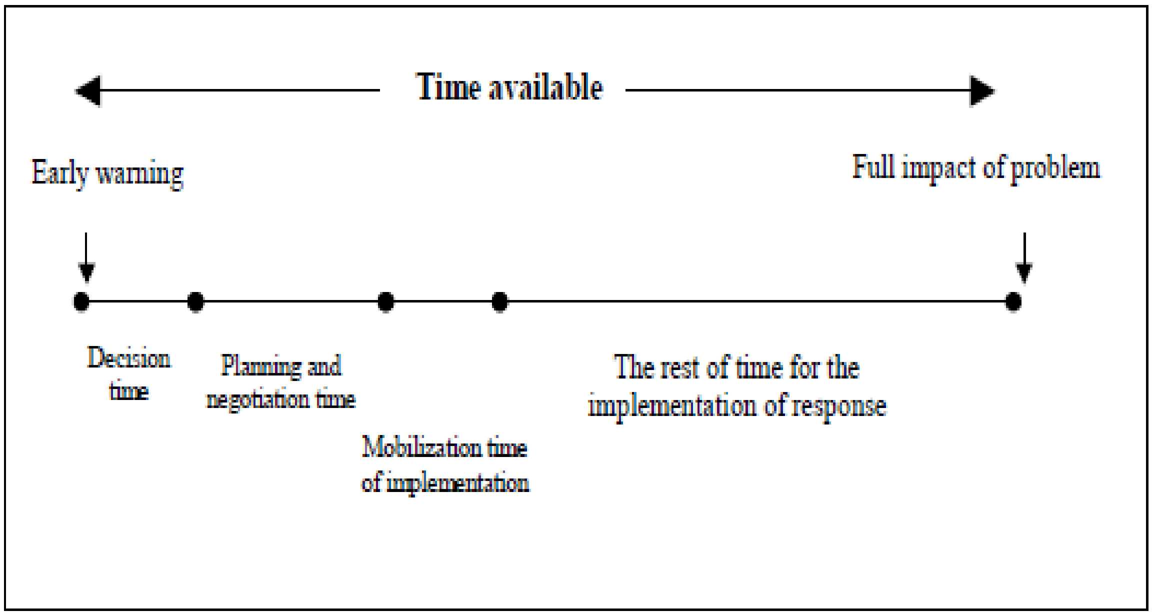

Time available
Early warning
Full impact of problem
Decision time
Planning and negotiation time
Mobilization time of implementation
The rest of time for the implementation of response

Figure 15: Sub-periods of time available (Nikander, 2002)

Time available is the total amount of time available before the problem signalled by the early warning hits with its full impact to the project (Nikander, 2002). The time available is dependent on many different factors like the time early warning was detected, current project situation and project environment, the decision making system of the organisation, the resources available to implement the response, the assumptions and beliefs of the interpreter, the type of problem and others. However, the author firmly believes that its very difficult to clearly indentify and measure the different sub-periods of time available as it would vary from organisation to organisation in fact even at the project and problem level in the same organisation. Also, Nikander (2002) hasn't provided enough information in his article about the method he used to derive these sub-periods.

2.5.1 Connection of Early Warnings and Project Risk Management

Now having described the basics of early warnings, it is important to illustrate how early warnings can be incorporated into project risk management. Project risk management has been a popular area of research in recent years. In this fast developing domain, risk management is looking forward for proactive solutions. Surprisingly, project risk management literature is very weak in this area. There hasn't been enough literature which talks about proactive methods in project risk management. Early warnings concept is certainly a potential way to make project risk management more proactive. Early warnings can potentially become the cause of a problem if not acted appropriately in the time available range and hence can be seen as a risk. In fact, Nikander (2002) asserts that the concepts “potential problem” and “risk” are somehow related and there is clear similarity between the concepts. Early warnings inform the interpreter the possibility of future problems in the project. Figure 16 below shows the interconnectedness of the concepts of early warnings and risk.

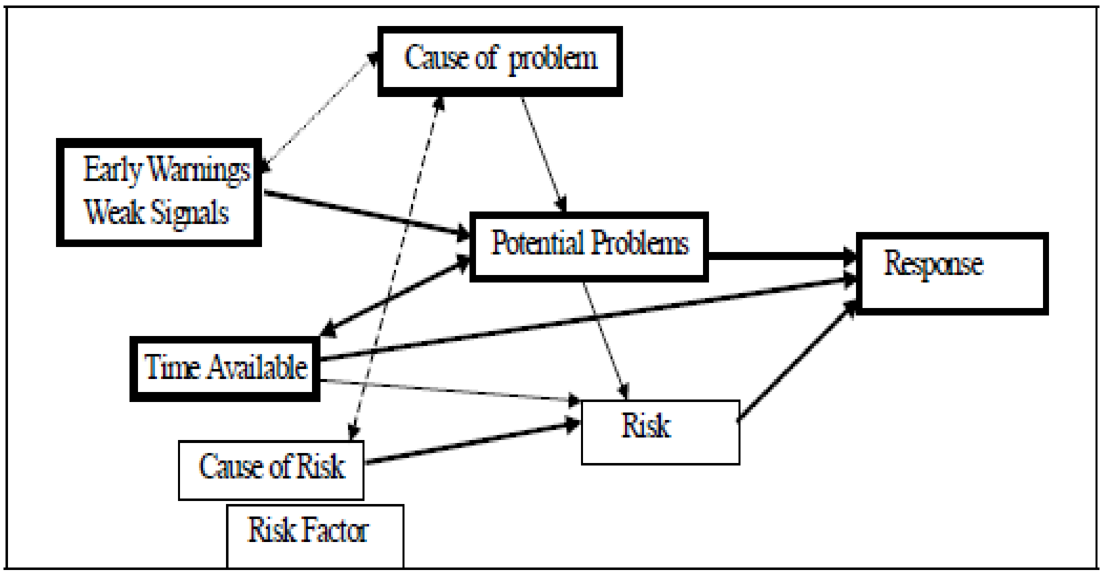


Figure 16: The interconnectedness of the concepts (Nikander, 2002)

However, Nikander (2002) believes that early warnings by itself don't provide any information about the probability of the problem (risk) to come true. They by themselves are hardly sources of future problem's information. But early warnings combined with other analytical information can make a difference. They don't convey a clear picture either, of when the problem might materialise (Nikander, 2002). In spite of that, Nikander (2002) strongly believes that the concept of early warnings should be a part of risk quantification phase of project risk management. The author believes that harnessing early warnings to identify potential risks can be a new way of being proactive in project risk management. We need to use each and every piece of available information to make the projects successful. Early warnings would enable project risk management to better manage and anticipate unforeseen project problems.

2.5.2 Character of the Early Warnings Phenomenon

Now having clear idea where early warnings sit with project management theories, the following figure 17 gives the complete character of early warnings phenomenon.

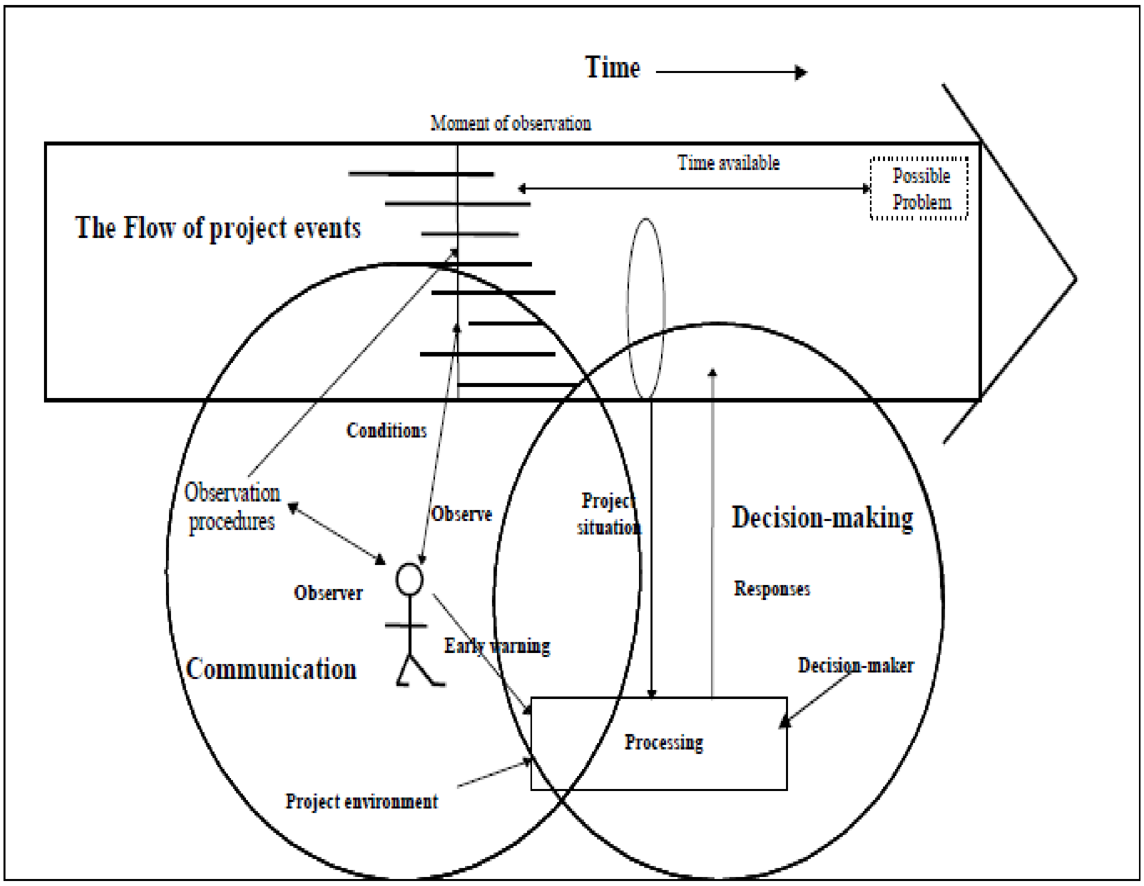


Figure 17: The Character of Early Warnings Phenomenon (Nikander, 2002)

The whole early warnings phenomenon is divided into two phases (Nikander, 2002):

1) Communication phase
2) Decision making phase

The communication phase consists of two integral functions. First is detecting the early warnings and second is its interpretation. The most important element in the communication phase is the observer who in the real sense observes and interprets the messages. The model views a project in the form of series of project events. These events are strictly time bounded. The observer obtains information from the project events. The observer uses different observation procedures to accumulate the information. He might use some of his pre-determined information of early warnings as a checklist (it may be from previous project or from his experience) to facilitate the recognition of early warnings. Once the early warning is detected it is interpreted by the observer. He would analyse the early warning and then attempt to find the impact of the warning sign which helps him to move to the decision making phase. In the decision making phase, the decision maker depending on the available information tries to identify the implications of the early warning on the projects and identifies the available responses. He then selects the appropriate responses and actions them in order to eliminate/reduce the possibility of a future problem. In order to do so, the decision maker uses decision support model which is explained in the next section. Another crucial factor which should be noted is the time available before the problem hits the project. The responses should be applied before the problem hits the project.

2.5.3 Decision Support Model of Early Warnings

The decision support model provides a base to implement the early warnings phenomenon. As illustrated in the figure 18 below, the model has six sequential stages.

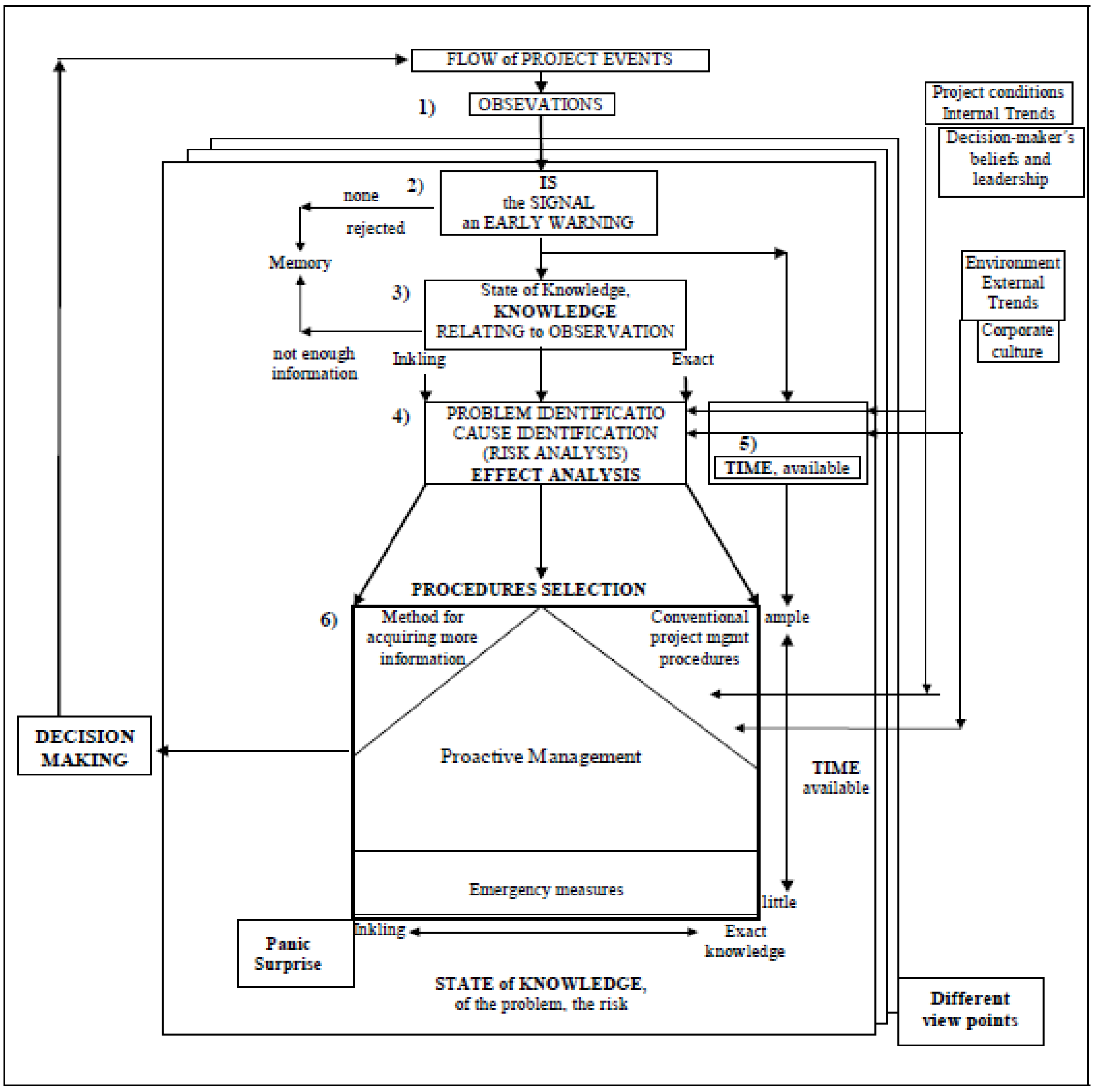

FLOW of PROJECT EVENTS
1) OBSEVATIONS
Project conditions Internal Trends
Decision-maker's beliefs and leadership
2) IS the SIGNAL an EARLY WARNING
none
rejected
Memory
3) State of Knowledge, KNOWLEDGE RELATING to OBSERVATION
not enough information
Inkling
Exact
Environment External Trends
Corporate culture
4) PROBLEM IDENTIFICATIO CAUSE IDENTIFICATION (RISK ANALYSIS) EFFECT ANALYSIS
5) TIME, available
PROCEDURES SELECTION
6)
Method for acquiring more information
Conventional project mgmt procedures
ample
Proactive Management
TIME available
DECISION MAKING
Emergency measures
little
Panic Surprise
Inkling
Exact knowledge
STATE of KNOWLEDGE, of the problem, the risk
Different view points

Figure 18: The Decision Support Model of Early Warnings (Nikander, 2002)

The first stage is observing and detecting early warnings. This task is always performed by a person in a project consisting of series of project events. The observer should be very sensitive while listening to the environment in order to detect early warnings. In fact, Ansoff suggested that observer should listen to the environment "with an ear on the ground" (as cited in Nikander, 2002). Similarly, Juran (1995) has also emphasised the importance of people as sensors in order to identify deviations or weak signals. The observer can take help of additional information (like previously known/detected early warning; how it looks like; when and where; and others) to detect early warnings. It is important to understand that each observer has its own interpretation and looks at the situation in his own perspective. He may also consider different viewpoints of his fellow project members in order to ameliorate his view and understanding. Hence the figure contains “Different view points” marked by overlapping frames around the whole process in the extreme lower right hand corner of the figure. The second stage starts with the interpretation process where the observer decides if the detected signal is an early warning. In this stage the significance or insignificance of the information is been evaluated. The detected signal may be a random, arbitrary event and irrelevant. In the third stage, the state of the early warning is been determined i.e. the implications of the detected information to the current project. If the retrieved information is too scarce or irrelevant even then it should be recorded and stored (more information on the same topic may surface in the future). In the fourth stage, the observer attempts to discover the probable problem (or risk) that emerged. Attempts are also made to identify its possible causes (causes of risk, risk factors) i.e. problem and cause identification tasks are been carried out. This entire evaluation process is influenced by many other factors like internal trends (e.g. current project situation), external trends (e.g. project environment, organisation structure), assumption and perspective of the observer, and others. The various risk analysis methods can also be utilized in this stage to facilitate the identification process. In the fifth stage, the available time is computed after recognising the effects of the risks. The question in to consideration here is: how much time is still present for the responses in order to address the problem (or risk)? It includes the discovering of the urgency of the situation. This stage is also influenced by many other factors like internal trends, external trends, assumption and perspective of the observer, and others. Ansoff’s classification (explained earlier) namely urgent, postponable and delayable can be utilised as a scale. In sixth (final) stage, efforts are made to identify appropriate responses for the impending problem. This decision making process is assisted by following (Nikander, 2002):

1) the information available about the impending problem (risk)
2) the urgency of the responses (time available)
3) internal trends
4) external trends

5) decision maker’s perspective
6) the effects of the concerned response to current and other projects

None of the acquired information should be disregarded and should be analysed carefully before making any decision. There are different situations encountered during the decision making process. The extreme point situations in the decision making matrix are as follows (Nikander, 2002):

1) If ample amount of time is available but acquired information is an inconclusive (or weak) then observation methods which would provide more information should be selected.
2) If ample amount of time is available and the acquired information about the problem is exact, then conventional project management approaches can be adopted.
3) If the time available is less and preciseness level of the acquired information lies between weak and exact, then different emergency responses should be selected.
4) If the problem is outside the matrix i.e. there was no information of the problem (risk) and hence comes as a surprise then when the risk materialises, some type of panic situation will possibly arise.

The middle part of the matrix consists of proactive management approach which was not explored by Nikander (2002) in his study. Further, Nikander (2002) has also admitted that this model is not proven and needs further studies to prove it. This study will extend this decision support model.

2.6 Gaps in the literature

Now having done thorough literature review it is possible to identify possible gaps which the current research may address.

The project management literature is full of facts and figures and whinging about the incapability of the software projects to be successful. These development projects mainly fail due to their inherent characteristics like human interaction, abstraction, complexity and uncertainty. The author believes that the "uncertainty" factor of a project can be handled in a better way than the normal traditional methods. Although the project management literature expressed the importance of it, but there has been very little work in terms of exploring different approaches to deal with uncertainty that are supported by empirical evidence. This motivated the author to identify an approach to deal with uncertainty arising from the inability to foresee or predict all events in the software development process.

The nature of modern software development is such that uncertainty will never be eliminated and therefore there is a definite need for new, innovative and proactive management style to deal with uncertainties. This could indirectly improve the chances of a project to be successful. A proactive method employing early warnings phenomenon could be implemented in project environment. Surprisingly, the project management literature is very weak in this area. Although the project management literature understands the importance of it, but there is very little research exploring this concept in the project environment. As far as the author knows, only one comprehensive study has been done in this field which was by Nikander (2002). The early warning phenomenon has considerable promise to become part of project risk management approach. However, the theories of project risk management are apparently not familiar with the early warnings phenomenon and there is little empirical evidence to support the perceived potential. The phenomenon is very poorly represented in the project risk management literature (although the idea of risk symptoms has been mentioned). The work done by Nikander (2002) is certainly a good head start but his model (especially his decision support model) needs more exploration and needs to be practically implemented and supported by a greater body of empirical evidence. Nikander (2002) has admitted the same in his work. Even though the concept of early warnings looks promising, there is very little research and information on this domain, especially if it comes to focus on empirical early warning signs (EWS). Hence this motivated to extend Nikander's work and practically implement the concept of early warning signs.

2.7 Summary

This section concludes the literature review of the software project failure phenomenon, different approaches to dealing with uncertainties, theory of weak signals and early warning signs. From the literature review it is apparent that many software projects fail to achieve their main goals in terms of scope, quality, cost and schedule. The failure rates have been astoundingly very high. The literature review also outlines different focal areas to deal with uncertainties. The author opts for Risk Management strategy and further insights in that domain were also forwarded. The concept of “theory of weak signals” was briefly explicated. A new and innovative concept namely early warnings phenomenon was also described. However, it seems that there appears to be very limited literature that focuses on early warnings phenomenon. The early warnings phenomenon can apparently be a part of project risk management. Finally, this chapter outlines the gaps in the literature. Even though this domain sounds promising but the analysis reveals that it is highly under researched area; which motivated the author to conduct this study. The following chapter will describe the research objectives and the research methodology opted by the study.

Chapter III
Research Objectives and Methodology

3.1 Introduction

Based on the outcome of the literature review, this chapter states the research objectives and the research questions of the study. It also describes the research methodology and the research design employed by the research.

3.2 Research Objectives

In the previous chapter the concept of early warning signs has been described. It appears that Nikander (2002) has done some work in this field and designed a decision support model which could be utilised to implement early warnings phenomenon. But, Nikander (2002) has also admitted that his model is not proven. This work is the first stage of an incremental approach to extend this model and provide empirical evidence to support the perceived benefits of the model. This research will focus on the implementation of proactive management style for decision making process.

It is suggested that implementing early warnings phenomenon may have positive effects on the project outcomes. But as yet there is no empirical evidence that demonstrates this. The aim of this research work is to provide some empirical evidence that there is a value in following and implementing early warnings phenomenon.

Once the early warning signs are detected and interpreted, attempts can be made to identify the probable problem and its cause(s). Also, the available time is computed. Once the probable problem is identified, different corrective actions (responses) needs be taken to bring a (would be) troubled project back on track.

A further aim of this research work is to provide some empirical evidence that there is a value in terms of undertaking corrective actions early by demonstrating that if we delay corrective actions then it has a negative impact on the project outcomes.

3.2.1 Research Questions

The following research questions are answered within this research:

- Does the implementation of early warning phenomenon have positive effects on the project outcomes?
- Is there positive impact on the project outcomes if the corrective actions are taken early?

Based on the research objectives and the identified research questions the research methodology has been described below.

3.3 Research Methodology

In order to address the research questions effectively, robustly and rigorously an appropriate methodology needs to be selected. Selection of appropriate research methodology is a crucial part of the research. Hence, before that it is important to identify the key characteristics of the research/problem domain:

- **Existing Problem:** As highlighted earlier there is a problem (the software project failure rates) in the current environment.
- **Very few previous studies:** For unknown reason there is very little research done on this domain.
- **Feasibility for Improvement:** There is a high scope for improvement.
- **Novelty and Innovative Solution:** This previously unsolved problem can be solved using new and innovative solutions.

These key characteristics of the research domain clearly align with both exploratory and constructivist in nature. Lukka (2003) defines constructive research approach as "A *research procedure for producing innovative constructions, intended to solve problems faced in the real world and, by that means, to make a contribution to the theory of the discipline in which it is applied*". It means that constructive research method intends of developing novel solutions to existing problems. According to Kasanen, Lukka and Siitonen (1993), constructive approach leads to the production of "constructions". Constructions here refer to the solutions (to a problem) which were never concocted up to now. Constructive approach stresses on building and evaluating the solutions. These solutions can be easily verified by their implementation. There is a clear element of innovation in constructive approaches (Kasanen et al., 1993).

This research utilises the System Development Research Methodology (SDRM) as suggested by Nunamaker and Chen (1990). It provides a governing framework for structuring the key activities. The research also follows the design science research guidelines as outlined by Hevner, March, Park and Ram (2004) to specify, design, develop and evaluate a corrective action model. SDRM methodology has been used extensively in software engineering and information systems development research domain and it can accommodate dynamic evolution of the research in order to create innovations, define new ideas, and develop new technical capabilities (Limbu, 2008). The next section describes the research design utilising the System Development Research Methodology (SDRM).

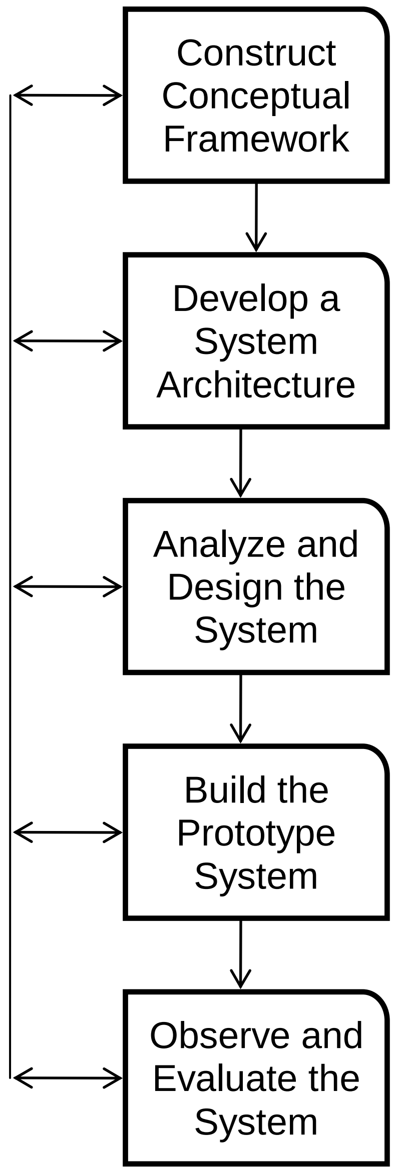

Construct Conceptual Framework
Develop a System Architecture
Analyze and Design the System
Build the Prototype System
Observe and Evaluate the System

Figure 19: Research process of the SDRM (Nunamaker and Chen, 1990)

3.4 Research Design

Based on the research objectives and the selection of the SDRM research methodology, the design of the research is as follows. Stage 1 comprises of extensive literature review of the problem domain. Stages 2, 3 and 4 comprises of creation of simulation models to address the research objectives. Stage 5 comprises of observation and evaluation of the systems using individual early warning signs (EWS) testing and controlled experimental study.

3.4.1 Construct a Conceptual Framework

This stage comprises of the definition of the problem. In order to accomplish that, this research is based on an extensive literature review of the domain under consideration. The software project failure problem has been investigated and different focal areas have been identified to deal with the uncertainties of the project. Further, early warning phenomenon has been explored. The existing state of knowledge has been synthesised and then meaningful research objectives have been derived based on the gaps and the limitations of the current situation. The research objectives are new, innovative and important in the field.

3.4.2 Design and Implementation

Within the context of the research, this phase represents the “Develop a System Architecture”, “Analyse and Design the System” and “Build the Prototype System” stages of the SDRM. SDRM lends support to the design and implementation of a solution that address the research objectives. In order to design a solution that has the potential to generate empirical evidence, a simulation tool known as SimSE was used to create models which addressed the research objectives. The software development industry has witnessed a boost in the use of simulation modelling especially in the field of software process improvement (Kellner, Madachy and Raffo, 1999). It has also been used for predicting the project costing; planning and re-planning activities; learning; tracking purposes; training; strategic management; tactical management; technology adoption; risk analysis; control and operational management; and others (Christie, 1999).

Simulation already has a long and successful history in aerospace and power industries (Christie, 1999). Simulation is a powerful tool to acquire a thorough understanding of complex behaviour and for conducting experiments and studies that cannot be easily carried out in actual environment. It provides useful insights if the data driving the model reflects the real world (Christie, 1999). These insights can provide organisations competitive advantages. Simulation tools have the capability of simulating the behaviour of real processes; however the input data driving the model should be accurate. The challenging part is the analysis of the data as it directly relates to the evaluation of the experiment. Simulations are most useful when conducting an experiment in the real world is too expensive or too difficult. Simulations provide a means to generate new and innovative solutions which would have been very expensive and difficult in the real world scenario. The use of simulation is therefore very appropriate for this research, as it allows the use of many more simulated software projects than real projects would be achieved in the same timescale. This research is essentially a pilot study that is investigating whether early reaction to perceived problems has the potential to direct a project back to a successful conclusion. Therefore a study that is more broad through the use of simulation can provide this evidence, which if positive will then support the implementation of a deeper study focused on real world projects.

In this research, simulation is therefore used to simulate the behaviour of a software development project including the allocation of activities to the staff of different experience levels by the user (the player) to achieve the project objectives. The user hence acts as a project manager and is responsible for the project outcomes. Models with appropriate structure and data (as demanded by the situation) are designed using the simulation tool. Several prototypes are constructed and repetitively iterated based on the feedback provided by the supervisors until the final models are ultimately available for the experiments. Significant time was spent on constructing the simulation models.

One of the research questions is to investigate if the implementation of early warnings phenomenon has positive effects on the project outcomes. This demands implementation of multiple early warnings signs in one integrated

EWS model. The model was built using SimSE tool and tested using controlled experimental study.

The second research question is to determine if there is a value in terms of undertaking corrective actions early. This was accomplished by designing a separate model for each early warning sign and then taking corrective actions at different time period. The evaluation was done in individual EWS testing experimental phase. More information about the models and its evaluation can be found in next chapter.

3.4.3 Observe and Evaluate the System

Once the model has been built, SDRM requires the observation and evaluation of the constructed model in order to answer the research questions. SDRM strongly emphasises on the evolution of the product and suggests that depending on the observed new phenomena it may lead to the discovery of new theory (Nunamaker and Chen, 1990). Evaluation thus becomes a very crucial component of the research design. The evaluation of the artifact should be rigorous as it determines the quality, utility and the efficacy of the designed component (Hevner et al., 2004). Hence the following two experimental techniques were utilised by this research:

a) Individual EWS Testing
b) Controlled experimental study

Adoption of mixed methods adds the required rigor element in the evaluation process and also was demanded by the situation. The evaluation process was focussed on answering the research questions effectively.

3.4.3.1 Individual EWS Testing

The individual EWS testing phase was carried out by the researcher. The objective of this phase was to ensure that each of the multiple early warning signs implemented by the model had positive effects on the project outcomes if the early warning signs were acted upon by the user early (sooner the better). A separate model was designed for each early warning sign. In order to demonstrate the objective, pseudo-experiments were conducted on each model (representing one early warning sign) and then evaluated against the evaluation parameters. The detailed description can be found in the experiment design section (next chapter) of the thesis.

3.4.3.2 Controlled Experimental Study

Controlled experiments personify one of the best scientific designs for establishing a causal relationship between changes and their influence on user-observable behaviour (Kohavi, Henne and Sommerfield, 2007). Controlled experiments typically generate large amounts of data, which can be analyzed to acquire deeper understanding of the factors influencing the outcome of interest, leading to new hypotheses and creating a virtuous cycle of improvements (Kohavi et al., 2007). This experimental design methodology tests for causal relationships and is very commonly used in explorative researches. In a controlled experiment, an independent variable is the only factor that is allowed to be adjusted, with the dependent variable as the factor that the independent variable will affect.

Multiple early warning signs were implemented and integrated in one integrated EWS model. This simulates a realistic and dynamic project environment where multiple early warning signs can trigger simultaneously. The main goal of the controlled experimental study was to demonstrate that implementing early warning phenomenon has positive effects on the project outcomes. 10 users participated in the controlled experimental study for the evaluation of the model. The detailed description can be found in the experiment design section (next chapter) of the thesis.

Experiments were conducted which contributed data to this aspect. These data was analysed and evaluated. The results were then discussed and the answers for the research questions were acquired. More detailed information of the evaluations and results will be provided in subsequent chapters.

Chapter IV
Results and Analysis

4.1 Introduction

This chapter describes in detail the early warning signs used in this research, the experiment design and the experiment results, driven by the research objectives and the selected research methodology which have been explained in previous chapter. It also provides information about the simulation tool SimSE used for this research.

4.2 Early Warning Signs (EWS)

In Chapter 2, the concept of early warning signs was explained. Through literature review, a total of 53 early warning signs were identified (see table A.1 in Appendix A). Out of those, following six important early warning signs (EWS) were selected and implemented using SimSE tool.

- **1 - Lack of top management support or commitment to the project:** It is very common that employees concentrate more on activities that their management considers important. And hence, it is not a surprise that this is the top rated EWS. Projects which get started from the bottom up face highest problems in terms of getting the required resource and support from the management (Kappelman, McKeeman and Zhang, 2006). IT projects gets stuck in enterprise politics if there is a fundamental disagreement between overall organisation priorities (Kappelman et al., 2006). In cases where the commitment between the top management and project is weak, middle managers do not deem it as an important project and hence do not allocate the required resources to the project which leads to project failure (Kappelman et al., 2006). Absence of top management in initial important meeting(s) of the project is a typical sign of weak commitment towards the project.

- **3 - Project manager(s) cannot effectively lead the team and communicate with clients:** Project managers who cannot effectively lead the team pose a serious threat to the project success (Kappelman et al., 2006). It is sometimes observed that successful developers or analysts are promoted to project managers; however the jobs are fundamentally different (like sales management and sales). Instead of performing the effort, the project manager has to plan and coordinate the effort (Kappelman et al., 2006). Communication between the stakeholders and with the staff is the key for the success of project management. Leadership is another essential project manager skill. Managers lacking these skills may lead to project failure (Kappelman et al., 2006). Project team members not allocated enough or appropriate tasks are indicators of a weak manager.

- **5 - Project stakeholders have not been interviewed for project requirements:** Every significant project has a number of stakeholders. These stakeholders contribute to the project requirements. If the stakeholders are not appropriately engaged in the requirements engineering process and not interviewed for the project requirements then it is guaranteed that the project requirements will not be up to the mark (Kappelman et al., 2006). These weak requirements may lead to development of a solution which the stakeholders were not expecting.

This may therefore lead to change in requirements later during the project. This late change will disturb the scope, budget and timelines of the project and may lead to its failure.

- **9 - Communication breakdown among project stakeholders:** Project success is a result of ongoing effort among project stakeholders. In this competitive world, change during the life of the project is unavoidable due to various reasons like competitor's strategic moves, resource availability, laws and regulations, and others (Kappelman et al., 2006). These change needs to be communicated and approved among the project stakeholders. There is little chance of completing the project successfully if there is communication breakdown among them.

- **10 - Key project stakeholders do not participate in major review meetings:** If the key project stakeholders do not participate in major review meetings then it is a sign of either communication breakdown or stakeholders are not engaged enough in the project. This also indicates that the project may not be of a high priority to the stakeholders and this may lead to resource issues as important resources would be assigned to other higher priority projects (Kappelman et al., 2006).

- **11 - Project team members do not have required knowledge/skills:** If the project team members do not have required skills to achieve the target estimated by the management then surely the project is getting into wrong direction. This risk is mostly encountered when the project is using novel technology or the complexity is high (Kappelman et al., 2006). The management needs to make sure that the project team members acquire the required skills (if they don't have already) before they start the project.

The reason for selecting the above early warning signs (EWS) out of 53 was mainly due to the implementation feasibility using SimSE tool that will enable the facilitation of the generation of empirical data. Some EWS were simply not possible to implement using SimSE tool and some were ignored due to time constraints. But the idea is that if we can demonstrate and prove the objectives using 6 EWS then it should be scalable and also work with more EWS. Implementation of more EWS is something which should be considered as future work for this research. The next section provides information about SimSE simulation tool which was used to implement these six EWS.

4.3 SimSE

The simulation tool used in this research is SimSE. SimSE was designed as an educational, interactive and graphical computer based environment which allows the creation and simulation of software engineering processes (Navarro and Hoek, 2004), however it has a much greater potential than just education. It allows the simulation of realistic software engineering environments comprised of real world components like people, budget, large complex projects, planning, deadlines, unexpected events and others (Navarro and Hoek, 2007). It is designed to provide a platform to experience different aspects of software engineering in a practical manner. SimSE is a single player based game where the player acts like a project manager and is in charge of taking all the decisions and actions which would lead to a successful piece of software product. In order to successfully complete an assigned task the player has to manage a team of developers (Navarro and Hoek, 2007). The typical activities that can be carried in the game are assigning tasks to appropriate developer, hiring and firing the employees, purchasing the tools, scheduling the meetings with customers, and others. At any point of the game the progress can be monitored by checking the values of the artefacts (attributes) like percent completeness of the design document, implementation completeness of the code, integration completeness of the code, correctness and erroneous of the code, completeness of the acceptance tests and others. Figure 20 below illustrates the architecture of SimSE.

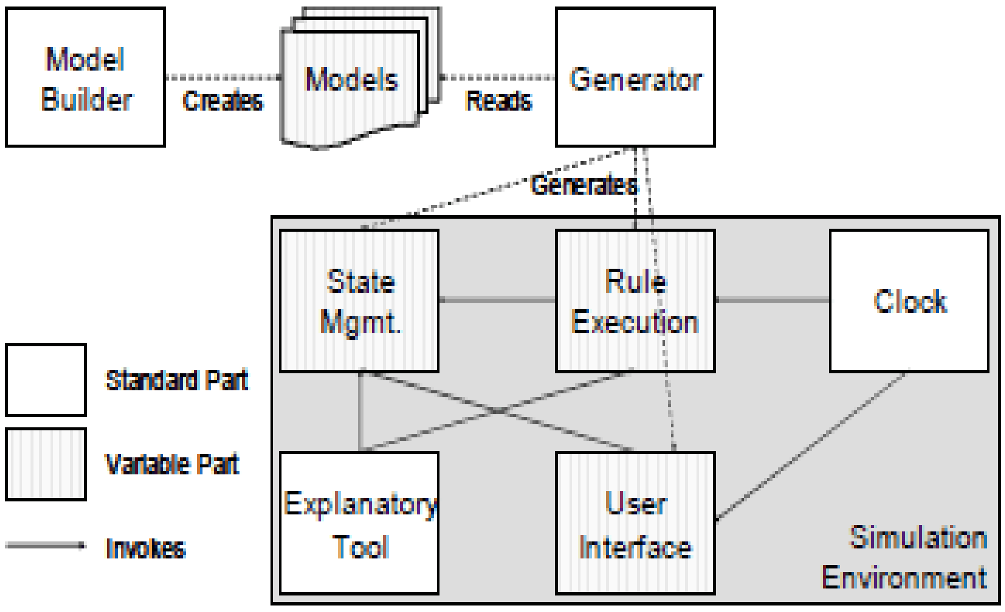
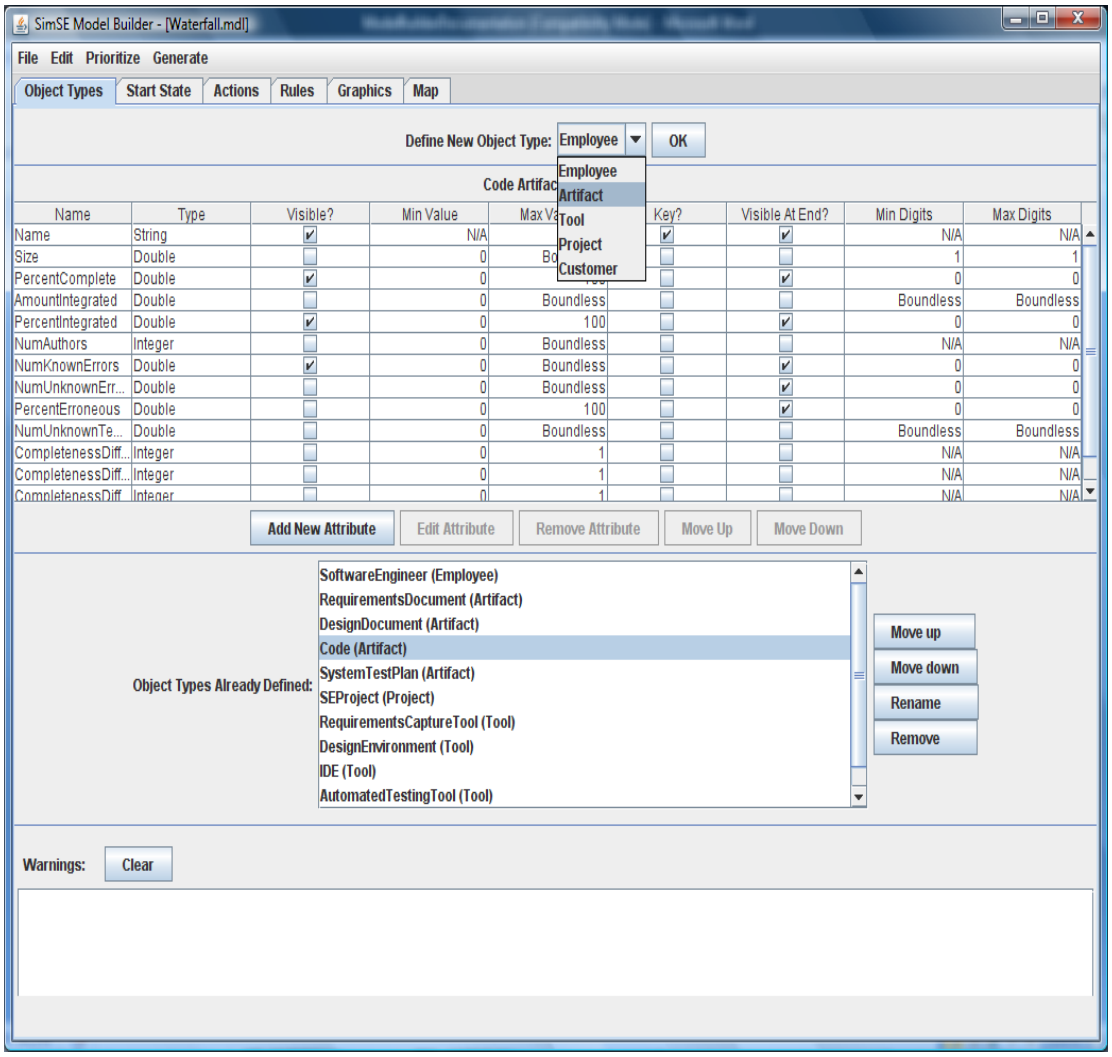


Figure 20: SimSE Architecture (Navarro and Hoek, 2004)

The model builder is used to create models that specify the details of projects, artefacts, employees, customers and tools in the simulation. It also allows defining the activities these entities can participate in and the governing rules which determine the behaviour of the game (Navarro and Hoek, 2004). The resulting model is a set of mathematical and logical relationships in the form of rules which represents real world software engineering environment. It is acknowledged that these rules may not precisely represent exact real world phenomena in a quantifiable way. However, the rules are grounded in research that defines qualitatively the typical characteristics that are visible in software projects (Navarro, 2006). Given the huge differences that can exist in software projects in terms of scale, scope, technology and people; it is assumed that the qualitative grounding (represented by the SimSE models) provides sufficient robustness and rigour for this current research.

The model builder completely hides the underlying modelling language (which is Java). The generator interprets the model and automatically generates the executable java code. The SimSE model builder consists of following five parts which together constitutes to a SimSE model (Navarro and Hoek, 2004):

- **Object Builder:** It is the first step for building a SimSE model. It is used to define the object types which would be used in the SimSE model. Any participating entity of the model will be an instantiation of an object type. A defined object type must be a descent of one of the following five meta-types: Artifact, Tool, Customer, Employee and Project. A set of attributes can be defined for object types as shown in figure 21. For example, a Software Developer would be an instance of meta-type Employee and may have set of associated attributes like Name (String), Software development experience (Integer), Pay rate (Double), Productivity (Double) and others.



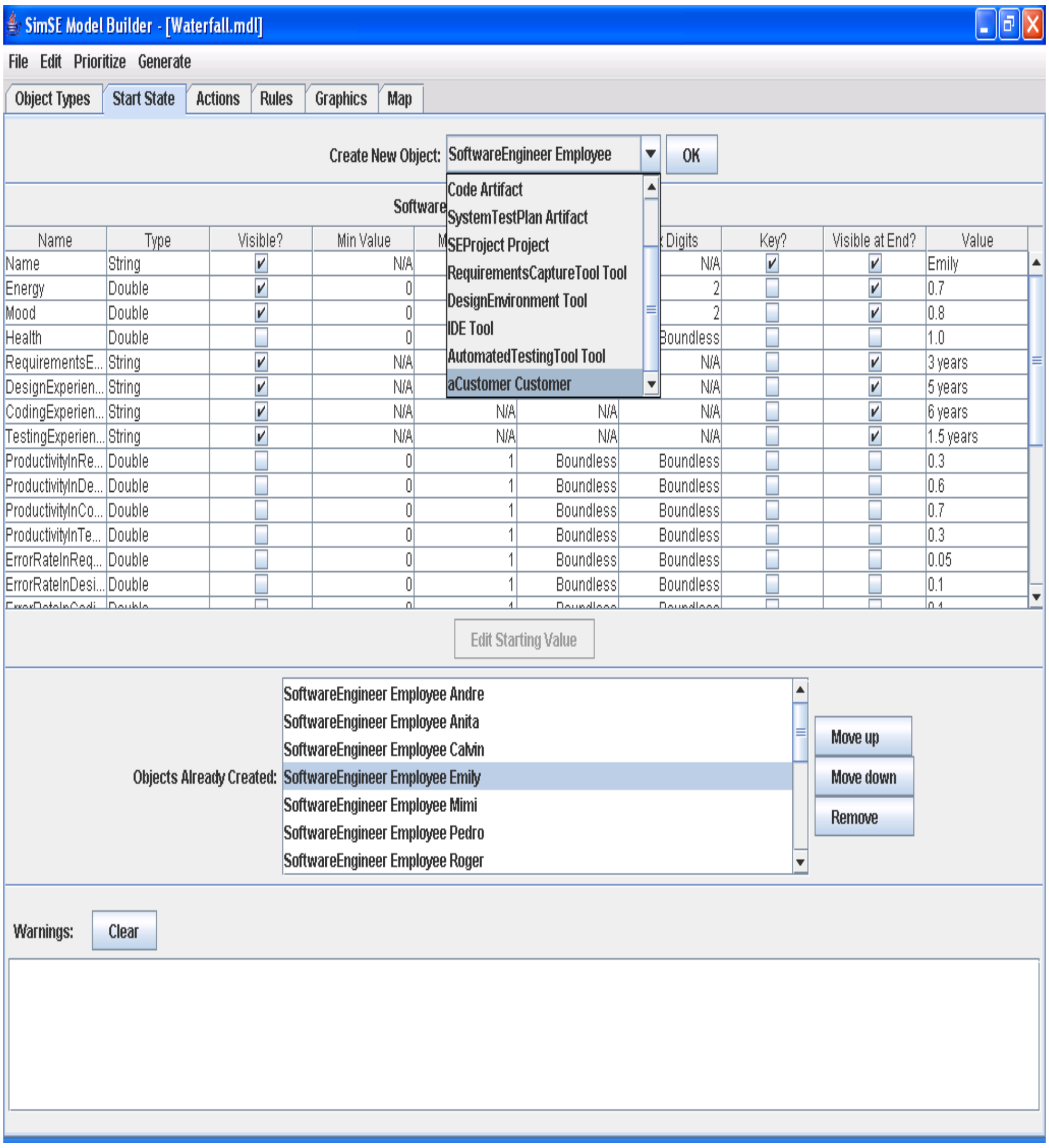
Name	Type	Visible?	Min Value	Max Va	Key?	Visible At End?	Min Digits	Max Digits
Name	String	✓	N/A		✓	✓	N/A	N/A
Size	Double		0	Bo			1	1
PercentComplete	Double	✓	0			✓	0	0
AmountIntegrated	Double		0	Boundless			Boundless	Boundless
PercentIntegrated	Double	✓	0	100		✓	0	0
NumAuthors	Integer		0	Boundless			N/A	N/A
NumKnownErrors	Double	✓	0	Boundless		✓	0	0
NumUnknownErr...	Double		0	Boundless		✓	0	0
PercentErroneous	Double		0	100		✓	0	0
NumUnknownTe...	Double		0	Boundless			Boundless	Boundless
CompletenessDiff...	Integer		0	1			N/A	N/A
CompletenessDiff...	Integer		0	1			N/A	N/A
CompletenessDiff	Integer		0	1			N/A	N/A

Object Types Already Defined:

SoftwareEngineer (Employee)
RequirementsDocument (Artifact)
DesignDocument (Artifact)
Code (Artifact)
SystemTestPlan (Artifact)
SEProject (Project)
RequirementsCaptureTool (Tool)
DesignEnvironment (Tool)
IDE (Tool)
AutomatedTestingTool (Tool)

Figure 21: SimSE Model Builder - Object Builder

- **Start State Builder:** Once the object builder defines the object types, the start state builder can be used to define the start state of the simulation. The start state refers to all the objects which are present when the simulation starts. Each object is an instantiation of one of the object types as defined in the object builder as shown in figure 22. It also allows specifying the start values of all the attributes of the objects. For example, a ‘Software Developer A’ may have start values like A (Name), 7 (Experience), 45 (Pay rate), 1 (Productivity) whereas ‘Software Developer B’ may have values like B, 4.5, 30.50, 0.75.



Create New Object: SoftwareEngineer Employee

Code Artifact
SystemTestPlan Artifact
SEProject Project
RequirementsCaptureTool Tool
DesignEnvironment Tool
IDE Tool
AutomatedTestingTool Tool
aCustomer Customer

Software

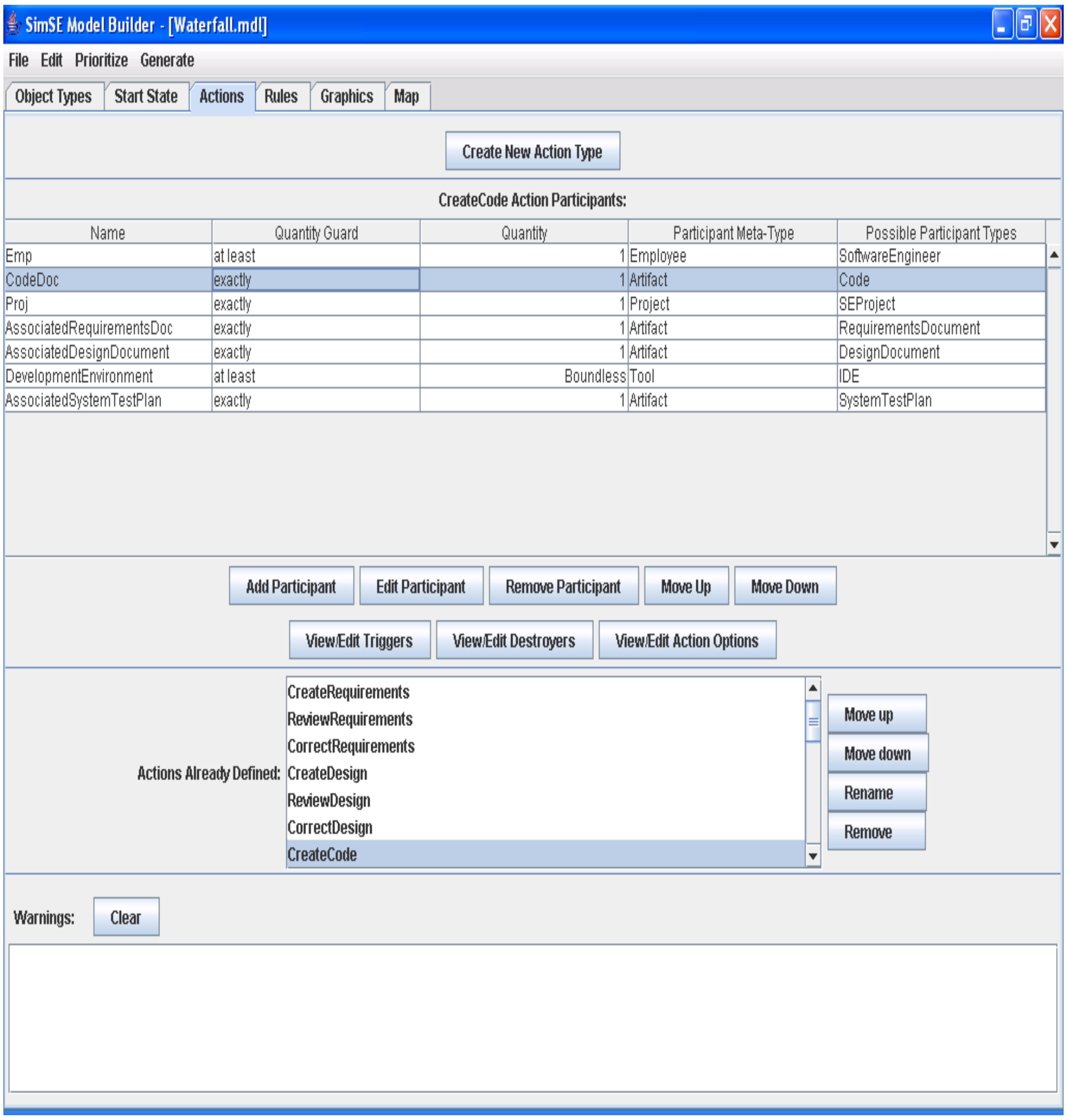
Name	Type	Visible?	Min Value	M		Digits	Key?	Visible at End?	Value
Name	String	✓	N/A			N/A	✓	✓	Emily
Energy	Double	✓	0			2		✓	0.7
Mood	Double	✓	0			2		✓	0.8
Health	Double		0			Boundless			1.0
RequirementsE...	String	✓	N/A			N/A		✓	3 years
DesignExperien...	String	✓	N/A			N/A		✓	5 years
CodingExperien...	String	✓	N/A	N/A	N/A	N/A		✓	6 years
TestingExperien...	String	✓	N/A	N/A	N/A	N/A		✓	1.5 years
ProductivityInRe...	Double		0	1	Boundless	Boundless			0.3
ProductivityInDe...	Double		0	1	Boundless	Boundless			0.6
ProductivityInCo...	Double		0	1	Boundless	Boundless			0.7
ProductivityInTe...	Double		0	1	Boundless	Boundless			0.3
ErrorRateInReq...	Double		0	1	Boundless	Boundless			0.05
ErrorRateInDesi...	Double		0	1	Boundless	Boundless			0.1

Objects Already Created:
SoftwareEngineer Employee Andre
SoftwareEngineer Employee Anita
SoftwareEngineer Employee Calvin
SoftwareEngineer Employee Emily
SoftwareEngineer Employee Mimi
SoftwareEngineer Employee Pedro
SoftwareEngineer Employee Roger

Warnings:

Figure 22: SimSE Model Builder - Start State Builder

- **Action Builder:** Action builder allows defining set of activities or actions in which objects can participate in (as shown in figure 23). For each action, it allows to define specific information like name, participant objects, condition that would trigger this action (action trigger) and condition that would stop this action (action destroyer). For example, a 'Test' artefact with one or more 'Tester' employees and one or more 'Test tools' could participate in a 'Testing' action, in which testers test a piece of software using appropriate test tools. The action trigger could be once the coding is 100% completed and the action destroyer could be once the testing is 100% complete.


SimSE Model Builder - [Waterfall.mdl]
File Edit Prioritize Generate
Object Types Start State Actions Rules Graphics Map
Create New Action Type
Add Participant Edit Participant Remove Participant Move Up Move Down
View/Edit Triggers View/Edit Destroyers View/Edit Action Options
Move up Move down Rename Remove
Warnings: Clear

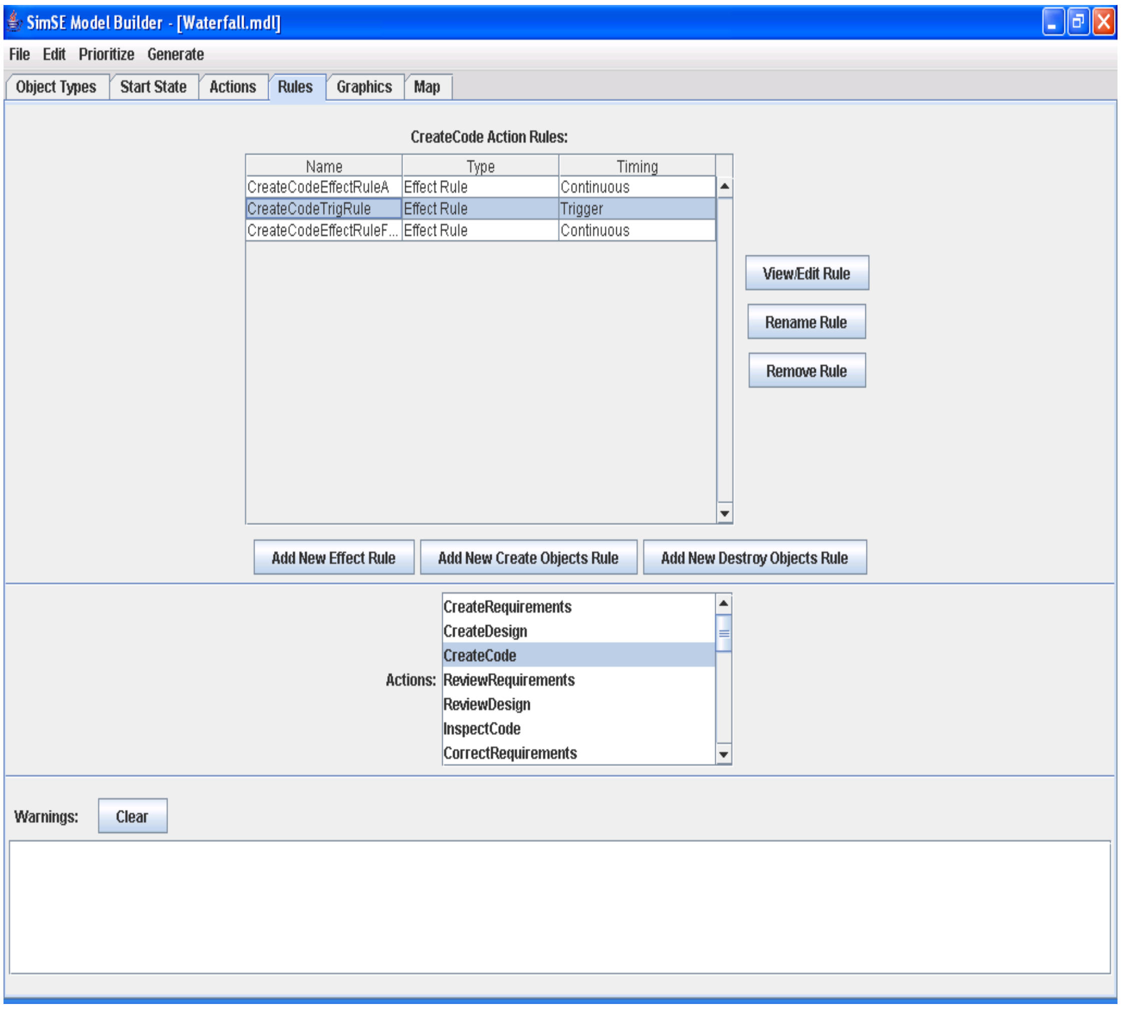
CreateCode Action Participants:

Name	Quantity Guard	Quantity	Participant Meta-Type	Possible Participant Types
Emp	at least	1	Employee	SoftwareEngineer
CodeDoc	exactly	1	Artifact	Code
Proj	exactly	1	Project	SEProject
AssociatedRequirementsDoc	exactly	1	Artifact	RequirementsDocument
AssociatedDesignDocument	exactly	1	Artifact	DesignDocument
DevelopmentEnvironment	at least	Boundless	Tool	IDE
AssociatedSystemTestPlan	exactly	1	Artifact	SystemTestPlan

Actions Already Defined: CreateRequirements, ReviewRequirements, CorrectRequirements, CreateDesign, ReviewDesign, CorrectDesign, CreateCode

Figure 23: SimSE Model Builder - Action Builder

- **Rule Builder:** The next task in building SimSE model is defining the rules for each and every action. Rule builder facilitates this. A rule is basically an effect of an active action on the simulation. A SimSE model classifies three different types of rules: *create objects rules; destroy objects rules;* and *effect rules* (as shown in figure 24) (Navarro and Hoek, 2004). Create objects rules causes creation of new objects whereas destroy objects rules causes deletion/removal of existing objects in the game. Effect rules allow specifying complex effects of an action on the attributes of the participants. For example, a 'Hiring' action may include create object rule for adding new employee; a 'Firing' action may include destroy rule for removing an existing employee; a 'Testing' action may include an effect rule for increasing the percent completeness of the testing performed depending on the testers currently working.


SimSE Model Builder - [Waterfall.mdl]
File Edit Prioritize Generate
Object Types
Start State
Actions
Rules
Graphics
Map
View/Edit Rule
Rename Rule
Remove Rule
Add New Effect Rule
Add New Create Objects Rule
Add New Destroy Objects Rule
Warnings:
Clear

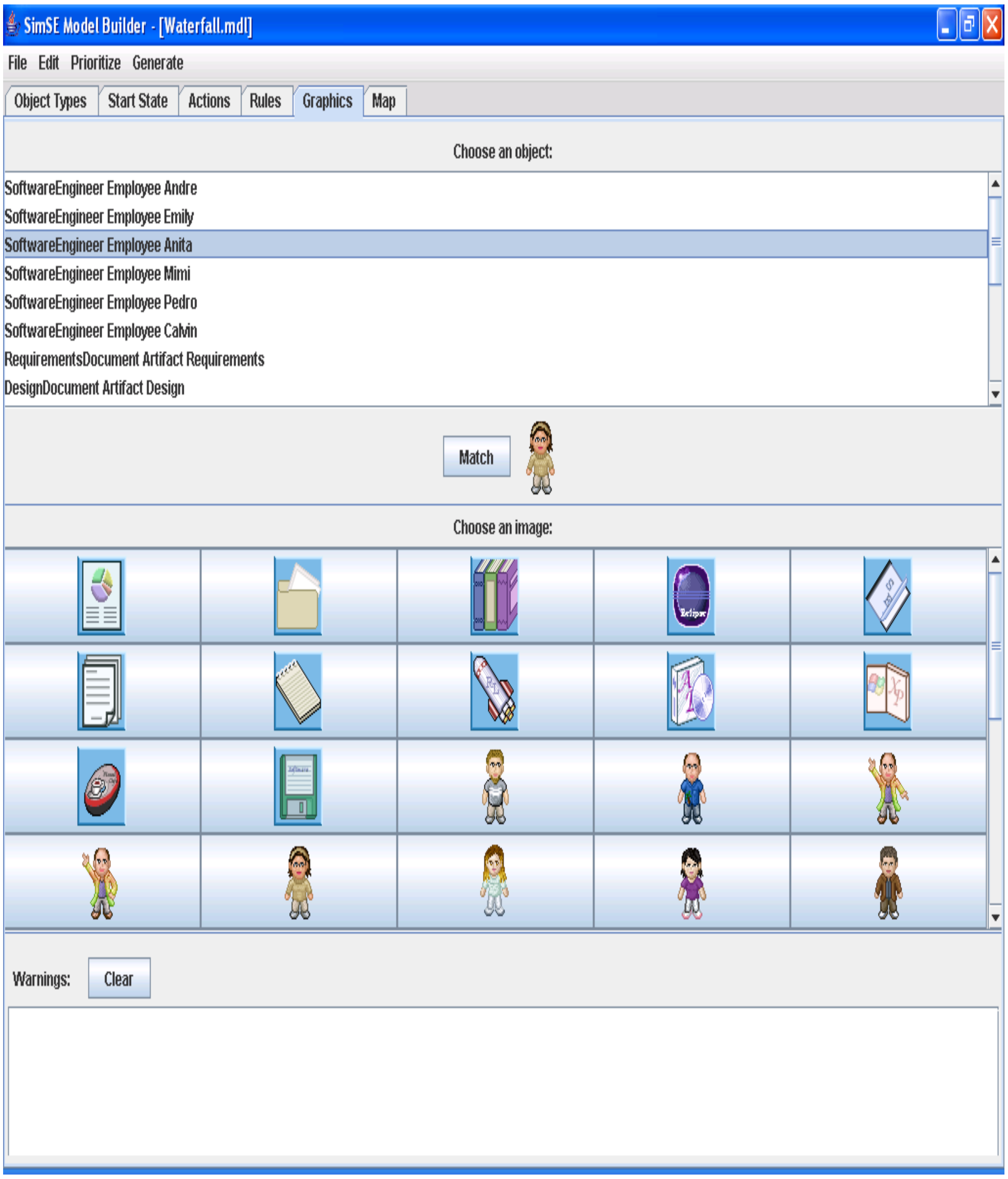
CreateCode Action Rules:

Name	Type	Timing
CreateCodeEffectRuleA	Effect Rule	Continuous
CreateCodeTrigRule	Effect Rule	Trigger
CreateCodeEffectRuleF...	Effect Rule	Continuous

Actions:
CreateRequirements
CreateDesign
CreateCode
ReviewRequirements
ReviewDesign
InspectCode
CorrectRequirements

Figure 24: SimSE Model Builder - Rule Builder

- **Graphics Builder:** The graphics builder is the final activity in the SimSE model building process and allows assigning the graphical images to each and every objects of the model (as shown in figure 25). Further, it also allows to define the office layout by specifying the location of objects, chairs, doors, walls, desks and others (as shown in figure 26).


SimSE Model Builder - [Waterfall.mdl]
File Edit Prioritize Generate
Object Types Start State Actions Rules Graphics Map
Choose an object:
Match
Choose an image:
Warnings: Clear

SoftwareEngineer Employee Andre
SoftwareEngineer Employee Emily
SoftwareEngineer Employee Anita
SoftwareEngineer Employee Mimi
SoftwareEngineer Employee Pedro
SoftwareEngineer Employee Calvin
RequirementsDocument Artifact Requirements
DesignDocument Artifact Design

Figure 25: SimSE Model Builder - Graphics Builder

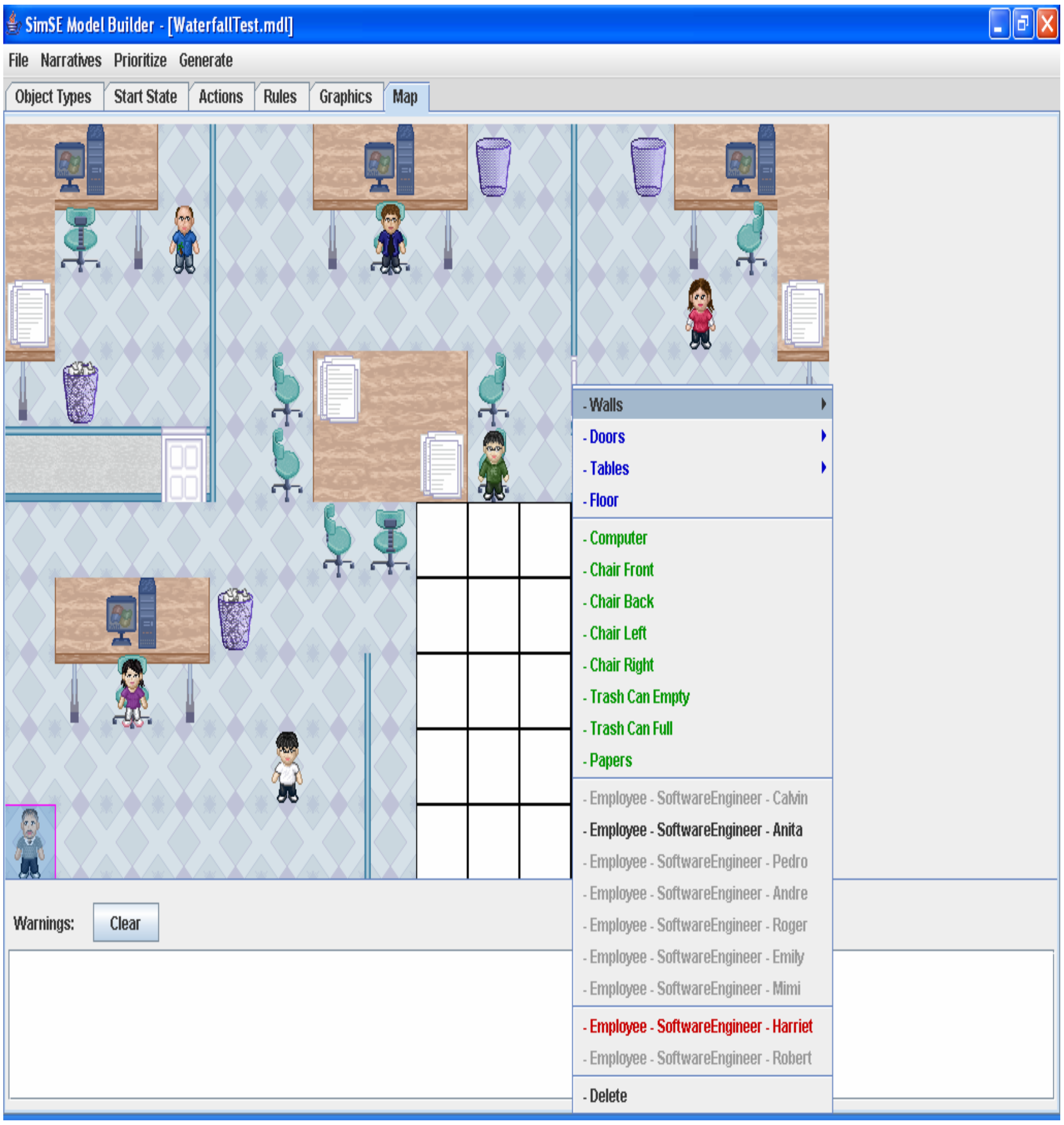
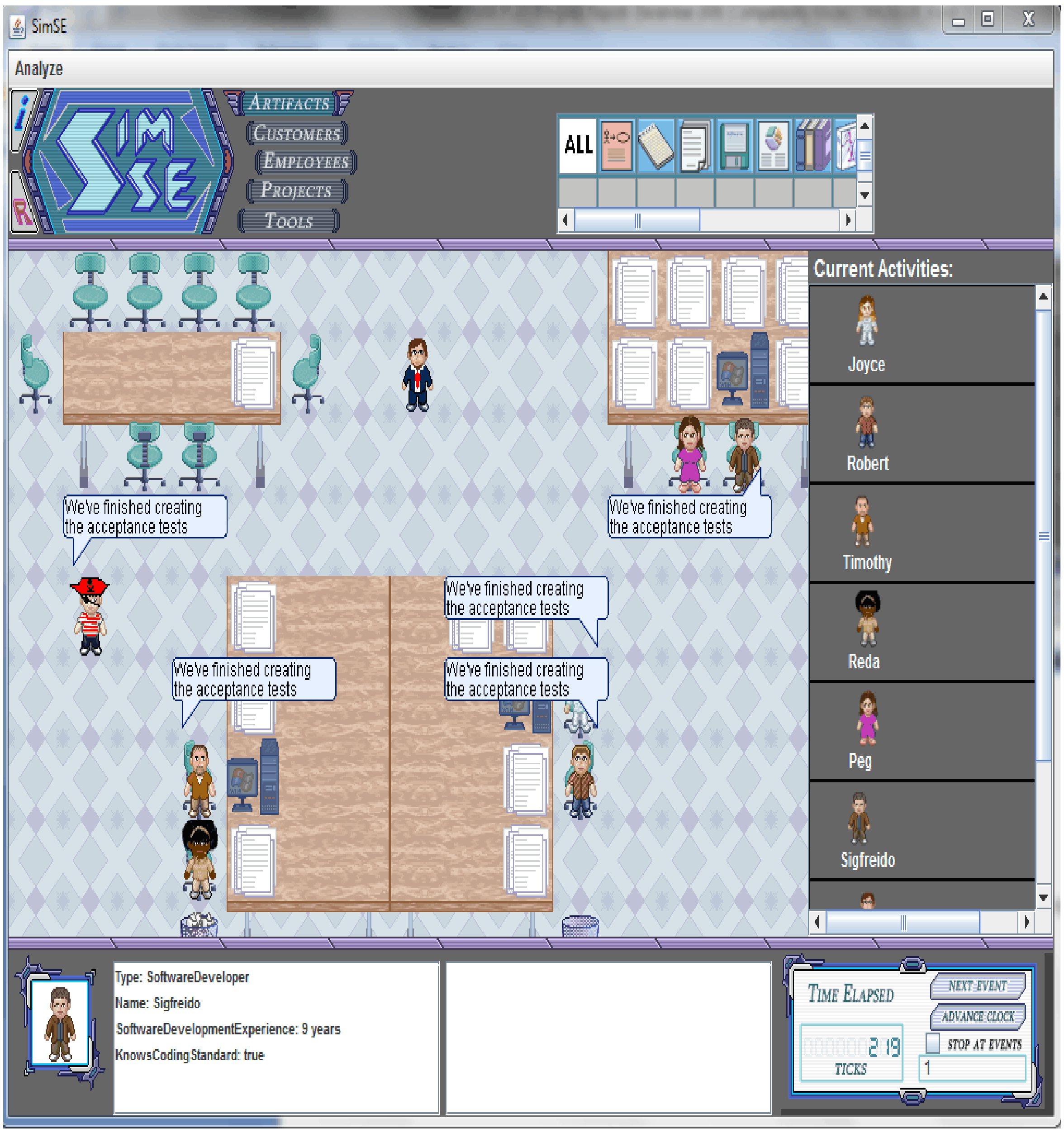

SimSE Model Builder - [WaterfallTest.mdl]
File Narratives Prioritize Generate
Object Types Start State Actions Rules Graphics Map
- Walls
- Doors
- Tables
- Floor
- Computer
- Chair Front
- Chair Back
- Chair Left
- Chair Right
- Trash Can Empty
- Trash Can Full
- Papers
- Employee - SoftwareEngineer - Calvin
- Employee - SoftwareEngineer - Anita
- Employee - SoftwareEngineer - Pedro
- Employee - SoftwareEngineer - Andre
- Employee - SoftwareEngineer - Roger
- Employee - SoftwareEngineer - Emily
- Employee - SoftwareEngineer - Mimi
- Employee - SoftwareEngineer - Harriet
- Employee - SoftwareEngineer - Robert
- Delete
Warnings: Clear

Figure 26: SimSE Model Builder - Graphics Builder

4.3.1 Simulation Environment

Once the model is built using the model builder and the code is generated using the generator the user is able to witness the simulation environment. One of the fundamental features of SimSE is its graphical user interface (as shown in figure 27).


SimSE
Analyze
Artifacts
Customers
Employees
Projects
Tools
ALL
Current Activities:
Joyce
Robert
Timothy
Reda
Peg
Sigfreido
We've finished creating the acceptance tests
We've finished creating the acceptance tests
We've finished creating the acceptance tests
We've finished creating the acceptance tests
We've finished creating the acceptance tests
Time Elapsed
219
TICKS
NEXT EVENT
ADVANCE CLOCK
STOP AT EVENTS
1

Type: SoftwareDeveloper
Name: Sigfreido
SoftwareDevelopmentExperience: 9 years
KnowsCodingStandard: true

Figure 27: SimSE Graphical User Interface

SimSE is able to enhance the user simulation experience by its visual and interactive GUI. The player can interact with the employees through right click menus on them which shows the list of available actions (e.g. create requirements document, review requirements document, create design document, start coding and others) (Navarro and Hoek, 2004). Information is communicated through the pop up bubbles that appear over the head of the employees. Detailed information about the employees can be obtained by clicking on their image (Navarro and Hoek, 2004). Detailed information of the projects, customers, tools and artefacts can also be obtained by clicking on the relevant tabs. There is a clock at the lower right corner of the GUI which drives the simulation. The clock ticks at regular time interval and at every clock tick the simulation engine checks which actions needs to be triggered or destroyed and which underlying rules needs to be executed. The user can either step the clock forward until the next event (i.e. till someone in the game wants to say something through pop up bubbles), or step the clock forward for a specified number of clock ticks (Navarro and Hoek, 2004).

4.3.2 Why SimSE?

Whilst SimSE was originally intended for educational purposes only, it is a powerful simulation environment. It allows simulating realistic software engineering environment comprised of real world components like people, budget, large complex projects, planning, deadlines, unexpected events and others (Navarro and Hoek, 2007). It is designed to provide a platform to experience different aspects of software engineering in a practical manner.

While there are some other simulation tools like SimPack, Vensim, SESAM, CSIM and Sims available; SimSE’s visual and interactive GUI has an edge over others. It is graphical, customizable, interactive and game based software engineering simulation environment. SimSE is very user friendly and easy to use. Most importantly it fits the situation in a study like this.

As a typical game duration is around an hour, the environment provides the ability to generate a breadth of empirical evidence related to software projects. Given that a typical software development project could take weeks, months or even years to complete, the use of SimSE is a useful approach for gaining some initial evidence to support the case for future work utilising real software projects.

4.4 Evaluation parameters

Before describing the experimental evaluation, it is important to decide upon the evaluation parameters used for this research. One of the most important things in the experimental method is its evaluation parameters. The three main evaluation parameters for this research are:

1) **Time for Releases:** The SimSE XP game requires the user to implement 80 user stories in 1800 clock ticks. The game has total 4 releases (one per iteration) with each release delivering 20 user stories. The time taken for each release is one of the evaluation parameters.

2) **Final Artefacts List:** Once the game is finished, it is important to record the final state of different artefacts in the game. It represents the overall completeness and correctness of the whole code (or product). The code or product is considered to be fully complete if all the 4 iterations are fully completed. Please refer to appendix A to get detail information of each artifact.

3) **Number of Customer Complains:** In the SimSE XP game, the customer complains (through pop up bubble) if he is expecting a release and hasn't received one yet. The third evaluation parameter of this research is the total number of times the customer complains throughout the game.

These three parameters together give a thorough (if not complete) picture of the final state of the project. More importantly, these three parameters together can help to evaluate all the six EWS in focus by this research. "Time for releases" parameter helps to understand the time taken to deliver the releases and therefore allows comparing and analysing games with different time releases. "Final Artefacts List" parameter helps to understand the state of the final product. It provides minute and important information like percent erroneous, percent integrated, design percentage, unit test completeness and others of the code. "Number of Customer Complains" parameter helps to understand the number of times the customer was unhappy during the project. This also provides other signals like communication breakdown between manager and customer, weak project manager and others.

4.5 Experimental Design

As explained in the previous chapter the data has been collected from two experimental phases, namely the testing of individual EWS scenarios and controlled experimental study. To expedite the generation of data in a timely manner, the experiment utilised an existing SimSE model that defines a project being managed using the XP methodology. This game was then customised in temperately to meet the requirements for the research by creating variation games that were based on an individual EWS and then a combined game that incorporated all EWS for the controlled experimental study.

4.5.1 Individual EWS Testing Setup and Procedure

As stated earlier, individual EWS testing was performed by the researcher. For each early warning sign a model was created (and hence in total we have six models) and pseudo-experiments were run. Individual EWS testing implies testing of each model (i.e. indirectly each EWS) independently to generate insights. Figure 28 shows the general overview of the procedure for the testing of the Individual EWS models.

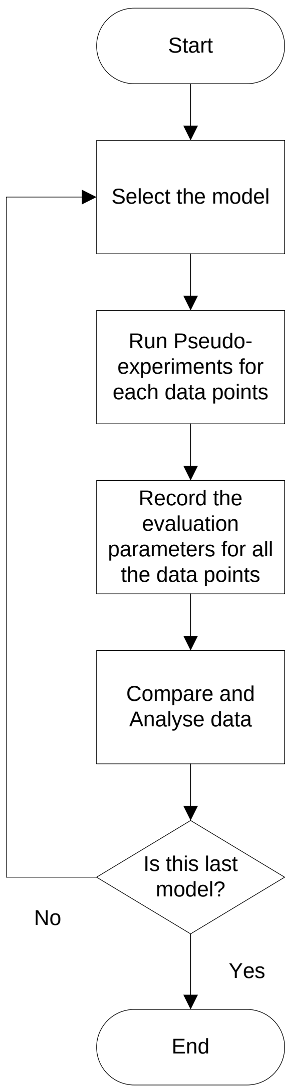

Start
Select the model
Run Pseudo-experiments for each data points
Record the evaluation parameters for all the data points
Compare and Analyse data
Is this last model?
No
Yes
End

Figure 28: Overview of “Individual EWS testing” procedure

The testing is started with the first of the six models. Once the model has been selected pseudo-experiments were run. Depending on how delayed the corrective action was taken the project result varies. Each result state represents one data point. More the data points mean more different final project results can be achieved. Further, more the data points better it is for the analysis. The number of data points is different for each model. For each data point, the evaluation parameters were recorded. Finally, once all the pseudo-experiments for each data points for the selected model were executed; the evaluation parameters for all the data points were compared and analysed and then moved to next model. For each model the loop (as shown in the flowchart) was iterated. The experimental results are presented later in this chapter.

4.5.2 Controlled Experimental Setup and Procedure

For the controlled experiment, all the six EWS were implemented in one integrated EWS model. This integrated EWS model simulates a realistic and dynamic project environment where multiple early warning signs can trigger simultaneously, forcing a player to make a choice related to their own perceived priorities. 10 volunteers took part in the experiment. The volunteers were mostly software practitioners working in IT organisations. The volunteers were asked to rate their Project Manager (PM) and Agile/XP experience from Low, Medium and High. The volunteers were having varied range of Project Manager (PM) and Agile/XP experience.

Figure 29 shows the general overview of the controlled experimental procedure.

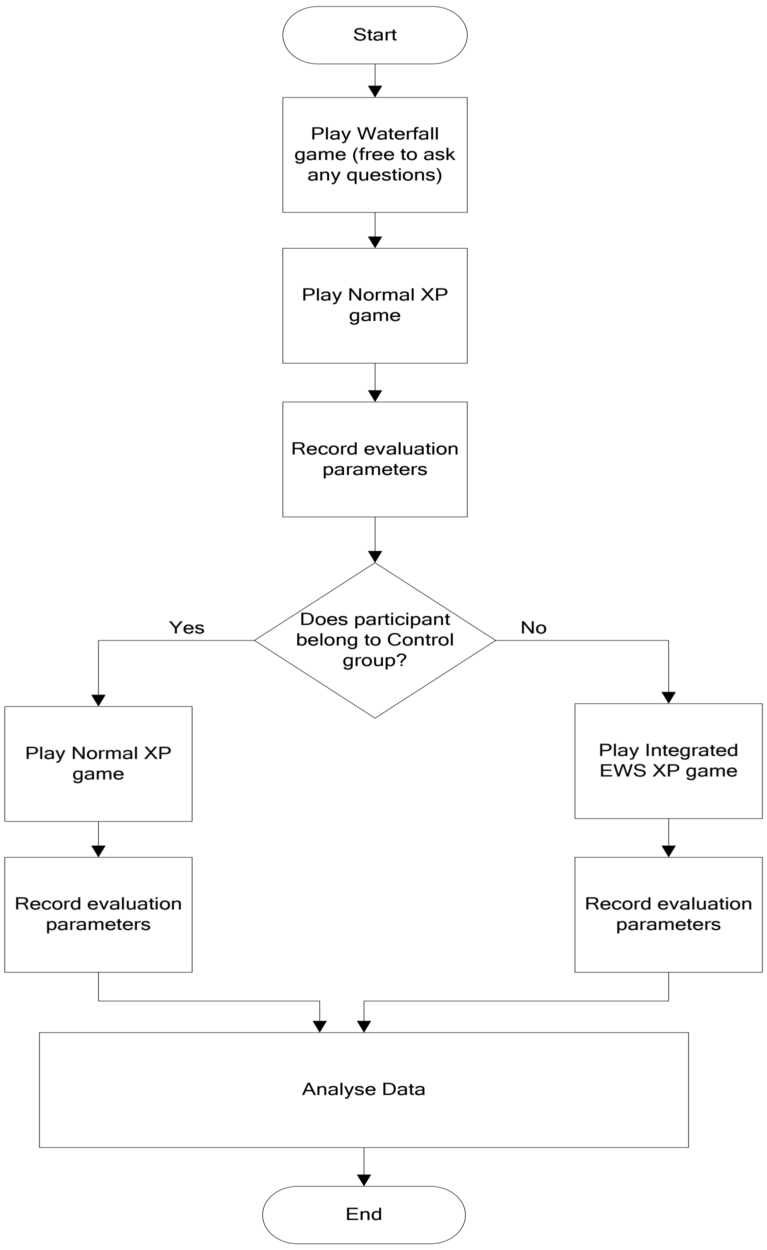

Start
Play Waterfall game (free to ask any questions)
Play Normal XP game
Record evaluation parameters
Does participant belong to Control group?
Yes
No
Play Normal XP game
Play Integrated EWS XP game
Record evaluation parameters
Record evaluation parameters
Analyse Data
End

Figure 29: Overview of Controlled Experimental procedure

The volunteers were divided equally in to two groups:

1) Control Group
2) EWS Group

Both the groups play the waterfall SimSE game once, in order to become familiar with the SimSE environment. The waterfall game is entirely different in terms of how it is played than the XP game chosen for the experiments. The game is only used to allow participants to become familiar with the mechanics of the environment without biasing the main experiments. Therefore, they were allowed to ask any questions while playing the waterfall game but not during the experiment. This removes any potential bias from the observer in terms of the game play. The control group play the normal XP SimSE game once, and their evaluation parameters were recorded. The normal XP game is the conventional XP game which won’t show any warning message(s) if the user misses any important actions. They were then given the opportunity to play the normal XP game again to improve their score. There is the potential that simply playing the game multiple times will provide a player with insight on how to improve their outcomes, therefore this is accommodated by the second game play when comparing the relative improvement of each group. The EWS group play the normal XP game once and then they play the integrated EWS XP game. The integrated EWS XP game will show warning message(s) and suggest the corrective action(s) if the user misses any important actions. So missing important actions here suggests something has gone wrong and acts as an early warning sign. This appears in the form of a warning message in the game. This overall simulates early warning phenomenon. For each user, the experiment followed the steps provided in the flowchart. Once all the ten volunteers were done with the experiments, results (evaluation parameters) of the control and EWS group were compared and analysed. The controlled experimental results are presented later in this chapter.

4.6 Experimental Results

Now having described the experimental design, this section provides the experimental results of individual EWS testing and controlled experiment phases. These were evaluated on the basis of three evaluation parameters explained earlier.

4.6.1 Individual EWS Testing Results

As said earlier, for each early warning sign a model was created and pseudo-experiments were run. For each model, depending on how delayed the corrective action(s) were taken different scenarios (or data points) were generated. These scenarios were then compared, analysed and interpreted.

4.6.1.1 EWS1 - Lack of top management support or commitment to the project

Using SimSE simulation tool an XP model was created which would provide early warning to the user (player) if there is a lack of top management support. This was identified if the management official(s) were not present in the release planning meeting of the project. Many important decisions are taken in the release planning meeting and presence of top management official(s) is a key for project success. The release planning meeting occurs once at the start of every project and the absence of management officials in the meeting is clearly a concern which may later on cause disastrous effects on the project. In the model, before the start of the release planning meeting the user (player) is asked whether the management official should be included in the meeting (Manager ‘Chang’ in this case) as shown in figure 30 below.

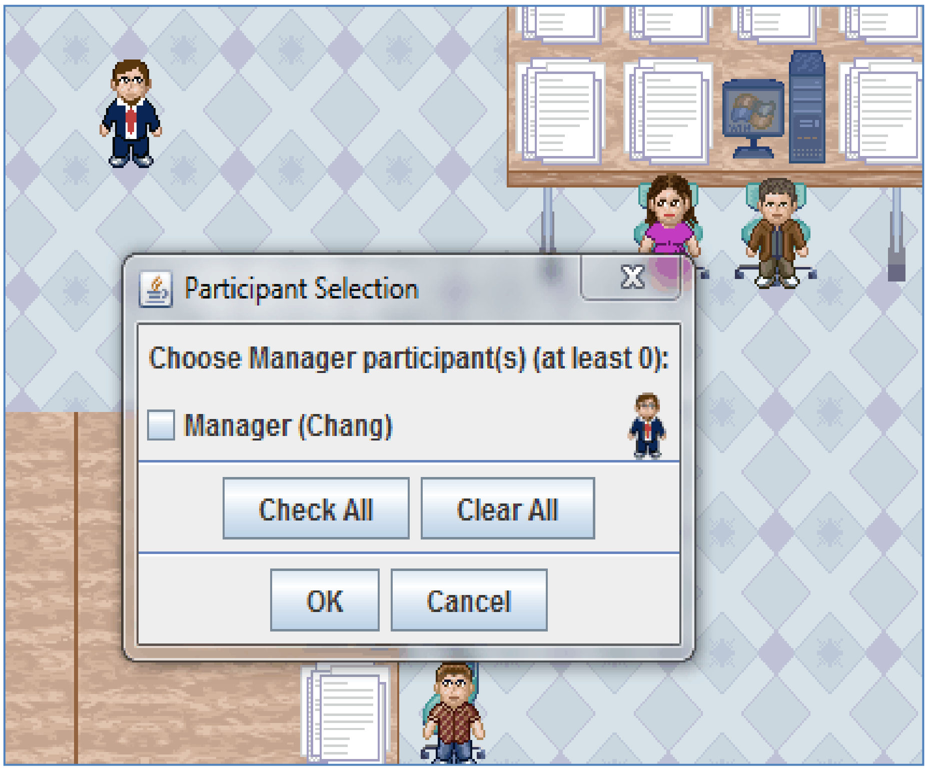
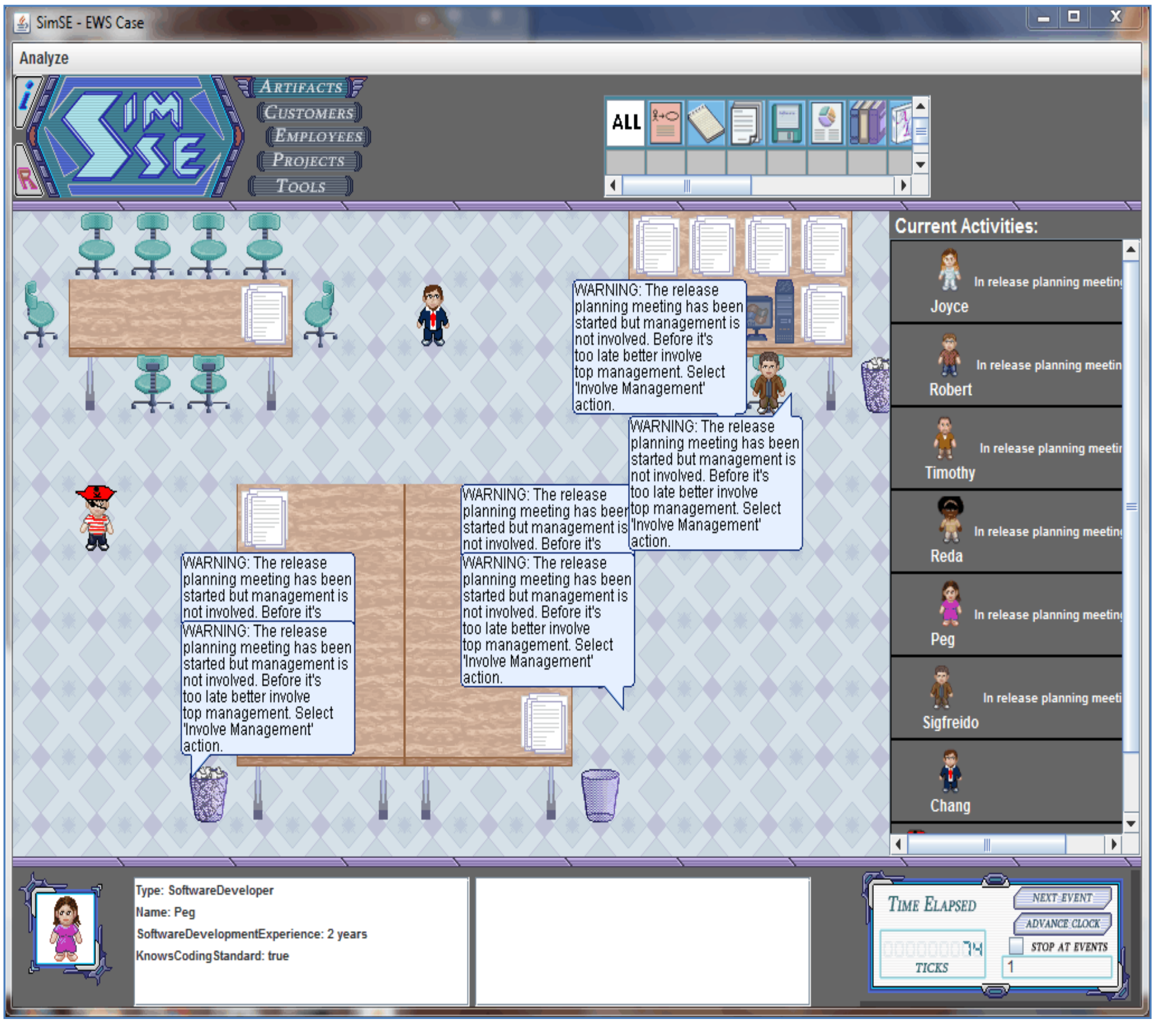


Figure 30: Selection of Manager in Release planning meeting of the project

If the user selects the manager then its fine and is assumed that there is top management support to the project. As shown in figure 31, if the user hasn't selected the manager and the meeting has already been started then an early warning message pops up immediately to intimate the user advising him to select 'Involve Management' action else project may have to face heavy losses. Once this warning message has been displayed 'Involve Management' action appears in the context menu of the project manager as shown in figure 32. Selecting 'Involve Management' implies getting management official to attend the meeting and thereby assuming the support of top management. On selecting the 'Involve Management' action a well done message appears immediately in the model as shown in figure 33.



WARNING: The release planning meeting has been started but management is not involved. Before it's too late better involve top management. Select 'Involve Management' action.

WARNING: The release planning meeting has been started but management is not involved. Before it's too late better involve top management. Select 'Involve Management' action.

WARNING: The release planning meeting has been started but management is not involved. Before it's

WARNING: The release planning meeting has been started but management is not involved. Before it's too late better involve top management. Select 'Involve Management' action.

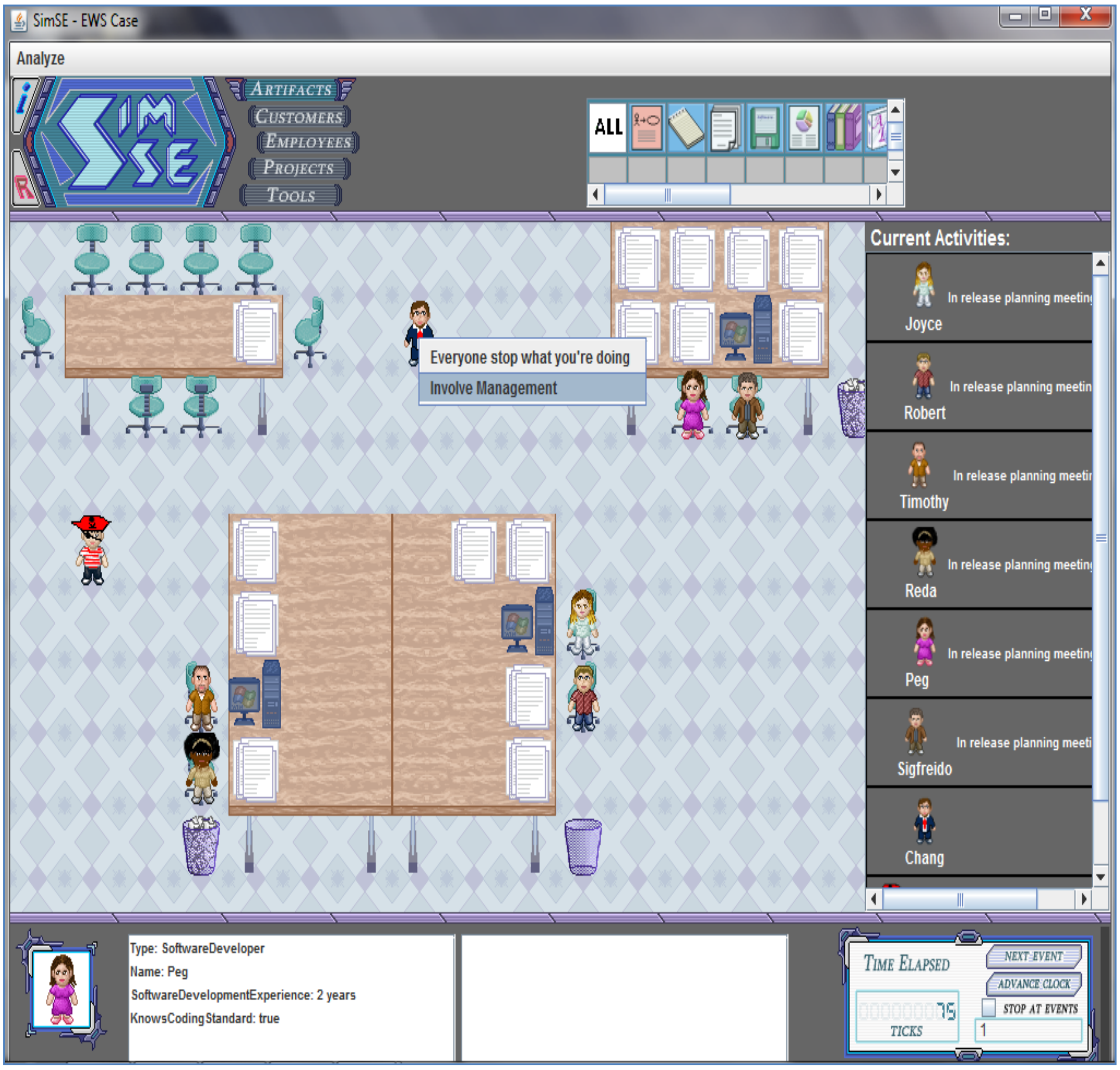
WARNING: The release planning meeting has been started but management is not involved. Before it's

WARNING: The release planning meeting has been started but management is not involved. Before it's too late better involve top management. Select 'Involve Management' action.

Joyce — In release planning meeting
Robert — In release planning meetin
Timothy — In release planning meetir
Reda — In release planning meeting
Peg — In release planning meeting
Sigfreido — In release planning meeti
Chang

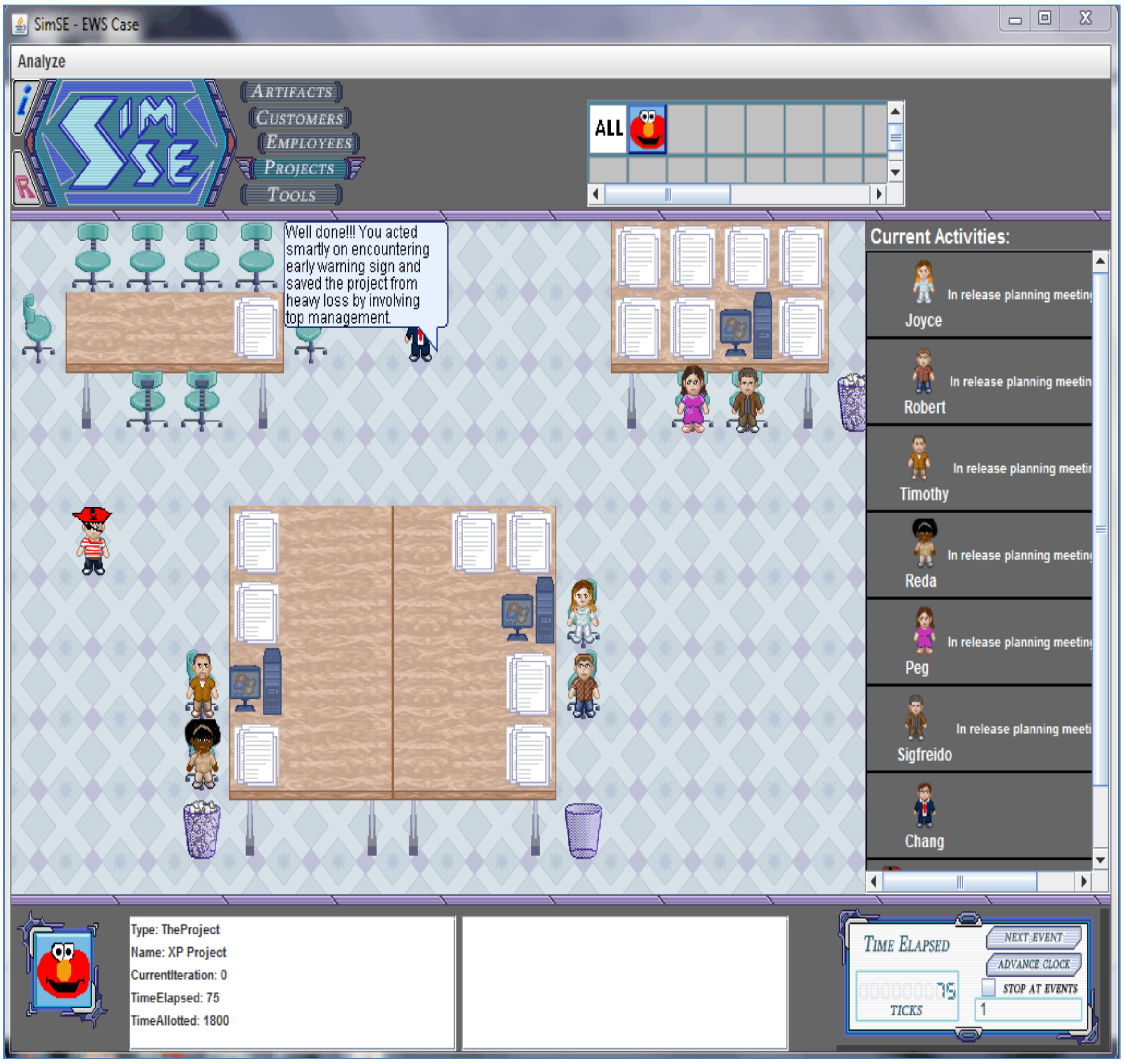
Type: SoftwareDeveloper
Name: Peg
SoftwareDevelopmentExperience: 2 years
KnowsCodingStandard: true

Figure 31: Warning message displayed for the absence of management in release planning meeting



Type: SoftwareDeveloper
Name: Peg
SoftwareDevelopmentExperience: 2 years
KnowsCodingStandard: true

Figure 32: 'Involve Management' action selected



Well done!!! You acted smartly on encountering early warning sign and saved the project from heavy loss by involving top management.

Joyce — In release planning meeting
Robert — In release planning meeting
Timothy — In release planning meeting
Reda — In release planning meeting
Peg — In release planning meeting
Sigfreido — In release planning meeting
Chang

Type: TheProject
Name: XP Project
CurrentIteration: 0
TimeElapsed: 75
TimeAllotted: 1800

Figure 33: Well done message appeared after selecting 'Involve Management' action

Even after the warning message has been displayed the user may choose not to select 'Involve Management' action and as a consequences project may face heavy losses. This XP model simulates a project which has a total of 4 iterations and implementation of 80 user stories in 1800 clock ticks. The clock tick is shown under 'Time Elapsed' section of the SimSE model (right hand bottom corner). Depending on how delayed the 'Involve Management' action has been selected the project result varies. 8 different scenarios (or data points with different results depending on the delayed selection of 'Involve Management' action) were identified for this early warning sign. The detailed description and results of these scenarios can be found in Appendix C1.

4.6.1.1.1 Analysis and Result

This section will compare and analyse the results of different scenarios and summarise the findings. Table 3 provides the summary of time for releases and customer complains in different scenarios.

	Release 1	**Release 2**	**Release 3**	**Release 4**	**Number of times Customer Complains**
Best Case	543	956	1369	1782	0
EWS Case	543	956	1369	1782	0
First Iteration	605	1018	1431	-	1
Between 1st and 2nd Iteration	667	1080	1483	-	2
Second Iteration	791	1266	1679	-	6
Between 2nd and 3rd Iteration	791	1389	-	-	9
Third Iteration	791	1514	-	-	11
Worst Case	791	1514	-	-	11

Table 3: Summary of time for releases and customer complains in different scenarios

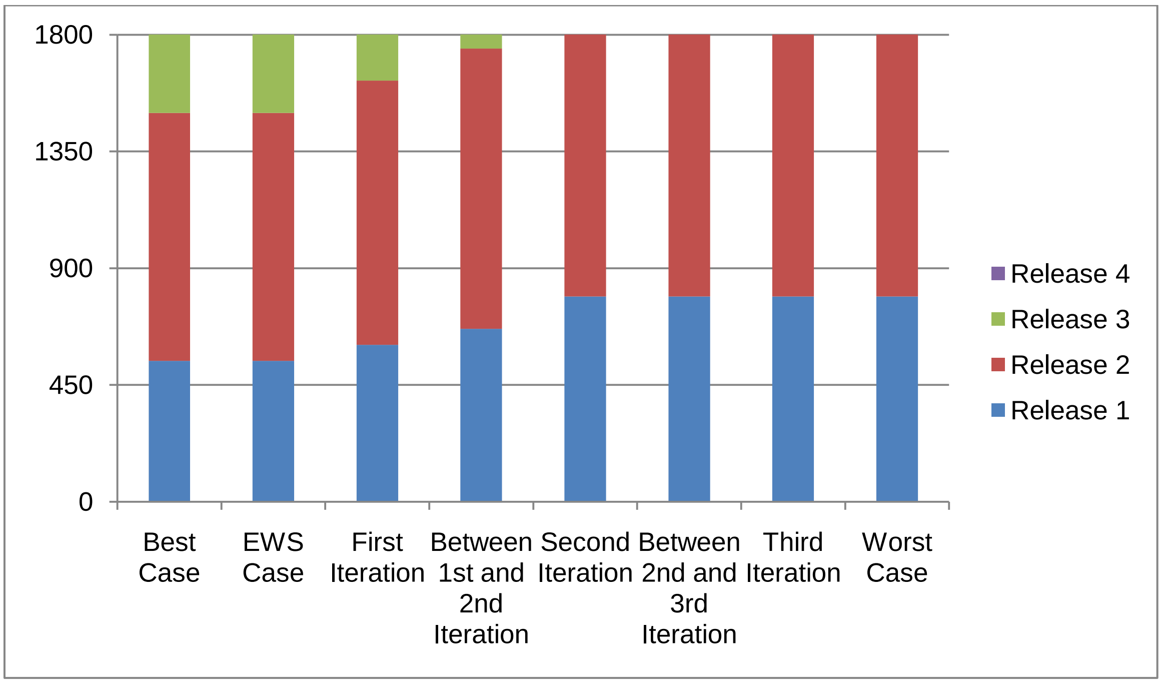


Figure 34: Graphical representation of summary of time taken by different scenarios

From ‘time for releases’ perspective it is quite evident that the simulation has performed better in cases/scenarios where corrective action was taken early. For example the performance (in terms of time for releases) of EWS case was better than first iteration case. Best case and EWS case implemented all the 80 user stories in defined time lines. First iteration, between 1st and 2nd iteration and second iteration cases all finished their first three iterations and were on their fourth iteration when the time was up. Between 2nd and 3rd iteration, third iteration and worst cases just finished their first two iterations and were on their third iteration when the time was up. The time for releases for ‘third iteration’ case and worst case were same but their final state of the artefacts was different (it is here where third iteration case performed better than worst case). An interesting point regarding the times for each iteration in the above figure is the cascade effect of a delay in one iteration causing the next iteration to be longer. It appears the longer an EWS is ignored, the greater its impact on downstream activities.

The second point of comparison is the completeness and correctness of the whole code (or product). The final state of the artefacts for each scenario represents the completeness and correctness of the product. The code or product is considered to be fully complete if all the 4 iterations are fully completed. A typical illustration of it is a best case scenario where the code is fully completed and the product is delivered to the customer (that too before the deadline). If NumUserStoriesIntegrated attribute is taken in to consideration then for best and EWS case the number of user stories integrated are 80; for first iteration case its 73; for 'between 1st & 2nd iteration', second iteration and 'between 2nd and 3rd iteration' case its 60; for third iteration and worst case its 40. It is apparent that the earlier corrective action is applied the more user stories are integrated. If PercentErroneous attribute under UserStories artifact is taken in to consideration with NumUserStoriesIntegrated attribute then it provides information about the number of correctly integrated user stories. Further, if you refer to the individual scenarios explained earlier and then compare and analyse their final state of the artefacts it is apparent that the earlier corrective action is applied the better it is for the project.

The third point of comparison is the number of times the customer complains Again the pattern is apparent that the earlier the user responds and applies recommended action the better project outcomes are attained.

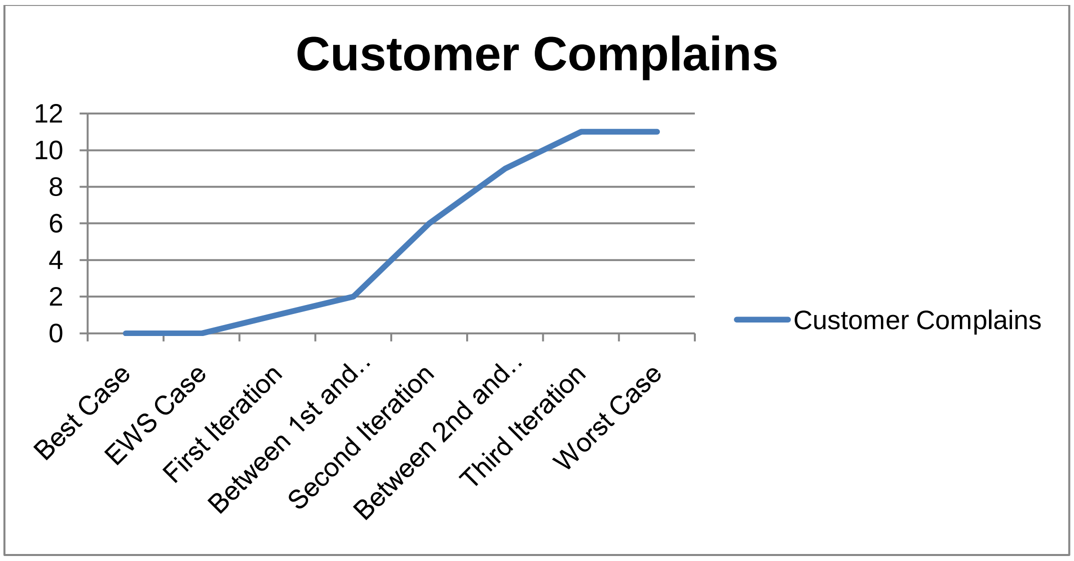
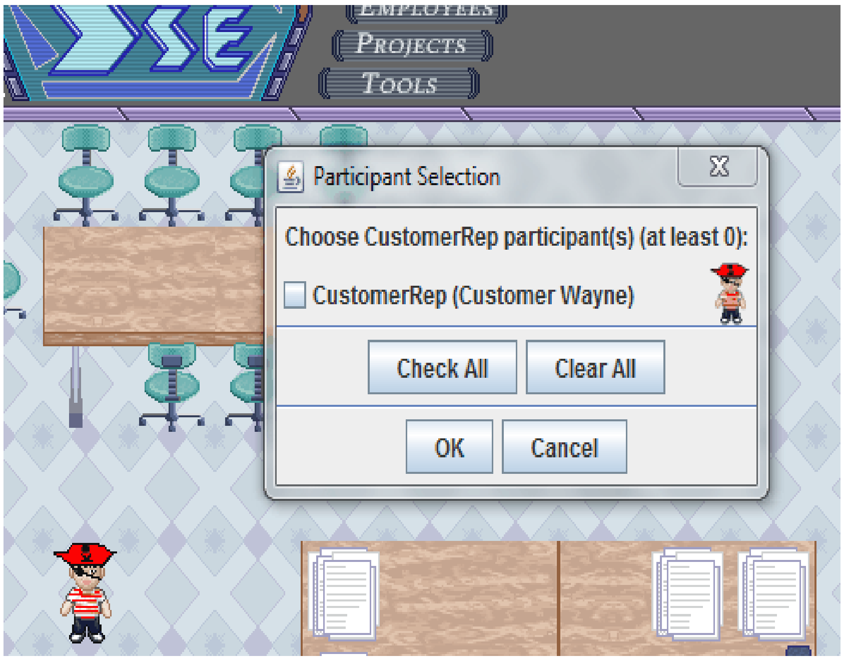


Figure 35: Graphical representation of Customer Complains in different scenarios

From all the three points of comparison it is very clear and discernible that the earlier the user responds to the warning sign (message) the better it is for the project to meet its deadline. The best case is the ideal scenario and worst case is the scenario where the user doesn’t react to the warning message at all. The worst case scenario has the highest impact to the project and the project fails to meet the expectations. It is clear that the project performance deteriorated from the best case to worst case.

4.6.1.2 EWS2 - Project stakeholders have not been interviewed for project requirements

Using SimSE simulation tool an XP model was created which would provide early warning to the user if the project stakeholders were not interviewed for the project requirements. This was identified if the customer representative(s) were not present in the release planning meeting of the project. In XP methodology important decisions are taken during the release planning meeting. The user stories (requirements) are discussed, elucidated and prioritised in the release planning meeting and the presence of customer representative(s) is of prime essence. The release planning meeting occurs once at the start of every project. As shown in figure 36, in the model before the start of the release planning meeting the user (player) is asked whether customer representative should be included in the meeting (customer ‘Wayne’ in this case).

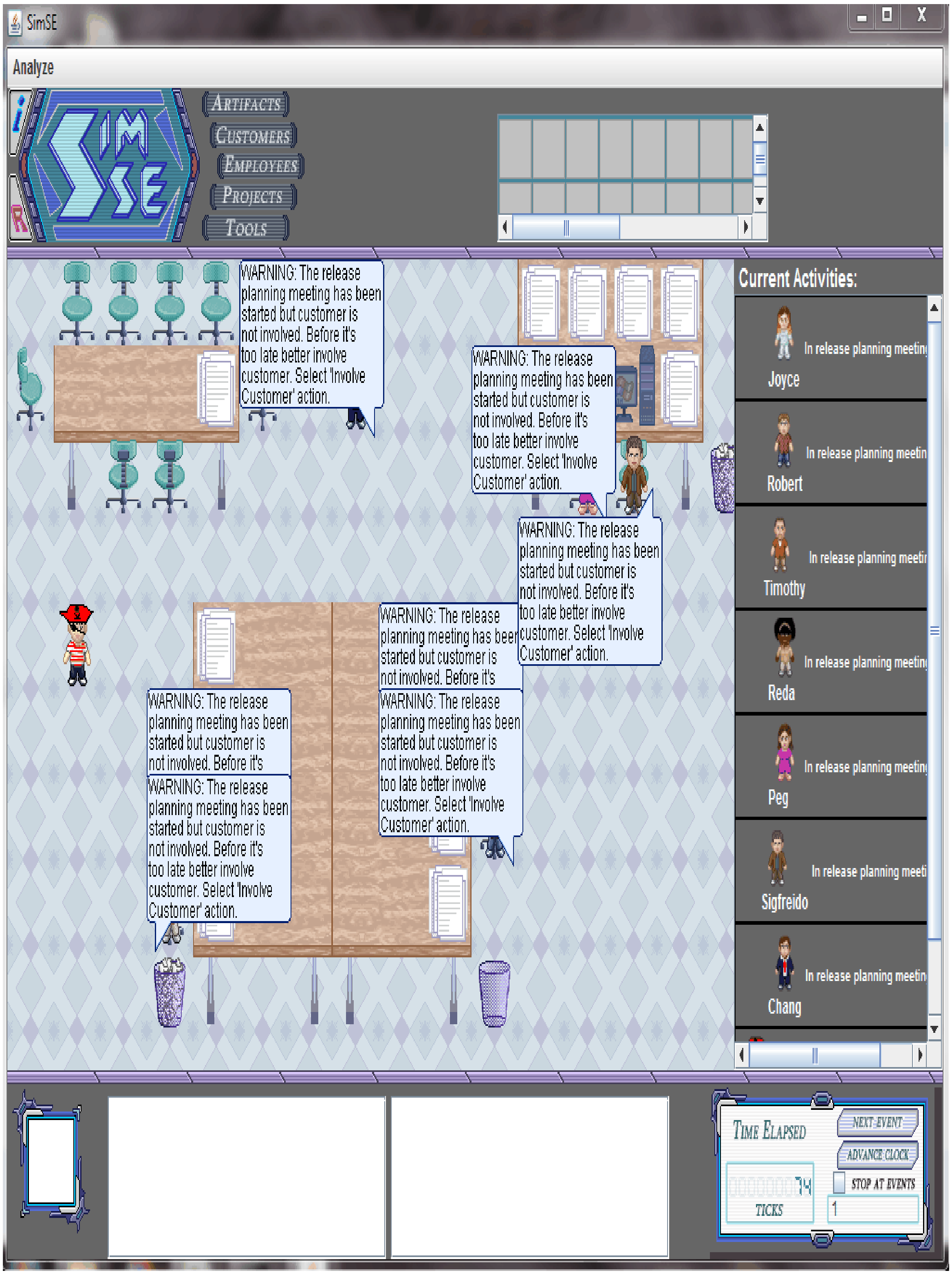


Choose CustomerRep participant(s) (at least 0):

CustomerRep (Customer Wayne)

Figure 36: Selection of customer representative in the release planning meeting of the project

If the user (player) selects the customer then its fine and is assumed that the project requirements (user stories in XP terminology) are discussed with the project stakeholder(s). As shown in figure 37, if the user (player) hasn't selected the customer and the meeting has already been started then an early warning message pops up immediately to intimate the user (player) advising him to select 'Involve Customer' action else project may have to face heavy losses. Once this warning message has been displayed 'Involve Customer' action appears in the context menu of the project manager as shown in figure 38. Selecting 'Involve Customer' implies getting customer representative to attend the meeting and thereby assuming that the project requirements have been discussed with the project stakeholders. On selecting the 'Involve Customer' action a well done message appears immediately in the model as shown in figure 39.



WARNING: The release planning meeting has been started but customer is not involved. Before it's too late better involve customer. Select 'Involve Customer' action.

WARNING: The release planning meeting has been started but customer is not involved. Before it's too late better involve customer. Select 'Involve Customer' action.

WARNING: The release planning meeting has been started but customer is not involved. Before it's too late better involve customer. Select 'Involve Customer' action.

WARNING: The release planning meeting has beer started but customer is not involved. Before it's

WARNING: The release planning meeting has been started but customer is not involved. Before it's too late better involve customer. Select 'Involve Customer' action.

WARNING: The release planning meeting has been started but customer is not involved. Before it's

WARNING: The release planning meeting has been started but customer is not involved. Before it's too late better involve customer. Select 'Involve Customer' action.

Joyce — In release planning meetin
Robert — In release planning meetin
Timothy — In release planning meetir
Reda — In release planning meetin
Peg — In release planning meetin
Sigfreido — In release planning meeti
Chang — In release planning meetin

Figure 37: Warning message displayed for the absence of customer representative in the release planning meeting

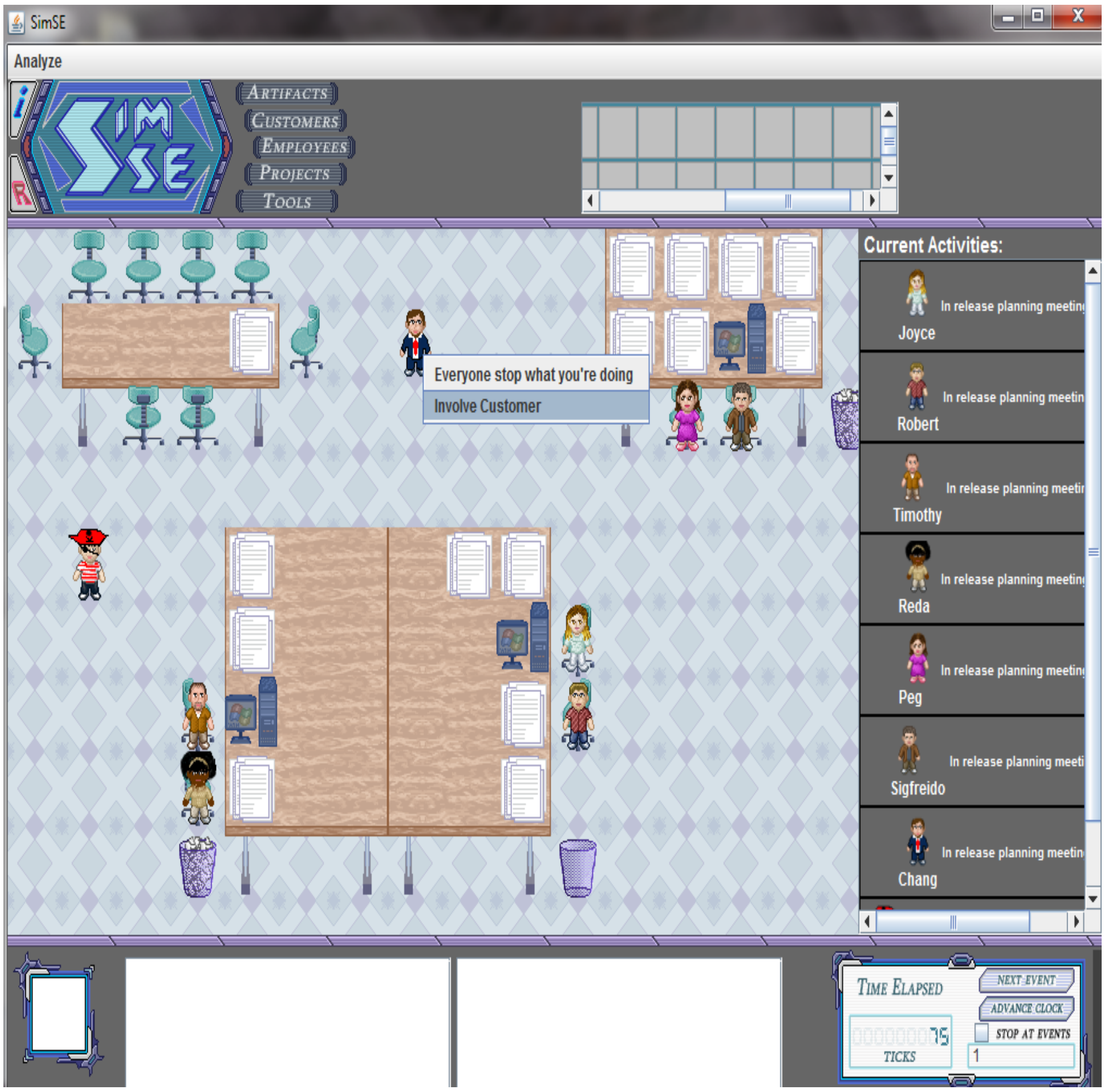
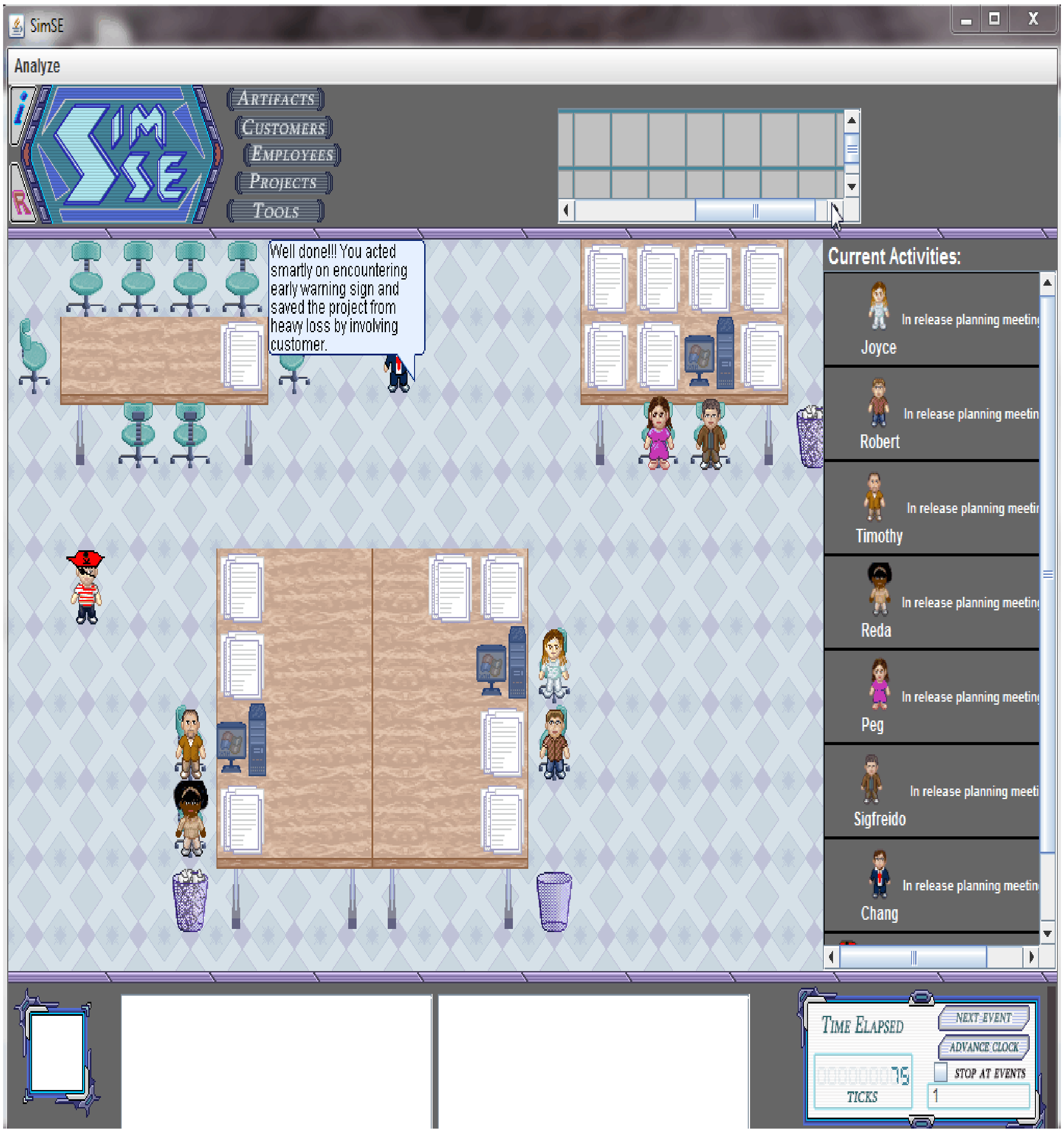

SimSE
Analyze
Artifacts
Customers
Employees
Projects
Tools
Current Activities:
Everyone stop what you're doing
Involve Customer
In release planning meeting
Joyce
In release planning meeting
Robert
In release planning meeting
Timothy
In release planning meeting
Reda
In release planning meeting
Peg
In release planning meeting
Sigfreido
In release planning meeting
Chang
Time Elapsed
Next Event
Advance Clock
Stop at Events
Ticks
1

Figure 38: 'Involve Customer' action selected



Well done!!! You acted smartly on encountering early warning sign and saved the project from heavy loss by involving customer.

In release planning meeting

Figure 39: Well done message appeared after selecting 'Involve Customer' action

Even after the warning message has been displayed the user may choose not to select the 'Involve Customer' action and as a consequences project may have to face losses. Depending on how delayed the 'Involve Customer' action has been selected the project result varies and 6 different scenarios were identified. The detailed description and results of these scenarios can be found in Appendix C2.

4.6.1.2.1 Analysis and Results

This section will compare and analyse the results of different scenarios (or data points) and summarise the findings. Table 4 below provides the summary of time for releases and customer complains in different scenarios.

	Release 1	**Release 2**	**Release 3**	**Release 4**	**Number of times Customer Complains**
Best Case	543	956	1369	1782	0
EWS Case	543	956	1369	1782	0
Second Iteration	573	986	1399	-	1
Third Iteration	573	1016	1429	-	2
Fourth Iteration	573	1016	1459	-	3
Worst Case	573	1016	1459	-	3

Table 4: Summary of time for releases and customer complains in different scenarios

In terms of time for releases, best and EWS case performed better than second iteration case; and second iteration case performed better than third iteration case. Further, third iteration case performed better than fourth iteration and worst case.

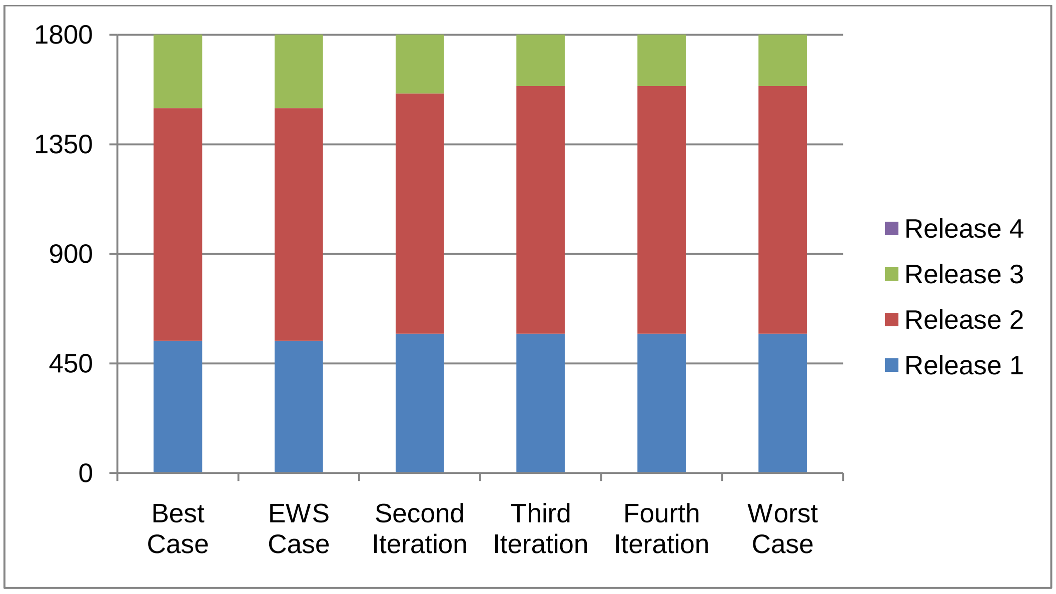


Figure 40: Graphical representation of summary of time taken by different scenarios

The second point of comparison is the completeness and correctness of the whole code (or product). The final state of the artefacts for each scenario represents the completeness and correctness of the product. The code or product is considered to be fully complete if all the 4 iterations are fully completed. A typical illustration of it is a best case scenario where the code is fully completed and the product is delivered to the customer (that too before the deadline). If NumUserStoriesIntegrated attribute is taken in to consideration then for best, EWS and second iteration case the number of user stories integrated are 80; for third iteration case its 75; for fourth iteration and worst case its 60. It is apparent that the earlier corrective action is applied the more user stories are integrated. If PercentErroneous attribute under UserStories artifact is taken in to consideration with NumUserStoriesIntegrated attribute then it provides information about the number of correctly integrated user stories. Further, if you refer to the individual scenarios explained earlier and then compare and analyse their final state of the artefacts it is apparent that the earlier corrective action is applied the better it is for the project.

The third point of comparison is the number of times the customer complains. Again the pattern is apparent that the earlier the user responds and applies recommended action the better project outcomes are attained.

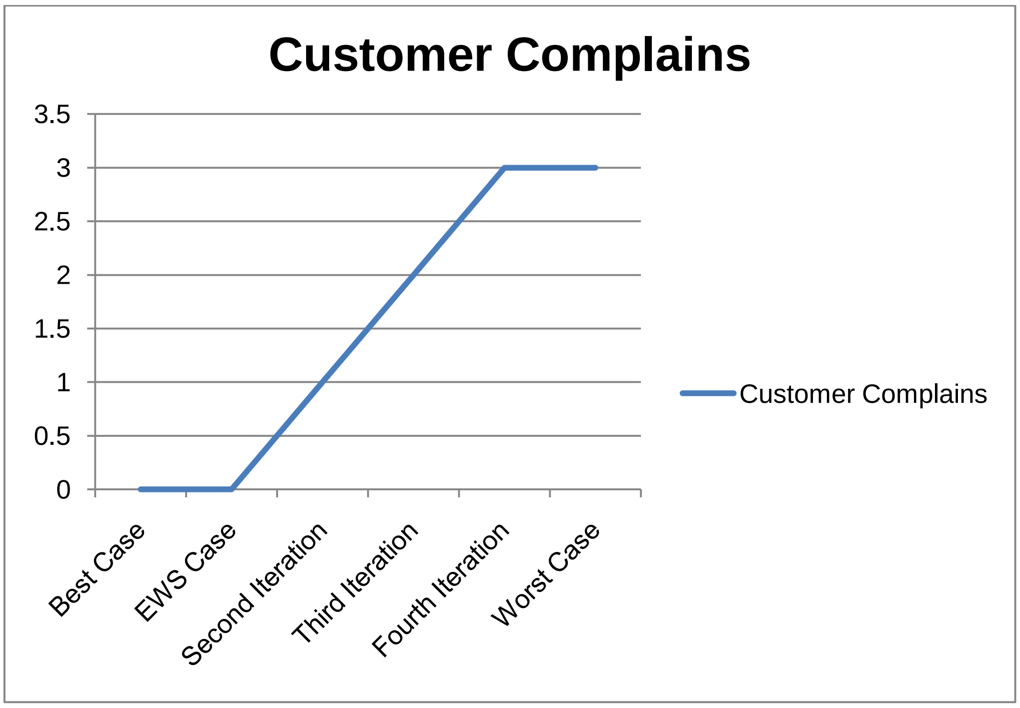
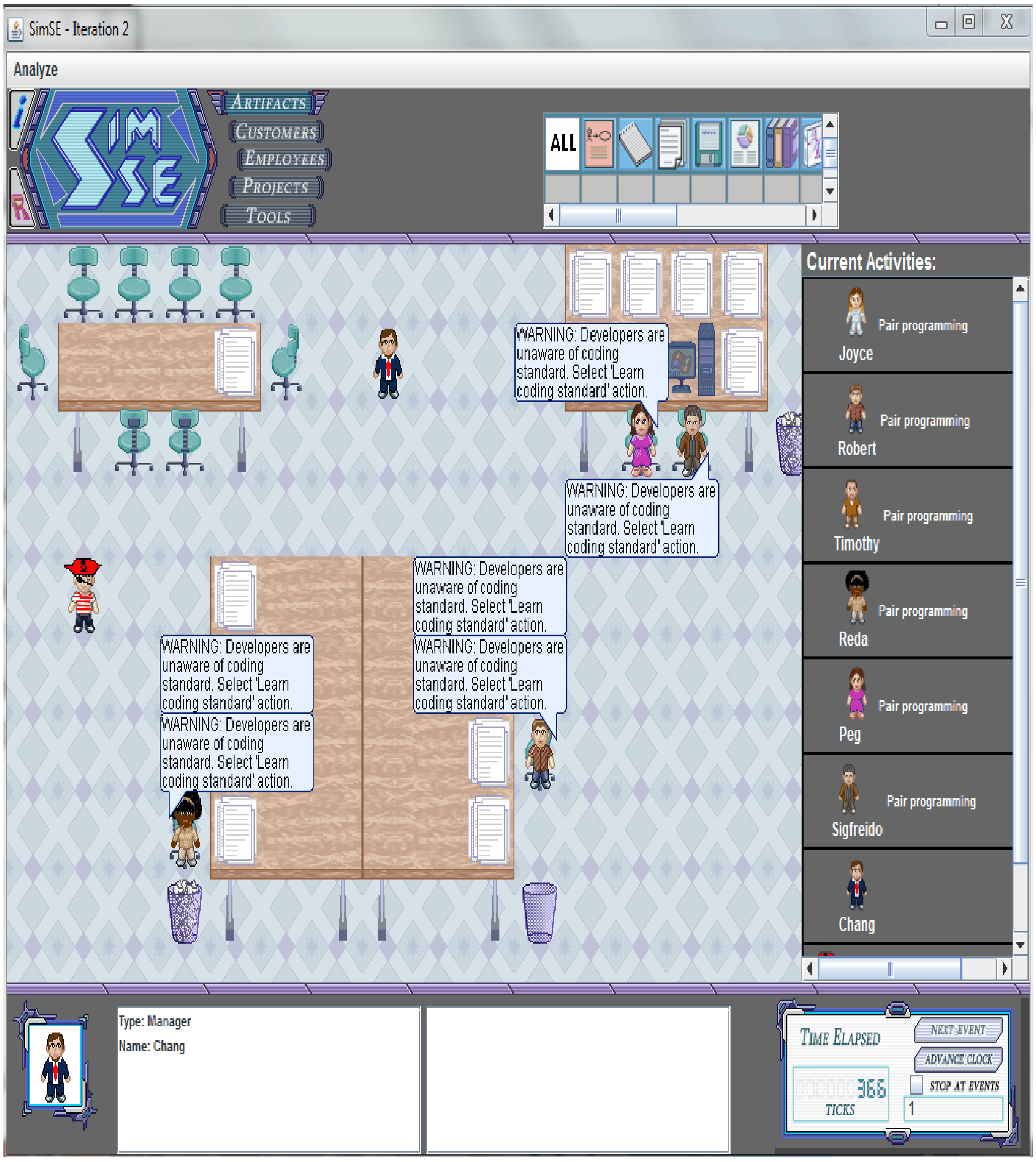


Figure 41: Graphical representation of Customer Complains in different scenarios

From all the three points of comparison it is very clear and discernible that the earlier the user responds to the warning sign (message) the better it is for the project to meet its deadline. The best case is the ideal scenario and worst case is the scenario where the user doesn't react to the warning message at all. The worst case scenario has the highest impact to the project and the project fails to meet the expectations. It is clear that the project performance deteriorated from the best case to worst case.

4.6.1.3 EWS3 - Project team members do not have required knowledge/skills

In order to exercise this condition a SimSE XP model was created which would provide early warning to the user if the project team members do not have required knowledge/skills. An early warning would trigger if the developers (project team members) were unaware of the coding standard (required knowledge/skills). This was identified if the "KnowsCodingStandard" attribute of the developers was false. At the start of the game (i.e. the start of running through the SimSE XP model) the developers are unaware of the coding standard. It is expected that the developers would up-skill themselves with the coding standard before they start off with their hands on coding for the project. As shown in figure 42, a warning message appears repetitiously in the game (model) so that the user is made aware that the developers need to be up-skilled.



WARNING: Developers are unaware of coding standard. Select 'Learn coding standard' action.

WARNING: Developers are unaware of coding standard. Select 'Learn coding standard' action.

WARNING: Developers are unaware of coding standard. Select 'Learn coding standard' action.

WARNING: Developers are unaware of coding standard. Select 'Learn coding standard' action.

WARNING: Developers are unaware of coding standard. Select 'Learn coding standard' action.

WARNING: Developers are unaware of coding standard. Select 'Learn coding standard' action.

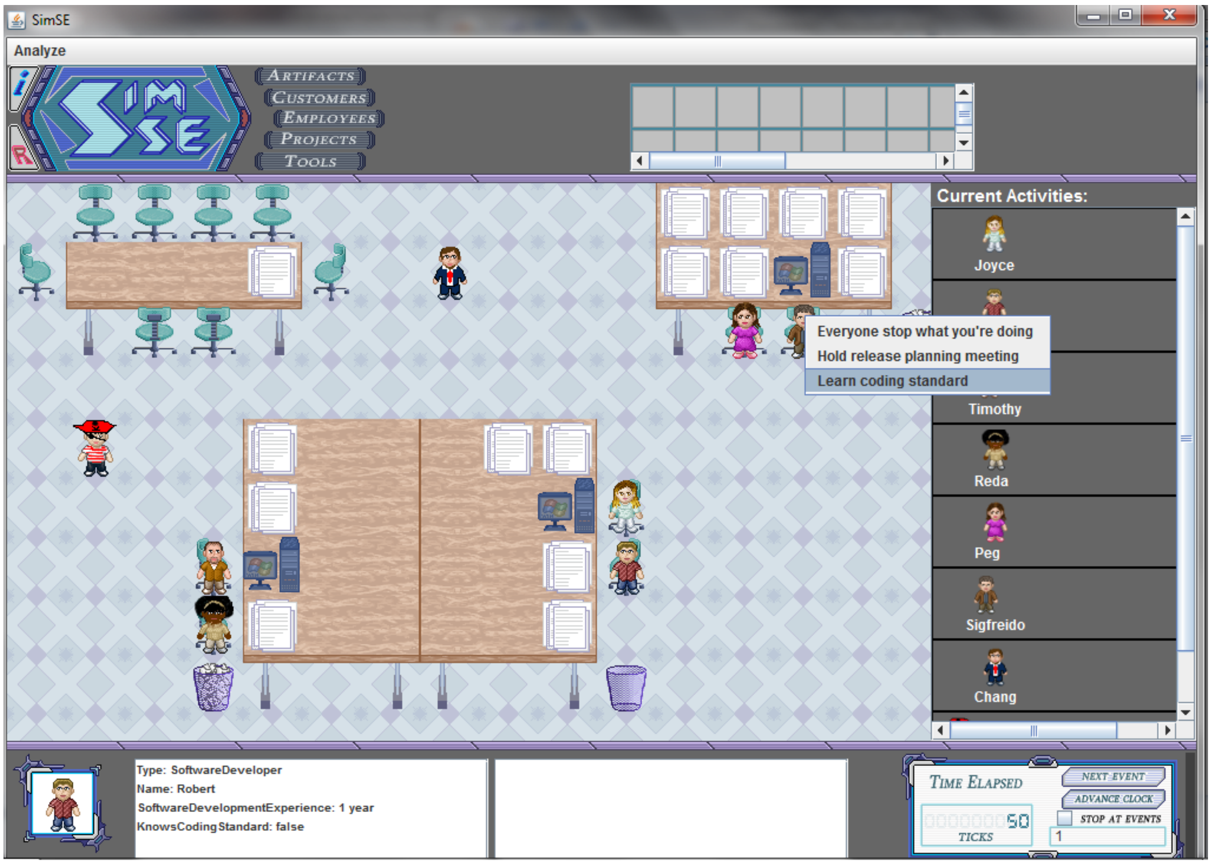
Pair programming Joyce
Pair programming Robert
Pair programming Timothy
Pair programming Reda
Pair programming Peg
Pair programming Sigfreido
Chang

Type: Manager
Name: Chang

366
1

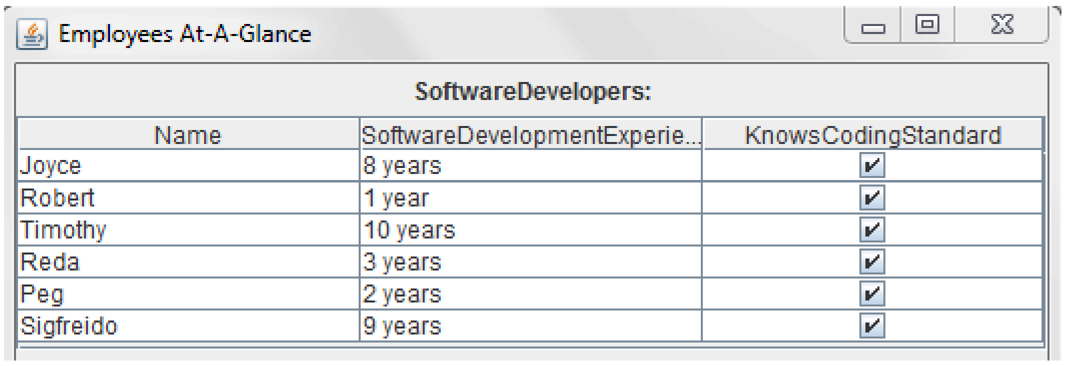
Figure 42: Warning message appears when developers are unaware of coding standard

The warning message advises the user to select ‘Learn coding standard’ action. If the user selects ‘Learn coding standard’ action from the context menu (as shown in figure 43) then its fine and is assumed that the developers have up-skilled themselves and have acquired the required knowledge/skills to perform the coding task. On the completion of ‘Learn coding standard’ action, the ‘KnowsCodingStandard’ attribute of the developers changes to true (as shown in figure 44).



Type: SoftwareDeveloper
Name: Robert
SoftwareDevelopmentExperience: 1 year
KnowsCodingStandard: false

Figure 43: ‘Learn coding standard’ action in the context menu


Employees At-A-Glance

SoftwareDevelopers:

Name	SoftwareDevelopmentExperie...	KnowsCodingStandard
Joyce	8 years	✓
Robert	1 year	✓
Timothy	10 years	✓
Reda	3 years	✓
Peg	2 years	✓
Sigfreido	9 years	✓

Figure 44: Software developer's artefacts once the developers are aware of coding standard

Even after the warning message has been displayed the user may choose not to select 'Learn coding standard' action and start coding for the project and as a consequences project may have to face losses. Depending on how delayed the 'Learn coding standard' action has been selected the project result varies and 5 different scenarios were identified. The detailed description and results of these scenarios can be found in Appendix C3.

4.6.1.3.1 Analysis and Results

Table 5 below provides the summary of time for releases and customer complains in different scenarios.

	Release 1	Release 2	Release 3	Release 4	Number of times Customer Complains
Best Case	543	956	1369	1782	0
EWS Case	543	956	1369	1782	0
Second Iteration	655	1088	1501	-	3
Third Iteration	655	1200	1633	-	6
Worst Case	655	1200	1745	-	9

Table 5: Summary of time for releases and customer complains in different scenarios

In terms of time for releases, best and EWS case performed better than second iteration case; and second iteration case performed better than third iteration case. Further, third iteration case performed better than worst case.

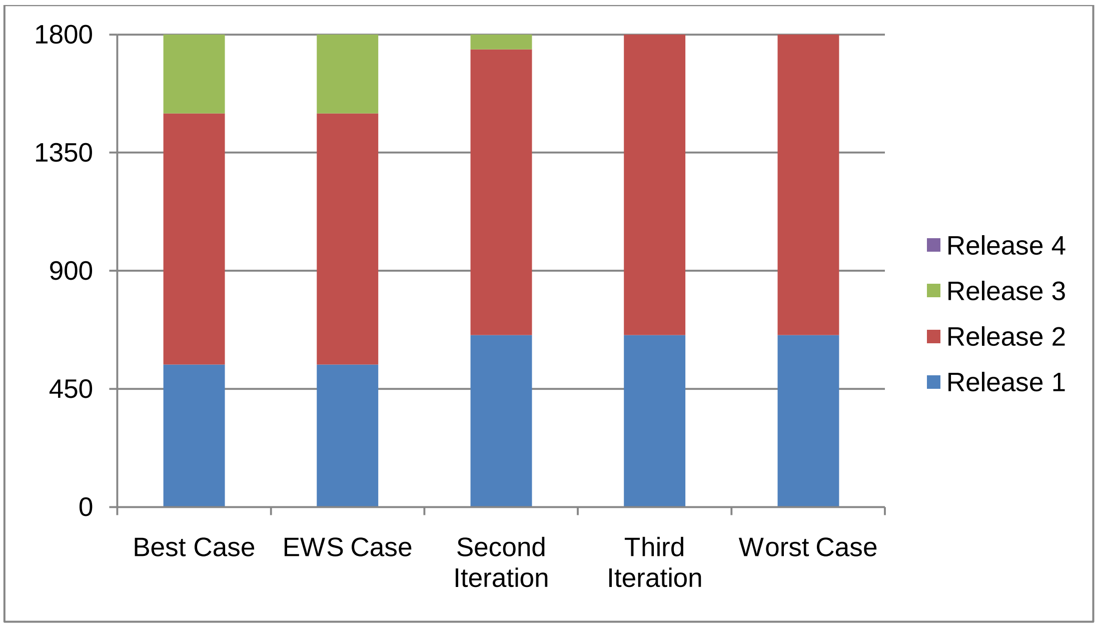


Figure 45: Graphical representation of summary of time taken by different scenarios

The second point of comparison is the completeness and correctness of the whole code (or product). If you refer to the individual scenarios explained earlier and then compare and analyse their final state of the artefacts it is apparent that the earlier corrective action is applied the better it is for the project.

The third point of comparison is the number of times the customer complains. Again the pattern is apparent that the earlier the user responds and applies recommended action the better project outcomes are attained.

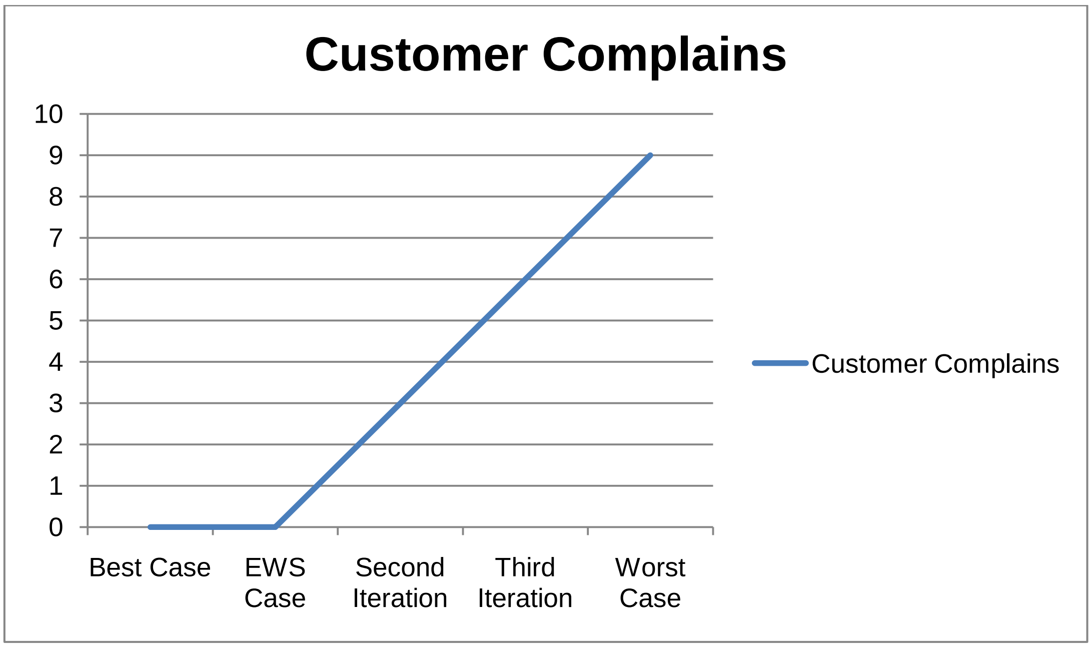
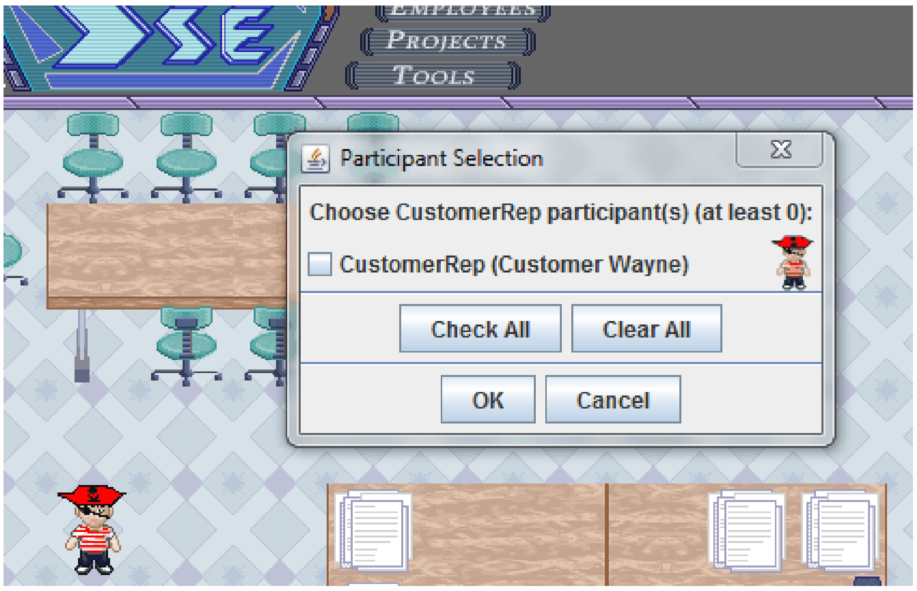


Figure 46: Graphical representation of Customer Complains in different scenarios

From all the three points of comparison it is very clear and discernible that the earlier the user responds to the warning sign (message) the better it is for the project to meet its deadline. The best case is the ideal scenario and the worst case has the highest impact to the project and the project fails to meet the expectations. It is clear that the project performance deteriorated from the best case to worst case.

4.6.1.4 EWS4 - Key project stakeholders do not participate in major review meetings

In order to exercise this condition a SimSE XP model was created which would provide early warning to the user if key project stakeholders do not participate in major review meetings. In XP methodology, it is important that the customer is involved in the creation of the acceptance tests as they are responsible for verifying the correctness of the acceptance tests. In this case, if the customer (key project stakeholder) is not involved in the acceptance tests meeting (major review meeting) then a warning message would trigger. This was identified if the customer representative was not involved in the acceptance tests meeting. In the model, before the creation of the acceptance tests the user is asked whether the customer representative should be involved in the meeting (Customer 'Wayne' in this case) (as shown in figure 47).



Choose CustomerRep participant(s) (at least 0):

☐ CustomerRep (Customer Wayne)

Figure 47: Selection of customer representative in the creation of acceptance tests

If the user selects the customer then its fine and is assumed that key project stakeholder (customer representative) is involved in major review meeting (creation of acceptance tests). If the user hasn't selected the customer and the meeting has already been started then an early warning message pops up immediately to intimate the user advising him to select 'Involve Customer' action else project might have to face heavy losses (as shown in figure 48).

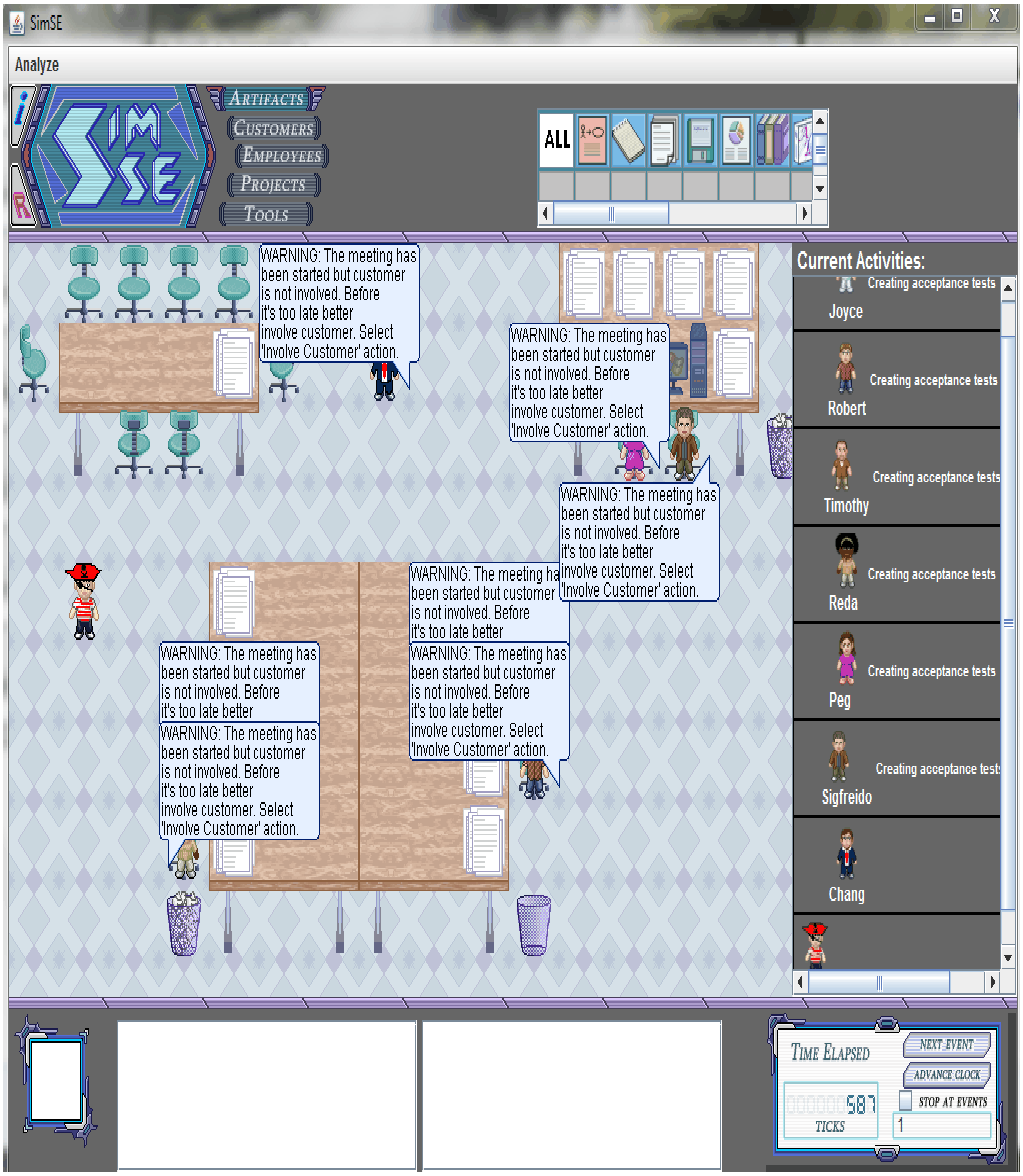

SimSE
Analyze
Artifacts
Customers
Employees
Projects
Tools
ALL
WARNING: The meeting has been started but customer is not involved. Before it's too late better involve customer. Select 'Involve Customer' action.
WARNING: The meeting has been started but customer is not involved. Before it's too late better involve customer. Select 'Involve Customer' action.
WARNING: The meeting has been started but customer is not involved. Before it's too late better involve customer. Select 'Involve Customer' action.
WARNING: The meeting ha been started but customer is not involved. Before it's too late better
WARNING: The meeting has been started but customer is not involved. Before it's too late better involve customer. Select 'Involve Customer' action.
WARNING: The meeting has been started but customer is not involved. Before it's too late better
WARNING: The meeting has been started but customer is not involved. Before it's too late better involve customer. Select 'Involve Customer' action.
Current Activities:
Creating acceptance tests
Joyce
Creating acceptance tests
Robert
Creating acceptance tests
Timothy
Creating acceptance tests
Reda
Creating acceptance tests
Peg
Creating acceptance tests
Sigfreido
Chang
Time Elapsed
587
TICKS
NEXT EVENT
ADVANCE CLOCK
STOP AT EVENTS
1

Figure 48: Warning message triggered for not involving customer in the creation of acceptance tests

Once this warning message has been displayed 'Involve Customer' action appears in the context menu of the project manager (as shown in figure 49).

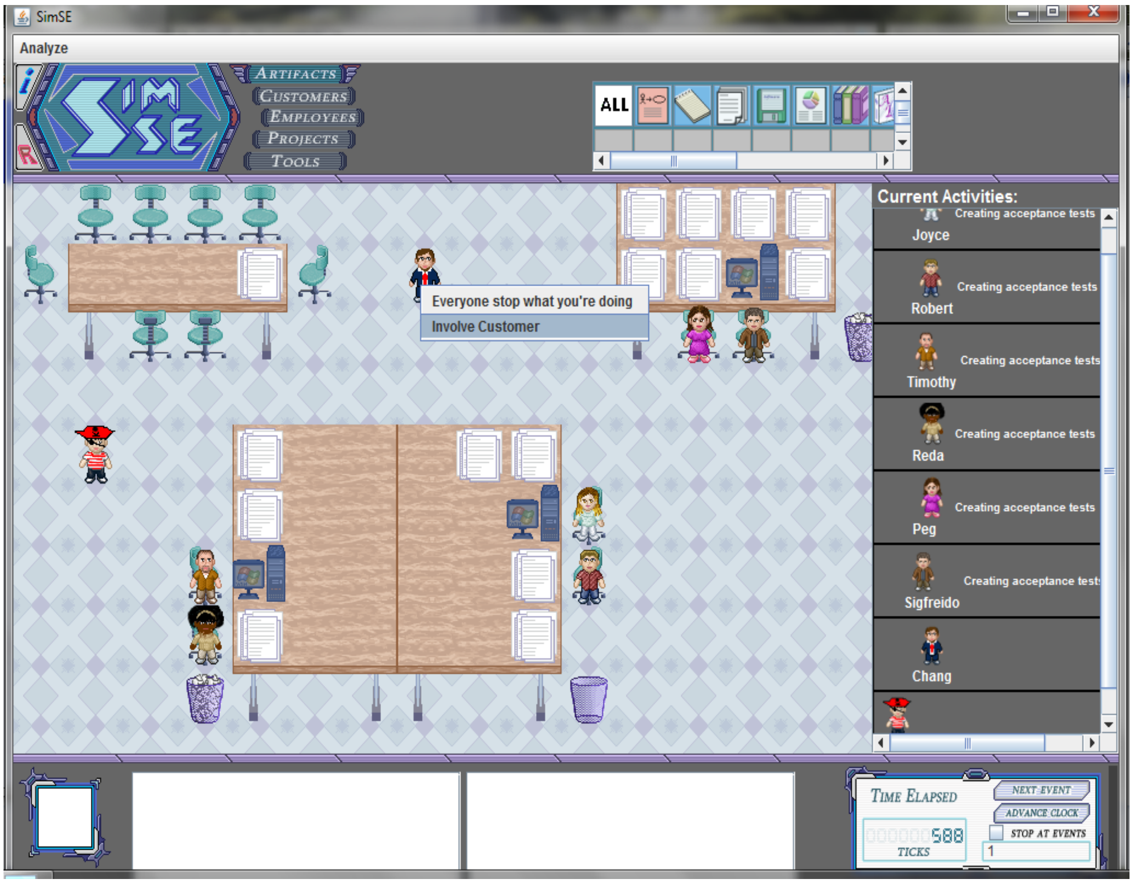
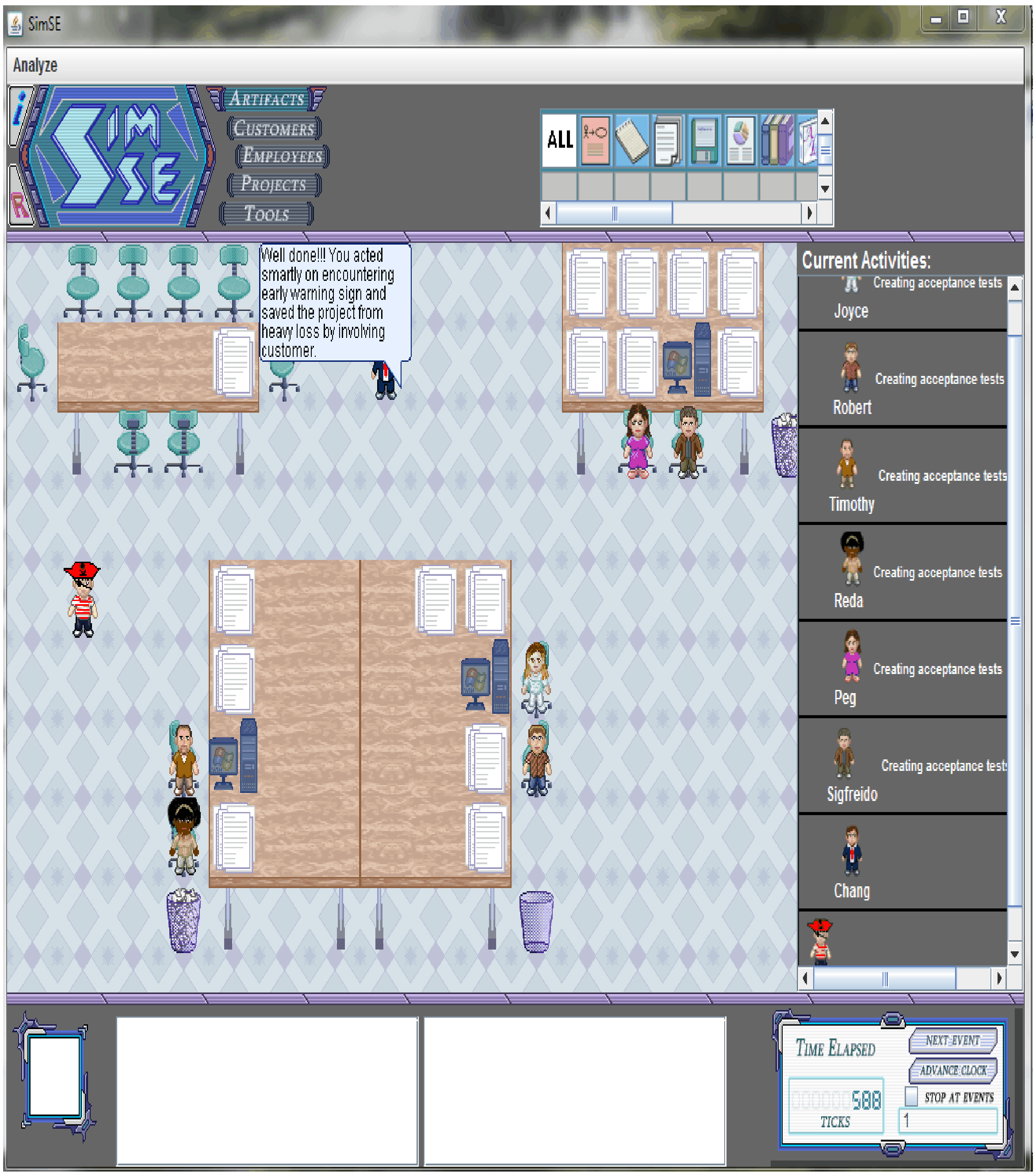


Figure 49: ‘Involve Customer ‘action appears in the context menu

Selecting ‘Involve Customer’ implies getting customer representative to attend the meeting (i.e. involving him in the creation of the acceptance tests) and thereby assuming that the key project stakeholder(s) is participating in major review meeting. As shown in figure 50, on selecting the ‘Involve Customer’ action a well done message appears immediately in the model. Even after the warning message has been displayed the user may choose not to select ‘Involve Customer’ action and under that situation if the user runs (executes) the acceptance tests then half of the tests fail because customer was not involved in the meeting (as shown in figure 51).



Well done!!! You acted smartly on encountering early warning sign and saved the project from heavy loss by involving customer.

Figure 50: Well done message appeared after selecting 'Involve Customer' action

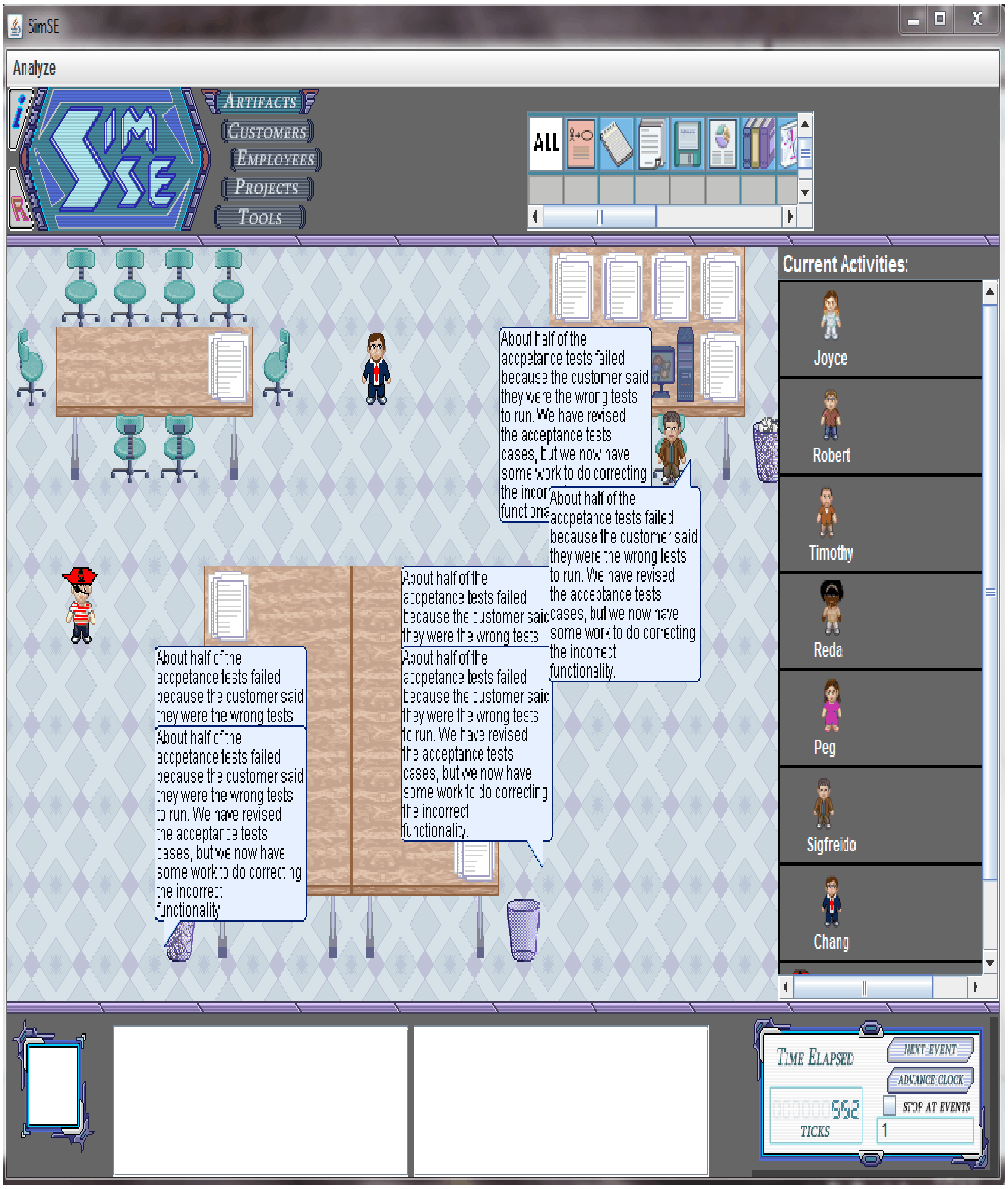

SimSE
Analyze
Artifacts
Customers
Employees
Projects
Tools
ALL
Current Activities:
Joyce
Robert
Timothy
Reda
Peg
Sigfreido
Chang
About half of the accpetance tests failed because the customer said they were the wrong tests to run. We have revised the acceptance tests cases, but we now have some work to do correcting the incorrect functionality.
Time Elapsed
552
Ticks
Next Event
Advance Clock
Stop at events
1

Figure 51: Message appeared on executing acceptance tests if the customer was not involved in its creation

Depending on how delayed the ‘Involve Customer’ action has been selected the project result varies and 4 different scenarios were identified. The detailed description and results of these scenarios can be found in Appendix C4.

4.6.1.4.1 Analysis and Results

Table 6 below provides the summary of time for releases and customer complains in different scenarios.

	Release 1	Release 2	Release 3	Release 4	Number of times Customer Complains
Best Case	543	956	1369	1782	0
EWS Case	552	974	1396	-	5
Second Iteration	800	1222	1644	-	5
Worst Case	800	1470	-	-	10

Table 6: Summary of time for releases and customer complains in different scenarios

In terms of time for releases, best case performed better than EWS case; EWS case performed better than second iteration case. Further, second iteration case performed better than worst case.

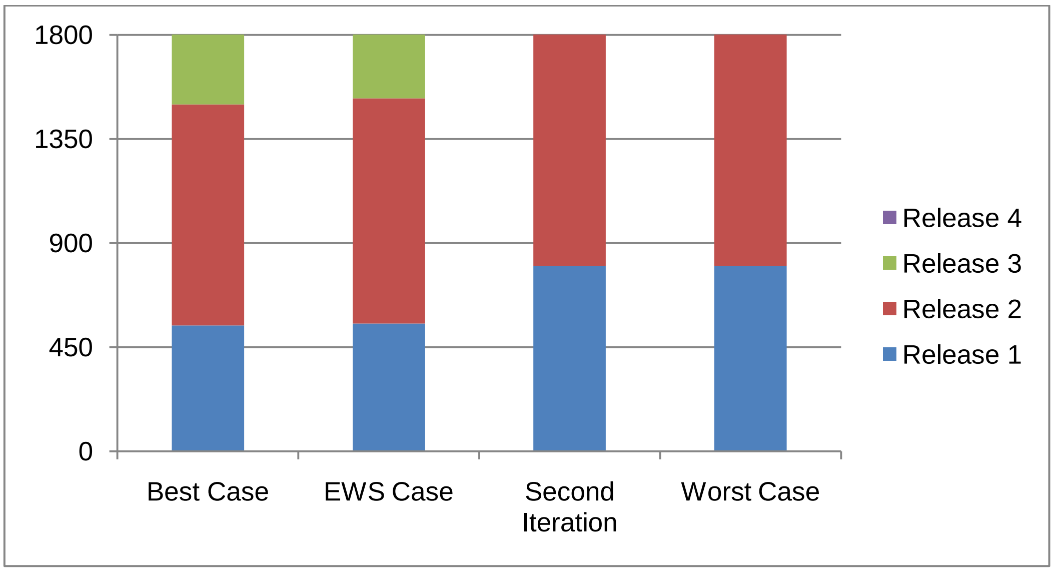


Figure 52: Graphical representation of summary of time taken by different scenarios

The second point of comparison is the completeness and correctness of the whole code (or product). If NumUserStoriesIntegrated attribute is taken in to consideration then for best and EWS case the number of user stories integrated are 80; for second iteration case its 60; and for worst case its 40. It is apparent that the earlier corrective action is applied the more user stories are integrated. Further, if you refer to the individual scenarios explained earlier and then compare and analyse their final state of the artefacts it is apparent that the earlier corrective action is applied the better it is for the project.

The third point of comparison is the number of times the customer complains.

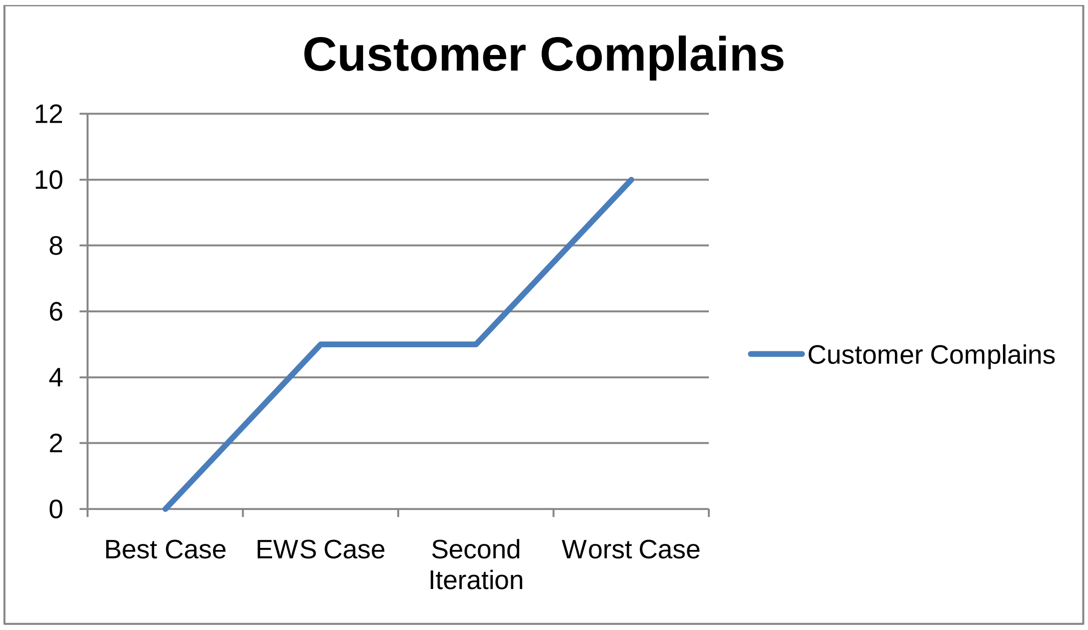
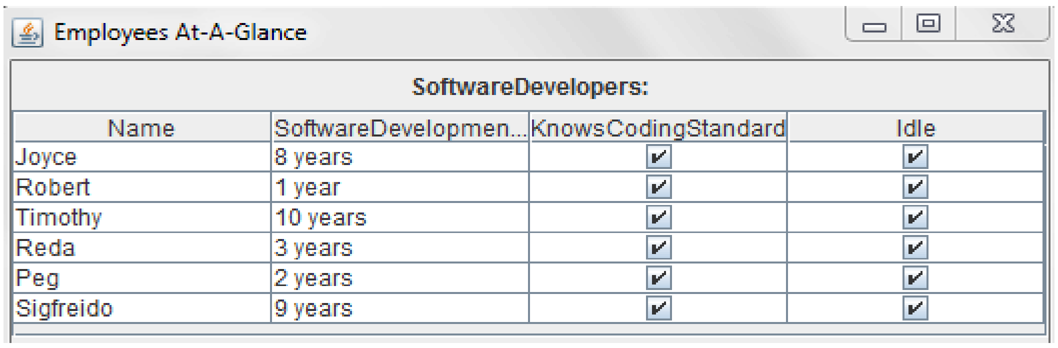


Figure 53: Graphical representation of Customer complains in different scenarios

From all the three points of comparison it is very clear and discernible that the earlier the user responds to the warning sign (message) the better it is for the project to meet its deadline. It is clear that the project performance deteriorated from the best case to worst case.

4.6.1.5 EWS5 - Project manager(s) cannot effectively lead the team

In order to demonstrate this condition a SimSE XP model was created which would provide early warning to the user if the project manager cannot effectively lead a team. In this case, if the team is idle and not assigned any work then we are assuming that project manager cannot effectively lead the team. Not assigning work to team member is definitely a sign of weak/inexperienced project manager. This was identified if the employee in the game was idle (i.e. not assigned any work) and hence its “idle” attribute is set (as shown in figure 54 below).

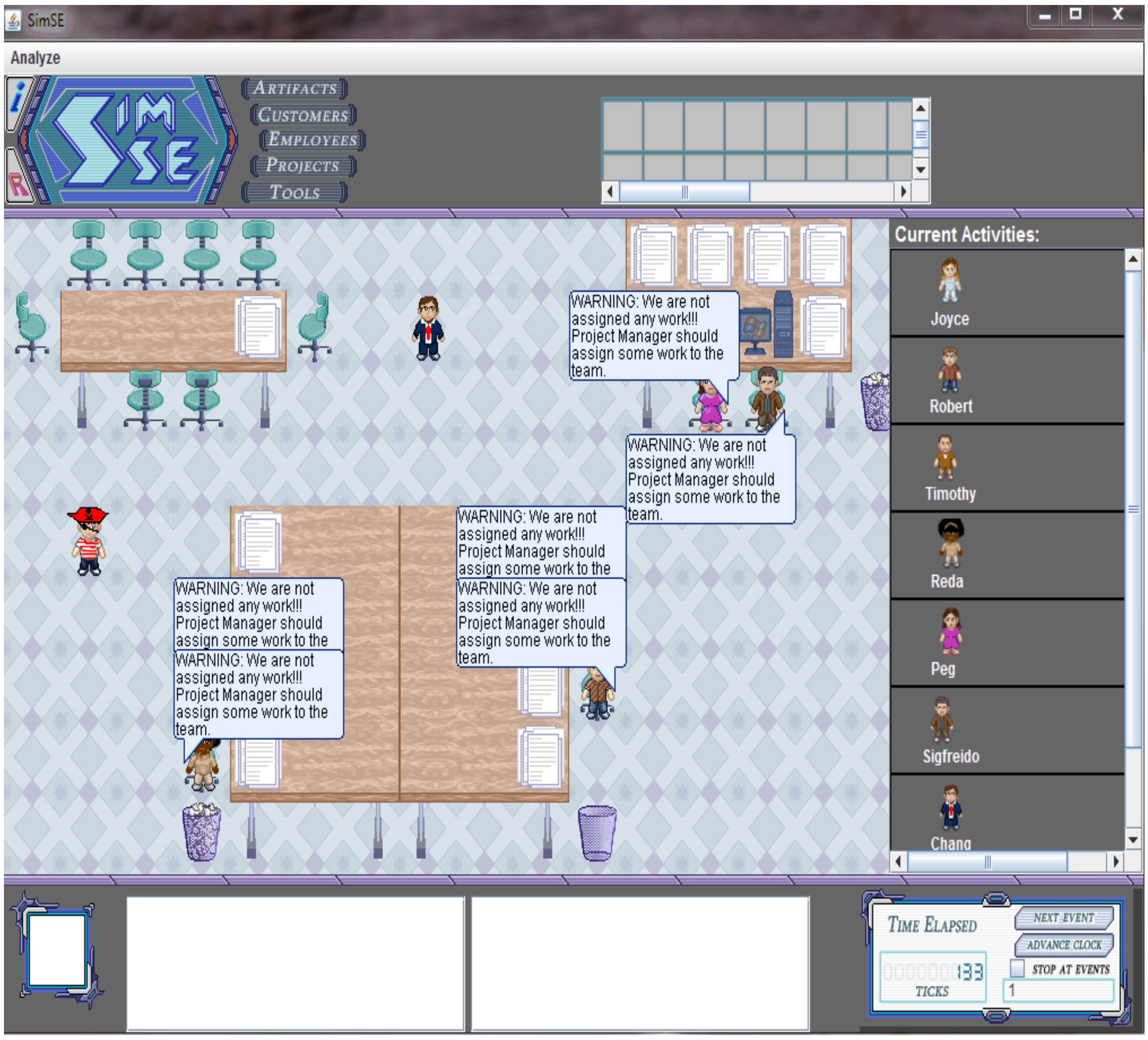

Employees At-A-Glance

SoftwareDevelopers:

Name	SoftwareDevelopmen...	KnowsCodingStandard	Idle
Joyce	8 years	✓	✓
Robert	1 year	✓	✓
Timothy	10 years	✓	✓
Reda	3 years	✓	✓
Peg	2 years	✓	✓
Sigfreido	9 years	✓	✓

Figure 54: Employees artefacts

As shown in figure 55, if the employees are idle then an early warning message would pop up saying “We are not assigned any work!!! Project Manager should assign some work to the team.”



WARNING: We are not assigned any work!!! Project Manager should assign some work to the team.

WARNING: We are not assigned any work!!! Project Manager should assign some work to the team.

WARNING: We are not assigned any work!!! Project Manager should assign some work to the

WARNING: We are not assigned any work!!! Project Manager should assign some work to the team.

WARNING: We are not assigned any work!!! Project Manager should assign some work to the

WARNING: We are not assigned any work!!! Project Manager should assign some work to the team.

Figure 55: Warning message pops up if the employees are idle

Once the warning message appears it is up to the user now (who is the project manager here) to assign some work to the team. If the user still misses it, the warning message would appear again after few clock ticks. This is hence continuous and insistent in nature. Depending on how delayed the user reacts to the warning message the project result varies. We have recorded 12 different scenarios (or data points) but only showing 6 data points in detail under Appendix C5. However, in the 'Analysis and Results' section we will discuss all the 12 data points.

4.6.1.5.1 Analysis and Results

Table 7 below provides the summary of time for releases and customer complains in different scenarios.

	Release 1	**Release 2**	**Release 3**	**Release 4**	**Number of times Customer Complains**
Best Case	543	956	1369	1782	0
EWS Case	549	964	1379	1799	0
Take 2	565	1000	1435	-	0
Take 3	586	1042	1498	-	3
Take 4	607	1084	1561	-	3
Take 5	628	1126	1624	-	6
Take 8	691	1252	-	-	9
Take 12	775	1420	-	-	10
Take 16	859	1588	-	-	12
Take 20	943	1777	-	-	16
Take 24	1028	-	-	-	16
Take 28	1112	-	-	-	16

Table 7: Summary of time for releases and customer complains in different scenarios

In terms of time for releases, best case has performed the best and 'take 28' case has performed the worst. The time for releases has gradually increased from best case to 'take 28' case.

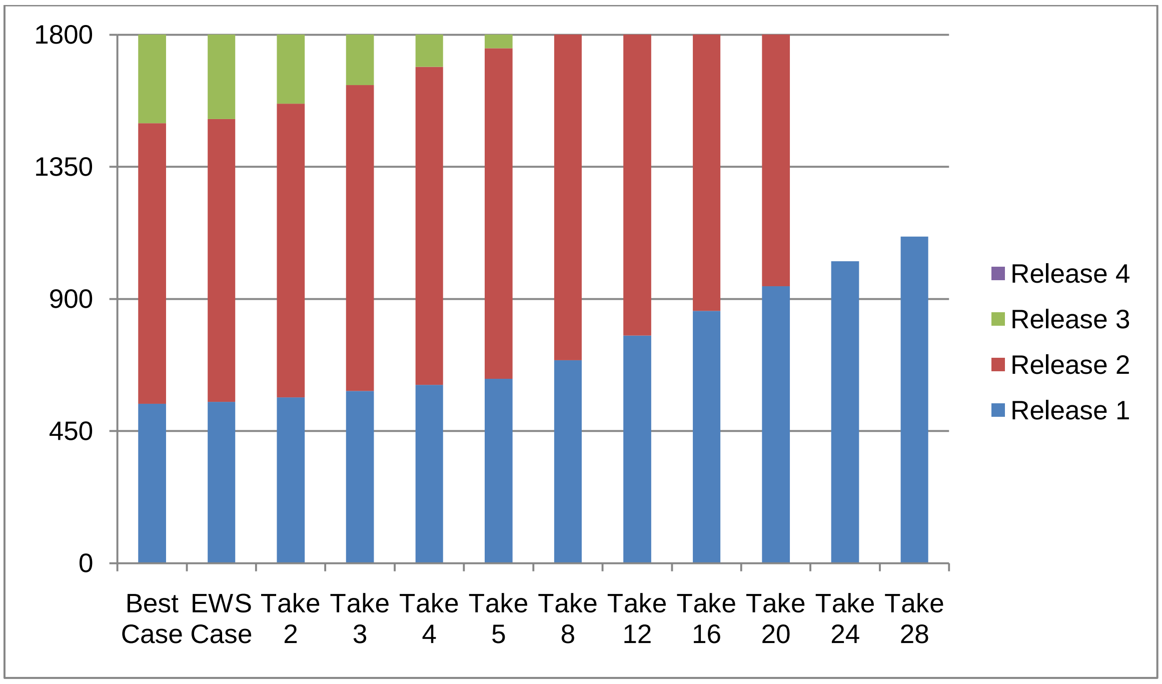


Figure 56: Graphical representation of summary of time taken by different scenarios

The second point of comparison is the completeness and correctness of the whole code (or product). The final state of the artefacts for each scenario represents the completeness and correctness of the product. Again, if NumUserStoriesIntegrated attribute is taken in to consideration then for best and EWS case the number of user stories integrated are 80; for 'take 4' case its 60; for 'take 12' and 'take 20' case its 40; and for 'take 28' case its 20. It is apparent that the earlier corrective action is applied the more user stories are integrated. Furthermore, if you refer to the individual scenarios explained earlier and then compare and analyse their final state of the artefacts it is apparent that the earlier corrective action is applied the better it is for the project.

The third point of comparison is the number of times the customer complains.

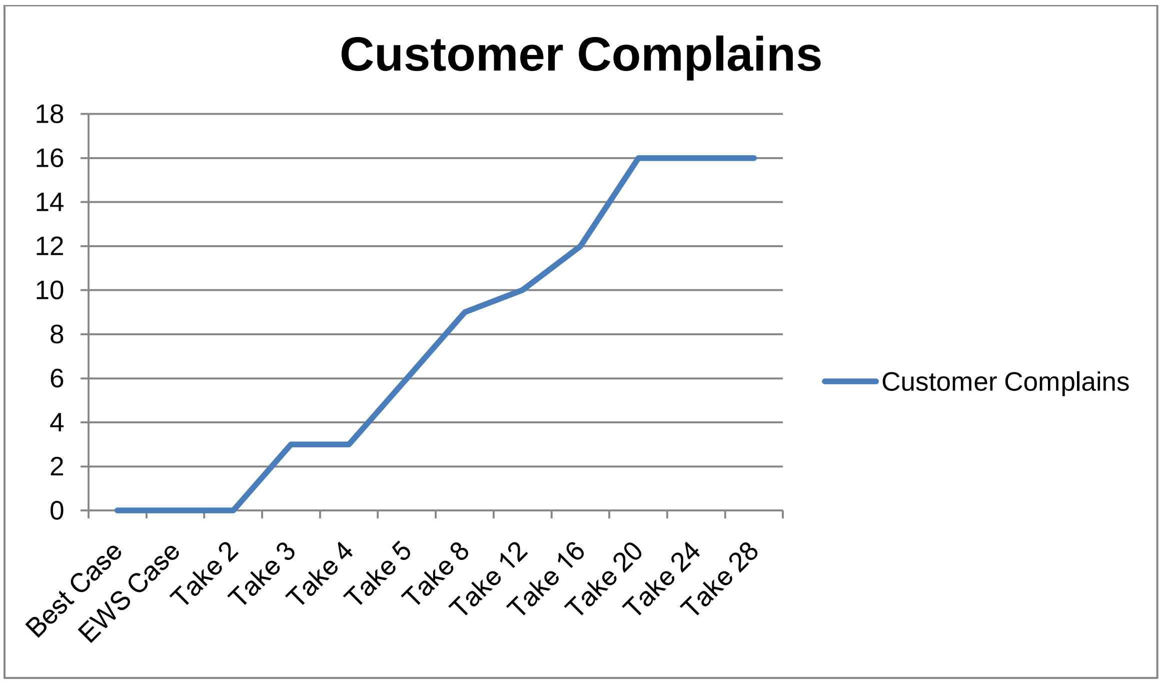


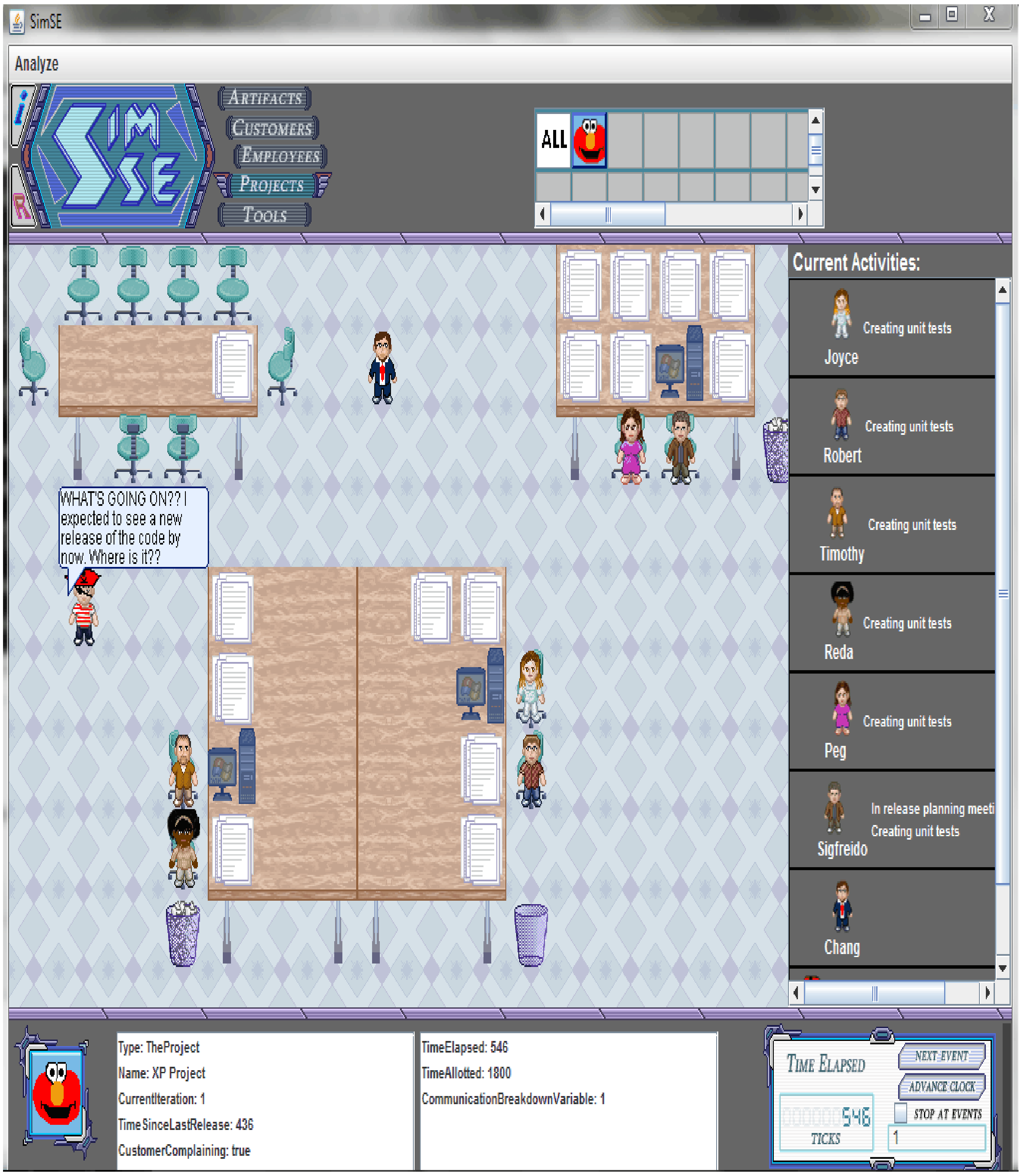
Figure 57: Graphical representation of Customer complains in different scenarios

From all the three points of comparison it is very clear and discernible that the earlier the user responds to the warning sign (message) the better it is for the project to meet its deadline. The best case is the ideal scenario and 'Take 28' is the worst scenario out of all the scenarios. The 'Take 28' scenario has the highest impact to the project and the project fails to meet the expectations. It is clear that the project performance deteriorated from the best case to 'take 28' case.

4.6.1.6 EWS6 - Communication breakdown among project stakeholders

Most of the early warning signs implemented in this research have an associated action once the warning message appears. For example, for EWS "lack of top management support or commitment to the project" we need to select action "Involve Management" once the warning message pops up. In real life this isn't always true, there may be instances where we always do not apply actions for some warning signs instead wait for some other sign or select a different action for other warning sign (here we keep the original warning sign (which wasn't having any action) in mind and combine with current warning sign to decide an appropriate action). In short, every early warning sign does not necessarily need to have an "immediate" action associated to it. In a project, the schedule has certain inertia and it may be that the action is in fact a delayed action, i.e. a modification of an already planned downstream activity.

In the implemented SimSE XP model, for this EWS, we don't have an immediate action associated to it. If a customer complains then it implies that there is communication breakdown among project stakeholders. As stated earlier, this SimSE XP model simulates a project which has a total of 4 releases. The customer is expecting these releases after certain clock ticks. If there is a change in this then it needs to be explicitly communicated to the project stakeholders (customer in this case). But if there is a communication breakdown then customer would still expect a release at certain times and would also complain if he doesn't get one. The following figure 58 shows a customer is complaining when he doesn't receive an expected new release. A customer complains if "TimeSinceLastRelease" attribute is greater than certain predefined value.

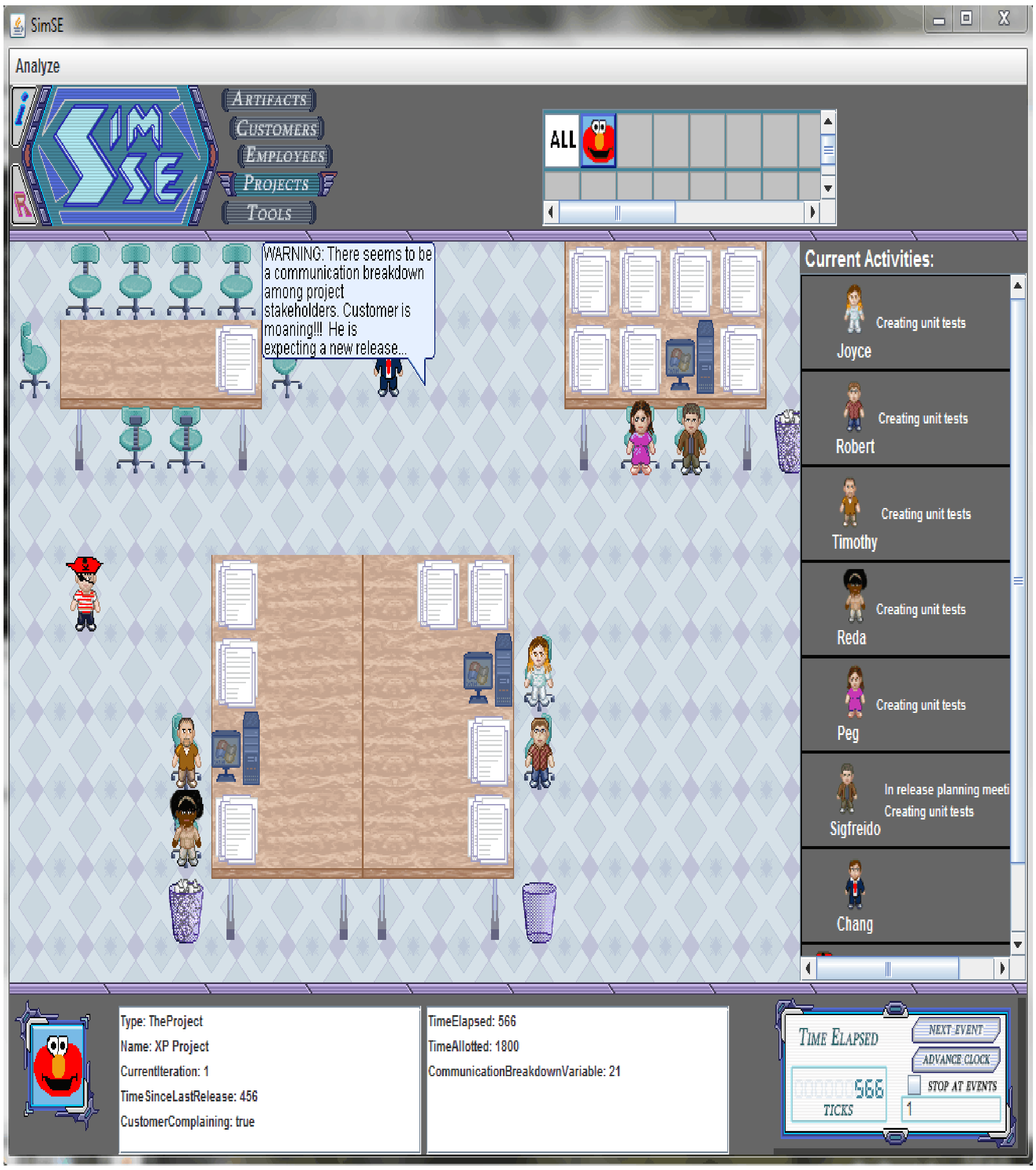


Type: TheProject
Name: XP Project
CurrentIteration: 1
TimeSinceLastRelease: 436
CustomerComplaining: true

TimeElapsed: 546
TimeAllotted: 1800
CommunicationBreakdownVariable: 1

Figure 58: Customer Complaining

As shown in figure 59, after few clock ticks, once the customer complains a warning message pops up saying, “There seems to be a communication breakdown among project stakeholders. Customer is moaning!!! He is expecting a new release.”



WARNING: There seems to be a communication breakdown among project stakeholders. Customer is moaning!!! He is expecting a new release...

Current Activities:

Joyce: Creating unit tests
Robert: Creating unit tests
Timothy: Creating unit tests
Reda: Creating unit tests
Peg: Creating unit tests
Sigfreido: In release planning meeti Creating unit tests
Chang

Type: TheProject
Name: XP Project
CurrentIteration: 1
TimeSinceLastRelease: 456
CustomerComplaining: true

TimeElapsed: 566
TimeAllotted: 1800
CommunicationBreakdownVariable: 21

Figure 59: Warning message for customer complaining

As suggested earlier, this EWS does not suggest any action(s) for warning message instead is designed to create awareness (rather than prescribing certain action(s)). This warning message may cause a modification of an already downstream planned activity in the schedule. The main reason for covering this type of EWS where we do not have any associated immediate action is to cover a range of real life scenarios.

4.6.2 Controlled Experimental Study Results

As said earlier, for controlled experiment all the six EWS were implemented in one integrated EWS model. This integrated EWS model simulates a realistic and dynamic project environment where multiple early warning signs can trigger simultaneously. The 10 volunteers were equally divided into two groups:

1) Control Group
2) EWS Group

The control group played the normal XP game twice whereas the EWS group played the normal XP game once followed by the integrated EWS XP game. The purpose of the control group is therefore to provide a baseline of comparison from which any implicit bias that can be attributed to playing the game (and “learning the rules”) is removed so that any benefit of the EWS approach can be appropriately quantified.

4.6.2.1 Control Group Results

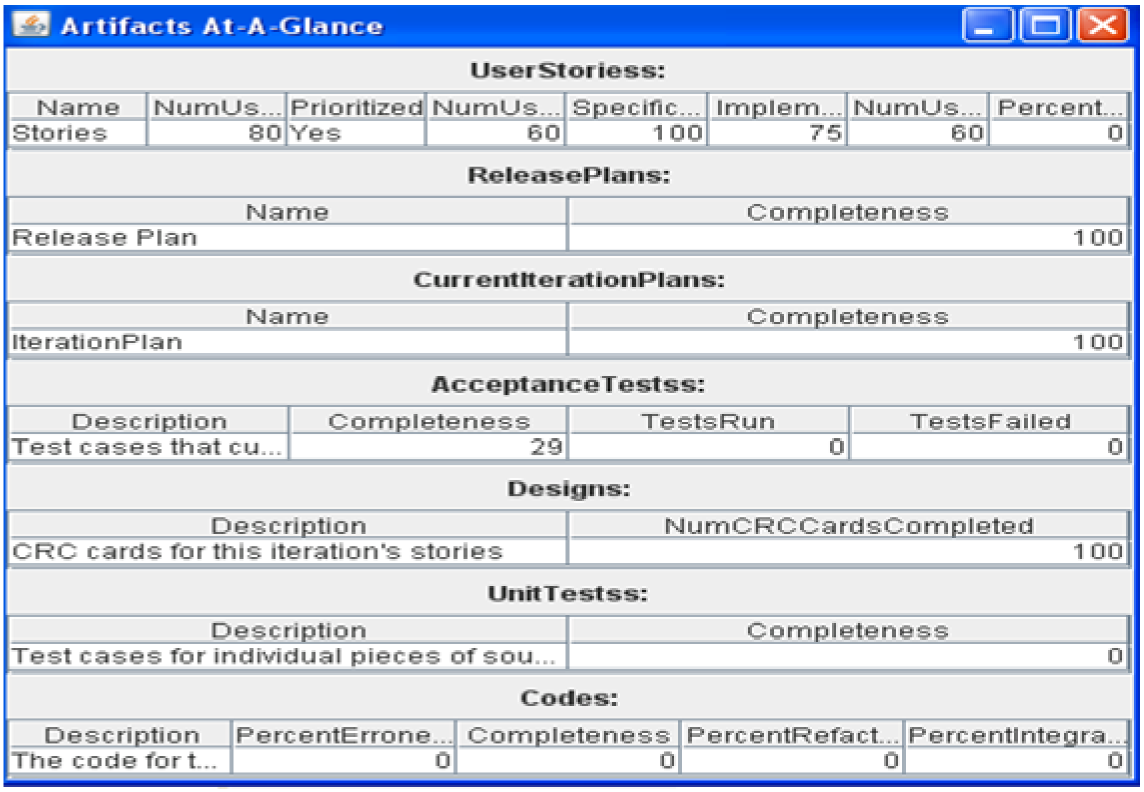
Out of ten volunteers, 5 belonged to the control group. The results of all the 5 volunteers are as follows:

4.6.2.1.1 Player 1

Player 1 rated their PM experience as high and their Agile/XP experience as medium. For Game1, release 1 was completed at clock tick 685; release 2 at 1098 and release 3 at 1639. Customer complained 6 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game1 - Normal XP game	685	1098	1639	-

Table 8: Clock ticks representing the completion of releases for game1



UserStoriess:

Name	NumUs...	Prioritized	NumUs...	Specific...	Implem...	NumUs...	Percent...
Stories	80	Yes	60	100	75	60	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that cu...	29	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of sou...	0

Codes:

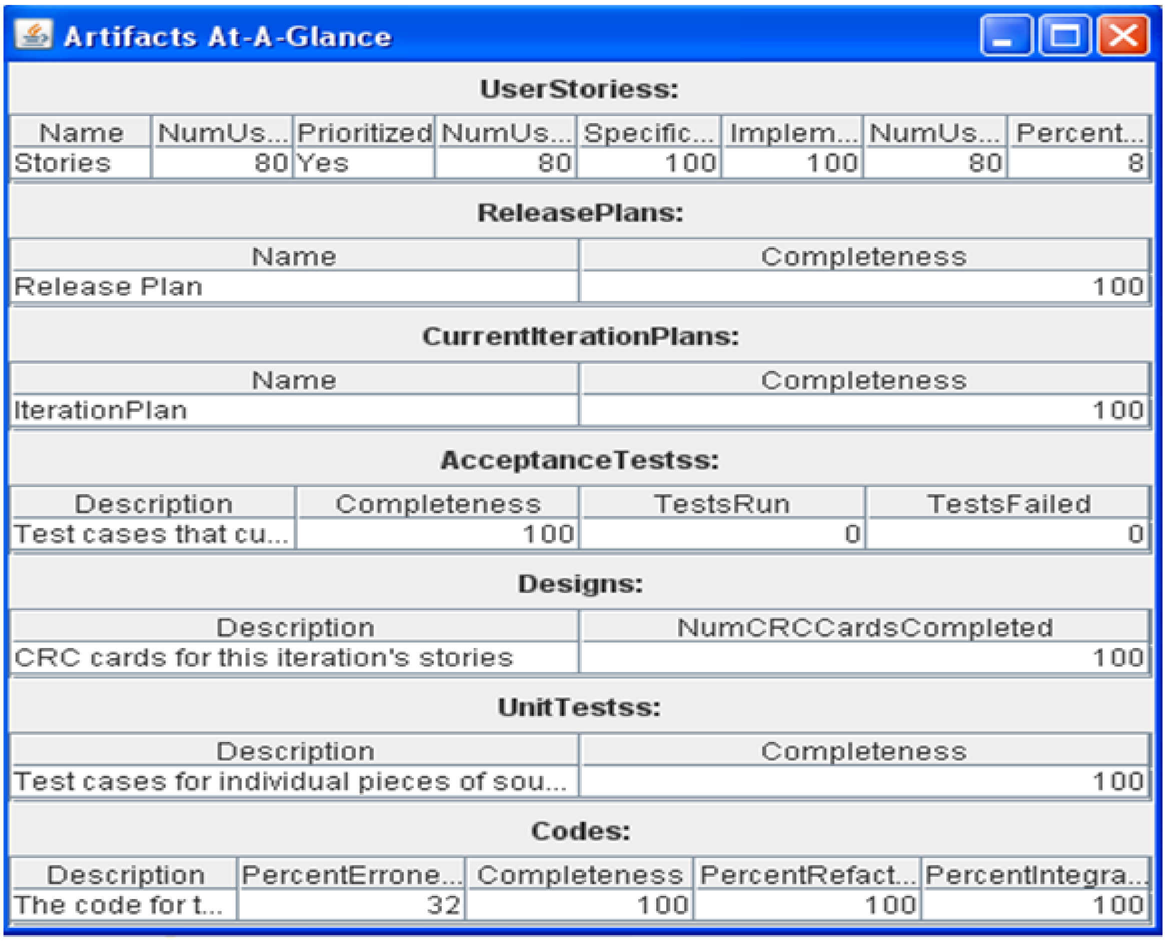
Description	PercentErrone...	Completeness	PercentRefact...	PercentIntegra...
The code for t...	0	0	0	0

Figure 60: Final state of the artefacts for game1

For Game2, release 1 was completed at clock tick 552; release 2 at 994 and release 3 at 1416. Customer complained 5 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game2 - Normal XP game	552	994	1416	-

Table 9: Clock ticks representing the completion of releases for game2


Artifacts At-A-Glance

UserStoriess:

Name	NumUs...	Prioritized	NumUs...	Specific...	Implem...	NumUs...	Percent...
Stories	80	Yes	80	100	100	80	8

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that cu...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of sou...	100

Codes:

Description	PercentErrone...	Completeness	PercentRefact...	PercentIntegra...
The code for t...	32	100	100	100

Figure 61: Final state of the artefacts for game2

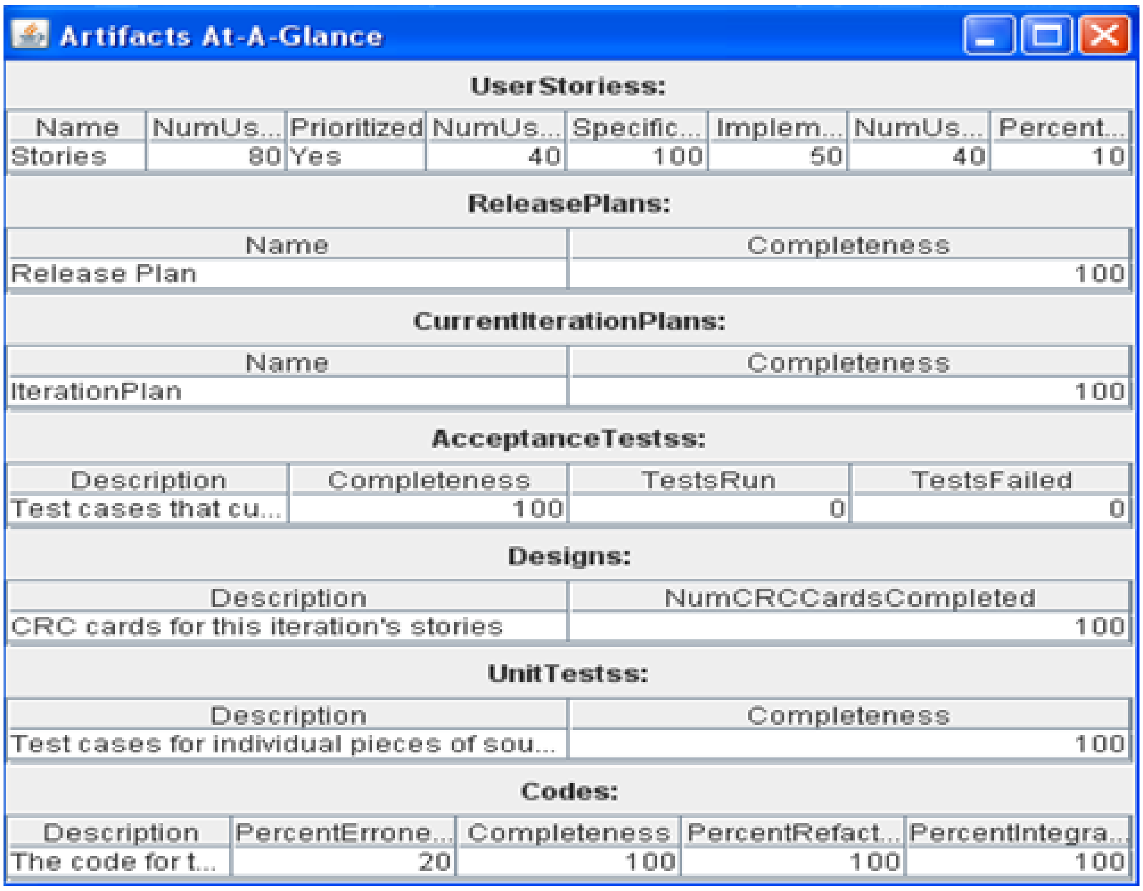
For player 1, there was slight improvement in the performance from game 1 to game 2. In game 1, the acceptance tests completeness in the 4th iteration was only 29%; and unit tests creation completeness and code completeness were 0. Whereas in game 2, the acceptance tests completeness, unit tests creation completeness and code completeness in the 4th iteration got up to 100%. The percent refactored and percent integration were also 100%. The implementation completeness and the total number of user stories integrated also increased from 75 and 60 to 100 and 80 respectively in game 2. But the percent erroneous (under Codes artifact) in the 4th iteration increased to 32% and hence overall the percent erroneous (under UserStories artifact) of the code was effectively increased to a total of 8%.

4.6.2.1.2 Player 2

Player 2 rated their PM experience as low and their Agile/XP experience as medium. For Game1, release 1 was completed at clock tick 1270. Customer complained 16 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game1 - Normal XP game	1270	-	-	-

Table 10: Clock ticks representing the completion of releases for game1



UserStoriess:

Name	NumUs...	Prioritized	NumUs...	Specific...	Implem...	NumUs...	Percent...
Stories	80	Yes	40	100	50	40	10

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that cu...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of sou...	100

Codes:

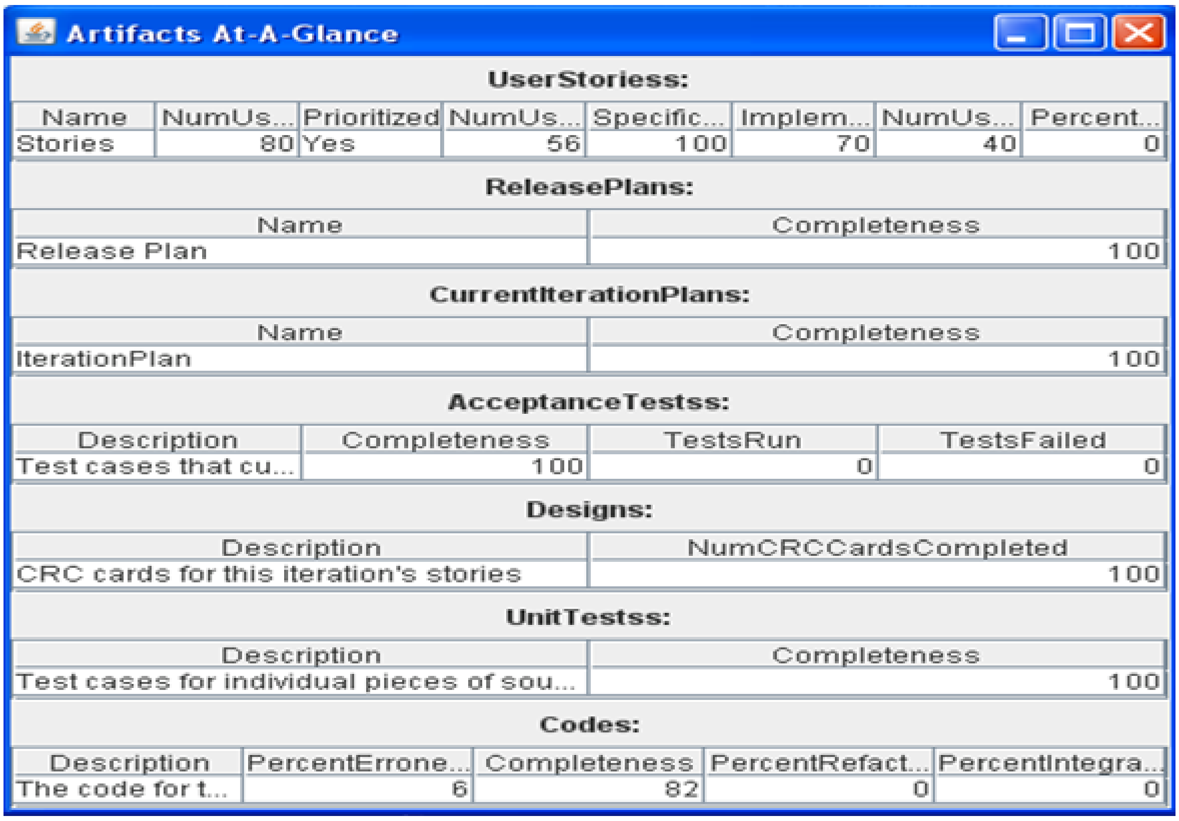
Description	PercentErrone...	Completeness	PercentRefact...	PercentIntegra...
The code for t...	20	100	100	100

Figure 62: Final state of the artefacts for game1

For Game2, release 1 was completed at clock tick 1020 and release 2 at 1478. Customer complained 11 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game2 - Normal XP game	1020	1478	-	-

Table 11: Clock ticks representing the completion of releases for game2



Artifacts At-A-Glance

UserStoriess:

Name	NumUs...	Prioritized	NumUs...	Specific...	Implem...	NumUs...	Percent...
Stories	80	Yes	56	100	70	40	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that cu...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of sou...	100

Codes:

Description	PercentErrone...	Completeness	PercentRefact...	PercentIntegra...
The code for t...	6	82	0	0

Figure 63: Final state of the artefacts for game2

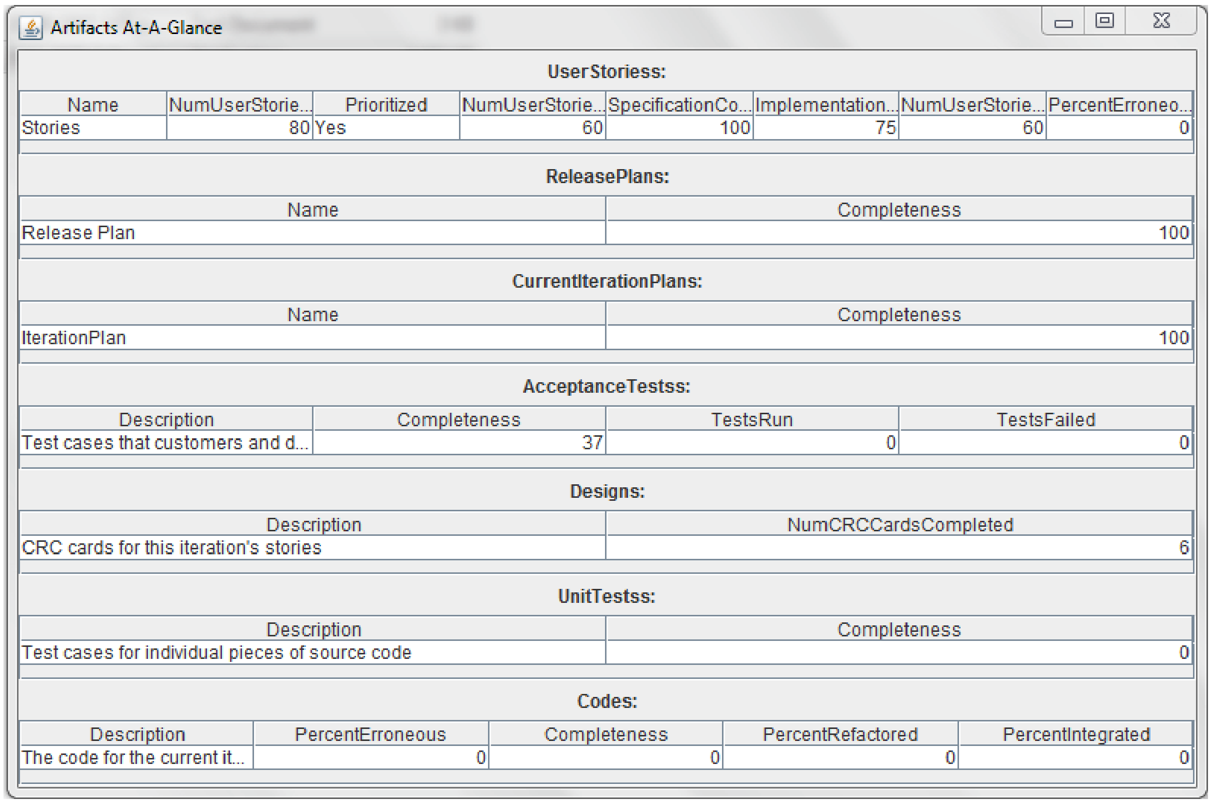
For player 2, there was slight improvement in the performance from game 1 to game 2. In game 1, the user was just about to end iteration 2 (as only running of acceptance tests was remaining). In game 2, the user managed to finish iteration 2 and got into iteration 3. In iteration 3, the code completeness was 82% and code erroneous was 6%. In game 1 the implementation completeness was 50 whereas in game 2 it was 70.

4.6.2.1.3 Player 3

Player 3 rated their PM experience as medium and their Agile/XP experience as low. For Game1, release 1 was completed at clock tick 738; release 2 at 1267 and release 3 at 1742. Customer complained 7 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game1 - Normal XP game	738	1267	1742	-

Table 12: Clock ticks representing the completion of releases for game1


Artifacts At-A-Glance

UserStoriess:

Name	NumUserStorie...	Prioritized	NumUserStorie...	SpecificationCo...	Implementation...	NumUserStorie...	PercentErroneo...
Stories	80	Yes	60	100	75	60	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers and d...	37	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	6

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	0

Codes:

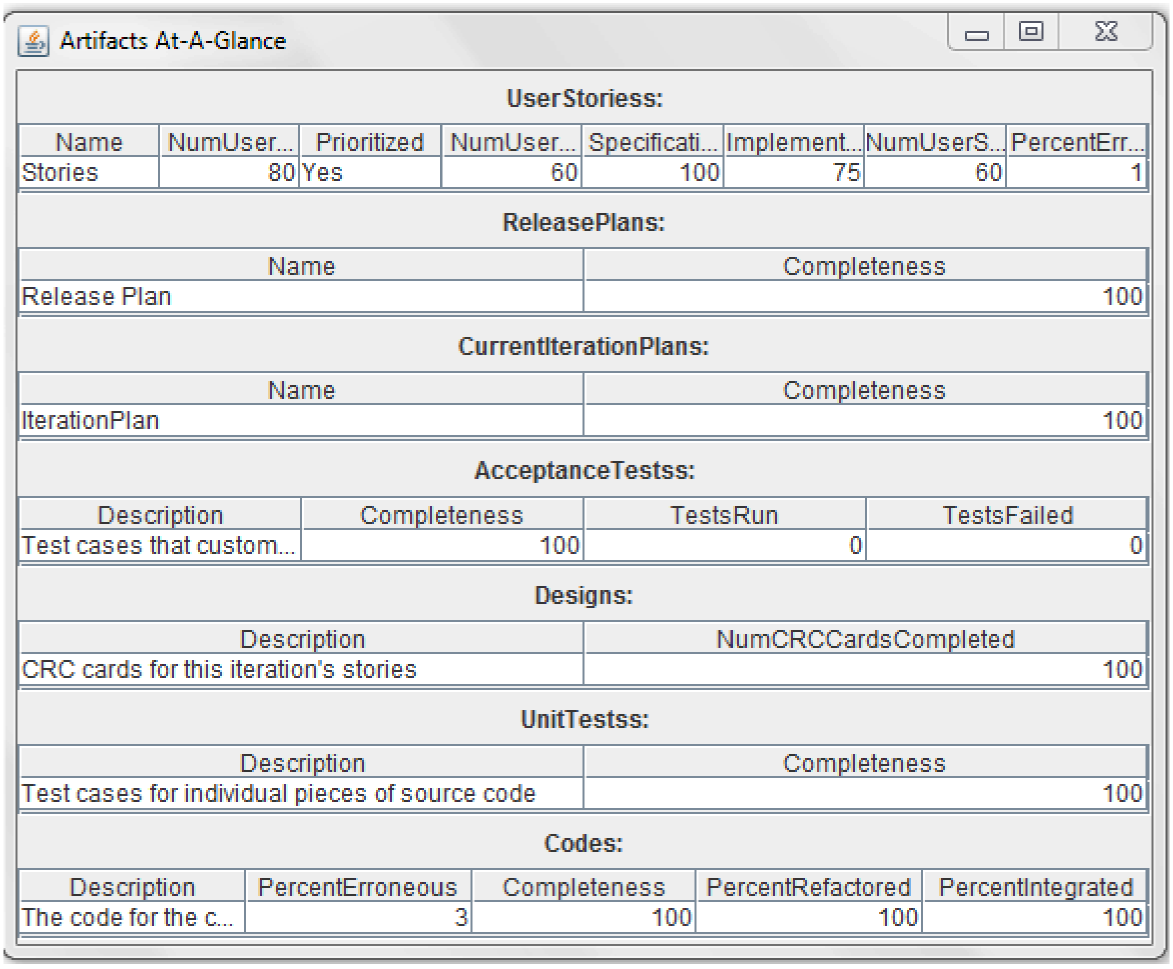
Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the current it...	0	0	0	0

Figure 64: Final state of the artefacts for game1

For Game2, release 1 was completed at clock tick 725 and release 2 at 1328. Customer complained 9 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game2 - Normal XP game	725	1328	-	-

Table 13: Clock ticks representing the completion of releases for game2



UserStoriess:

Name	NumUser...	Prioritized	NumUser...	Specificati...	Implement...	NumUserS...	PercentErr...
Stories	80	Yes	60	100	75	60	1

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that custom...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the c...	3	100	100	100

Figure 65: Final state of the artefacts for game2

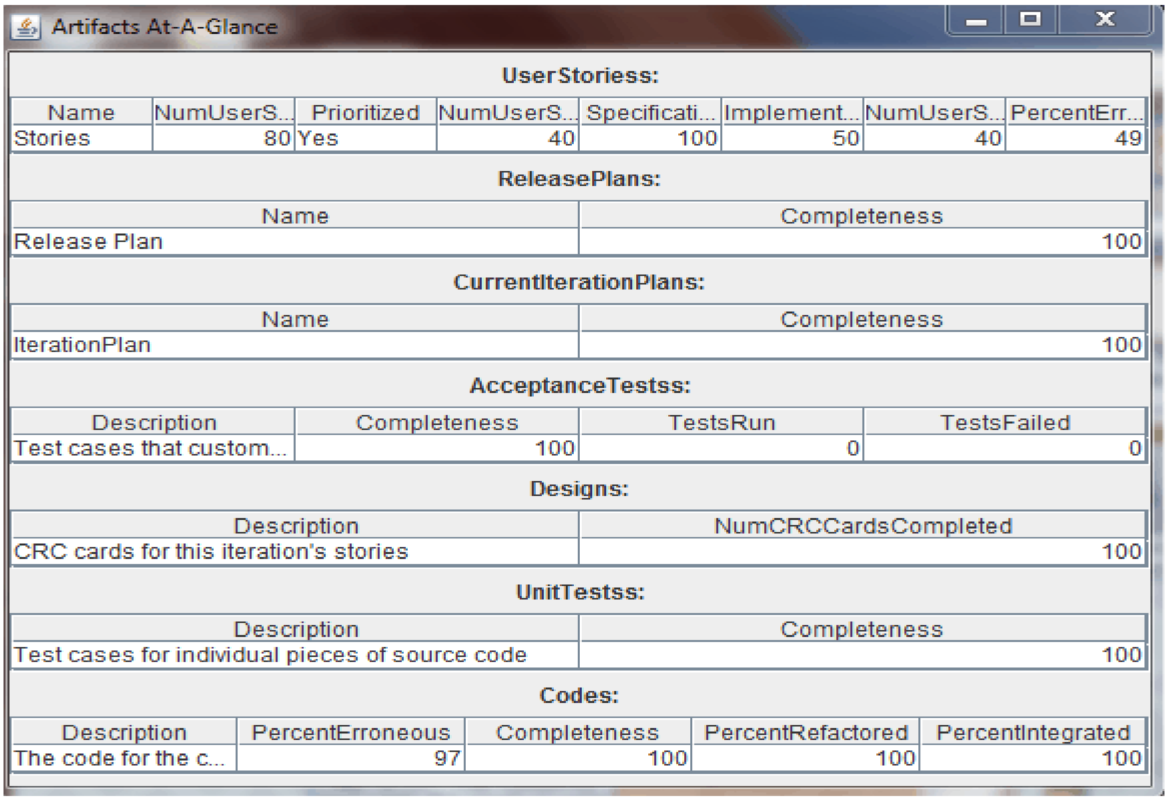
For player 3, the performance deteriorated from game 1 to game 2. In game 1, the user completed three iterations and was on 4th iteration whereas in game 2, the user completed only two iterations and was on 3rd iteration when the game ended. Also, the customer complained 7 times in game 1 whereas 9 times in game 2.

4.6.2.1.4 Player 4

Player 4 rated their PM experience as low and their Agile/XP experience as low (these experience levels represent a situation where PM is junior and working on a project employing new methodology for him). For Game1, release 1 was completed at clock tick 1329. Customer complained 17 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game1 - Normal XP game	1329	-	-	-

Table 14: Clock ticks representing the completion of releases for game1


Artifacts At-A-Glance

UserStoriess:

Name	NumUserS...	Prioritized	NumUserS...	Specificati...	Implement...	NumUserS...	PercentErr...
Stories	80	Yes	40	100	50	40	49

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that custom...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

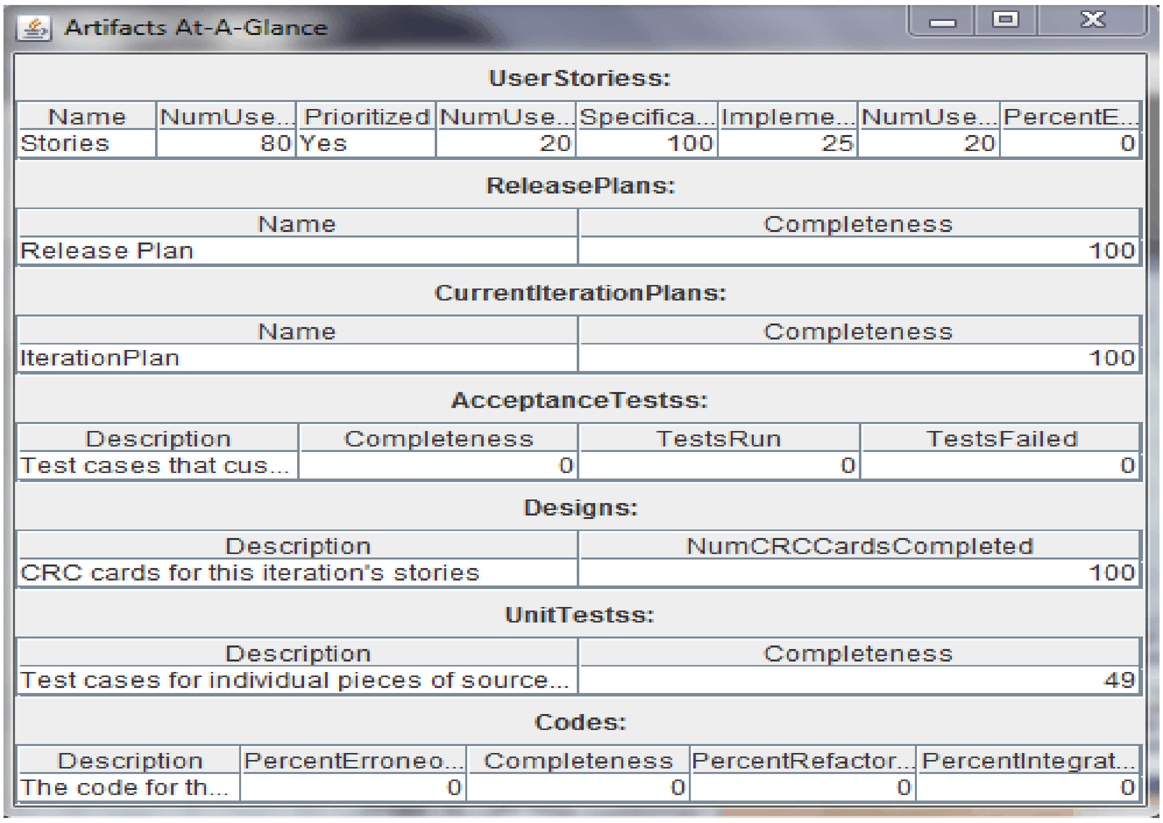
Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the c...	97	100	100	100

Figure 66: Final state of the artefacts for game1

For Game2, release 1 was completed at clock tick 1159. Customer complained 18 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game2 - Normal XP game	1159	-	-	-

Table 15: Clock ticks representing the completion of releases for game2



Artifacts At-A-Glance

UserStoriess:

Name	NumUse...	Prioritized	NumUse...	Specifica...	Impleme...	NumUse...	PercentE...
Stories	80	Yes	20	100	25	20	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that cus...	0	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source...	49

Codes:

Description	PercentErroneo...	Completeness	PercentRefactor...	PercentIntegrat...
The code for th...	0	0	0	0

Figure 67: Final state of the artefacts for game2

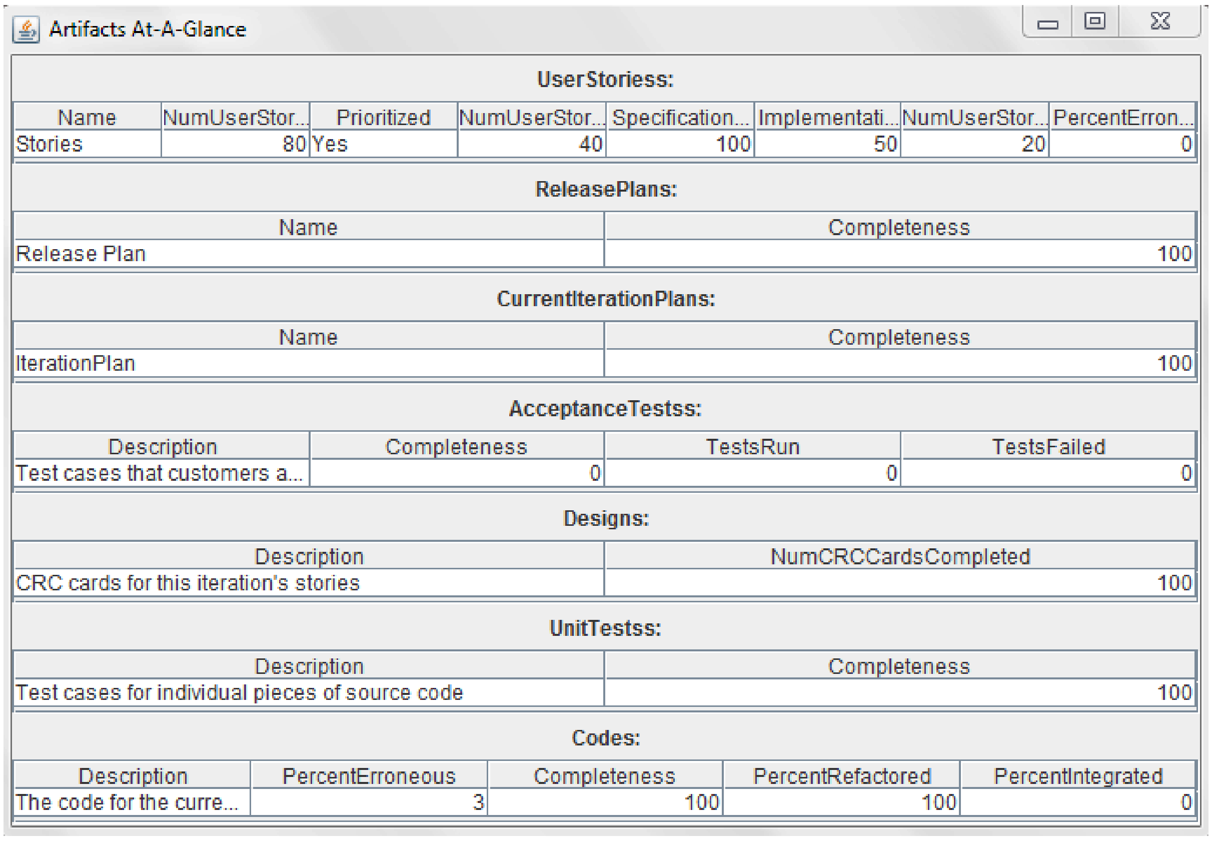
For player 4, both the games managed to finish only first iteration. Even though the time for first release was quicker in game 2 (compared to game 1) but there was no apparent improvement from game 1 to game 2. As in game 2, the unit tests creation completeness was only 49% and coding in the 2nd iteration was not even started where as in game 1, the code completeness and unit tests creation completeness were 100% but the percent erroneous was alarming 97%. Unlike other players, it is difficult to compare the final state of artefacts of game 1 and game 2 for player 4. Further, the customer complained 17 times in game 1 whereas 18 times in game 2.

4.6.2.1.5 Player 5

Player 5 rated their PM experience as low and their Agile/XP experience as low. For Game1, release 1 was completed at clock tick 1138. Customer complained 16 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game1 - Normal XP game	1138	-	-	-

Table 16: Clock ticks representing the completion of releases for game1


Artifacts At-A-Glance

UserStoriess:

Name	NumUserStor...	Prioritized	NumUserStor...	Specification...	Implementati...	NumUserStor...	PercentErron...
Stories	80	Yes	40	100	50	20	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers a...	0	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

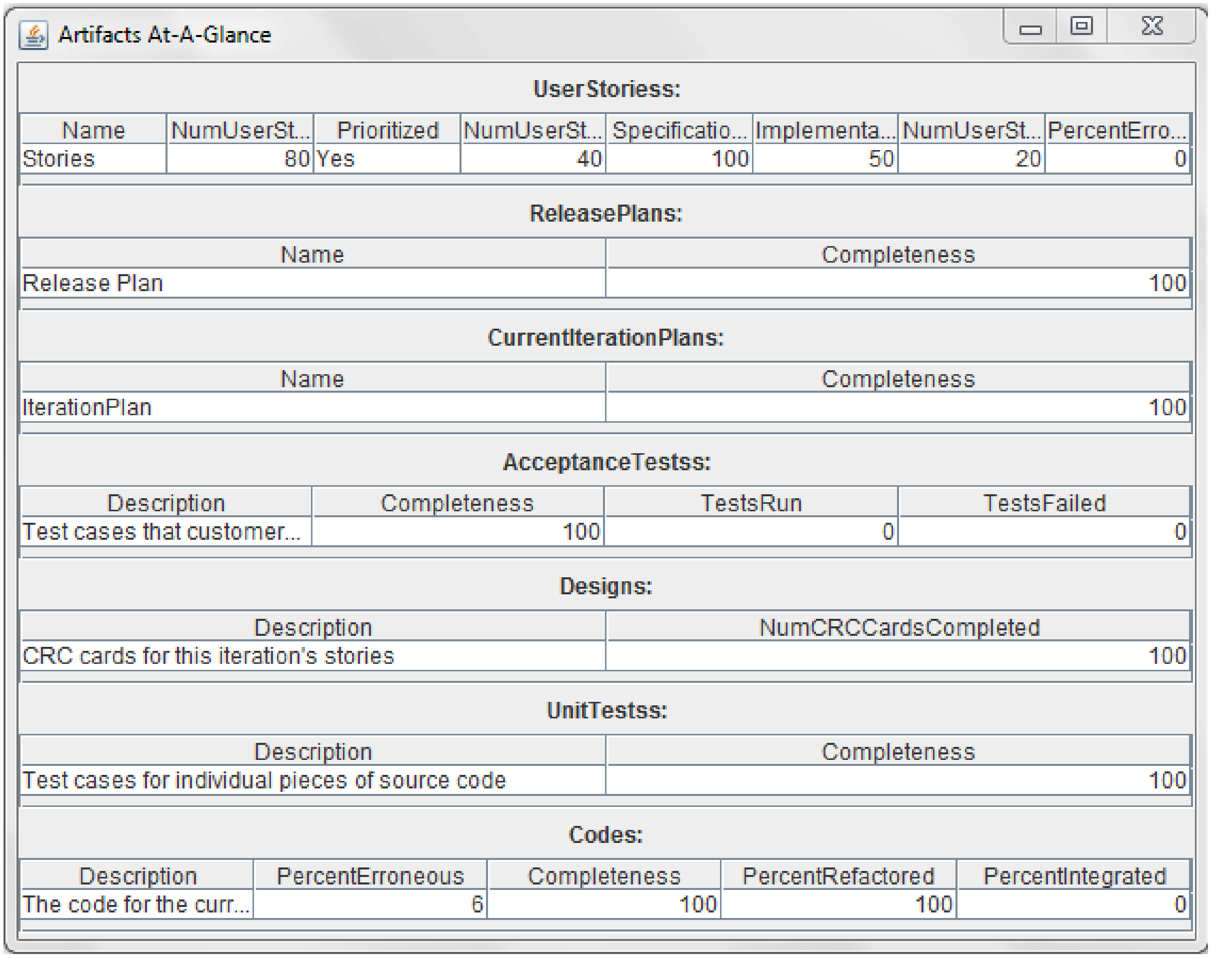
Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curre...	3	100	100	0

Figure 68: Final state of the artefacts for game1

For Game2, release 1 was completed at clock tick 1113. Customer complained 16 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game2 - Normal XP game	1113	-	-	-

Table 17: Clock ticks representing the completion of releases for game2



Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementa...	NumUserSt...	PercentErro...
Stories	80	Yes	40	100	50	20	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customer...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curr...	6	100	100	0

Figure 69: Final state of the artefacts for game2

For player 5, the final state of the artefacts for game 1 and game 2 were similar. Both the games finished their first iteration and were on second iteration when the game ended. In game 2 the percent erroneous of code in the 2nd iteration was 6% compared to 3% in game 1. Whereas, the acceptance tests completeness in game 2 increased to 100 from 0 in game 1. The number of customer complains were same in both the games (i.e. 16 times).

4.6.2.2 EWS Group Results

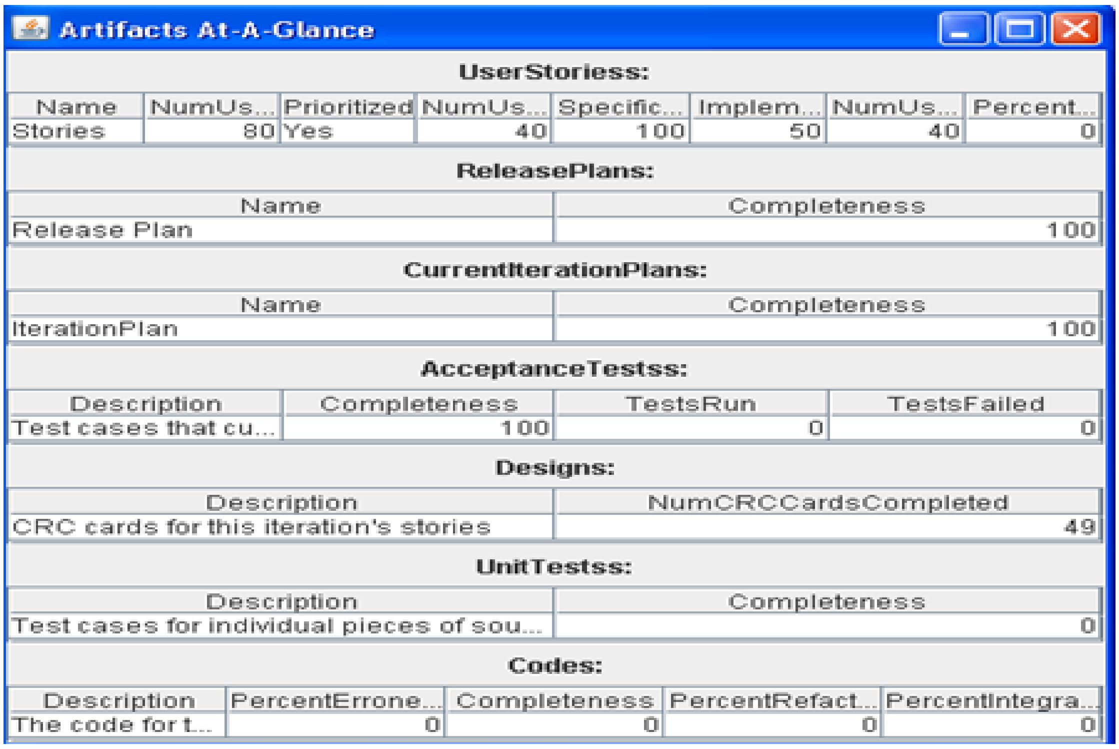
Out of ten volunteers, 5 belonged to the EWS group. The EWS group played the normal XP game once followed by the integrated EWS XP game. The results of all the 5 volunteers are as follows:

4.6.2.2.1 Player 6

Player 6 rated their PM experience as medium and their Agile/XP experience as low. For Game1, release 1 was completed at clock tick 830 and release 2 at 1708. Customer complained 15 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game1 - Normal XP game	830	1708	-	-

Table 18: Clock ticks representing the completion of releases for game1


Artifacts At-A-Glance

UserStoriess:

Name	NumUs...	Prioritized	NumUs...	Specific...	Implem...	NumUs...	Percent...
Stories	80	Yes	40	100	50	40	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that cu...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	49

UnitTestss:

Description	Completeness
Test cases for individual pieces of sou...	0

Codes:

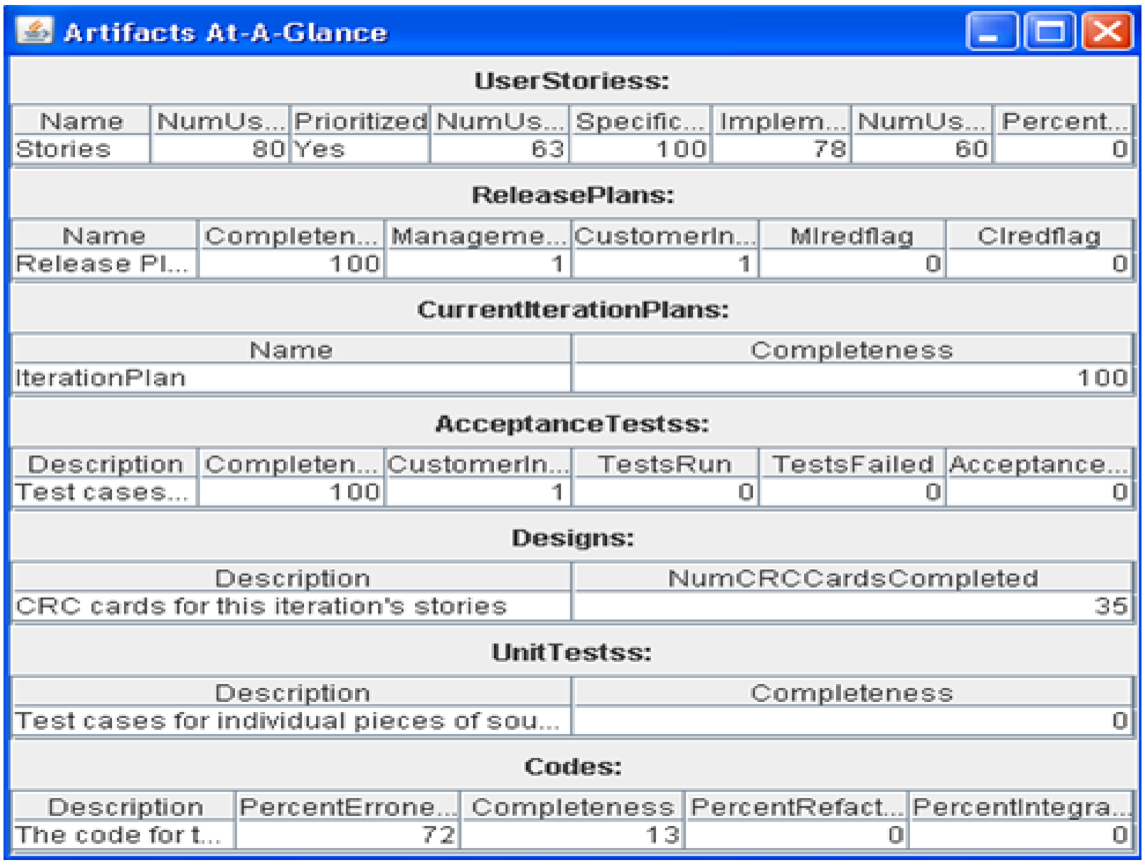
Description	PercentErrone...	Completeness	PercentRefact...	PercentIntegra...
The code for t...	0	0	0	0

Figure 70: Final state of the artefacts for game1

For Game2, release 1 was completed at clock tick 739; release 2 at 1231 and release 3 at 1672. Customer complained 7 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game2 – Integrated EWS XP game	739	1231	1672	-

Table 19: Clock ticks representing the completion of releases for game2


Artifacts At-A-Glance

UserStoriess:

Name	NumUs...	Prioritized	NumUs...	Specific...	Implem...	NumUs...	Percent...
Stories	80	Yes	63	100	78	60	0

ReleasePlans:

Name	Completen...	Manageme...	CustomerIn...	MIredflag	CIredflag
Release Pl...	100	1	1	0	0

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completen...	CustomerIn...	TestsRun	TestsFailed	Acceptance...
Test cases...	100	1	0	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	35

UnitTestss:

Description	Completeness
Test cases for individual pieces of sou...	0

Codes:

Description	PercentErrone...	Completeness	PercentRefact...	PercentIntegra...
The code for t...	72	13	0	0

Figure 71: Final state of the artefacts for game2

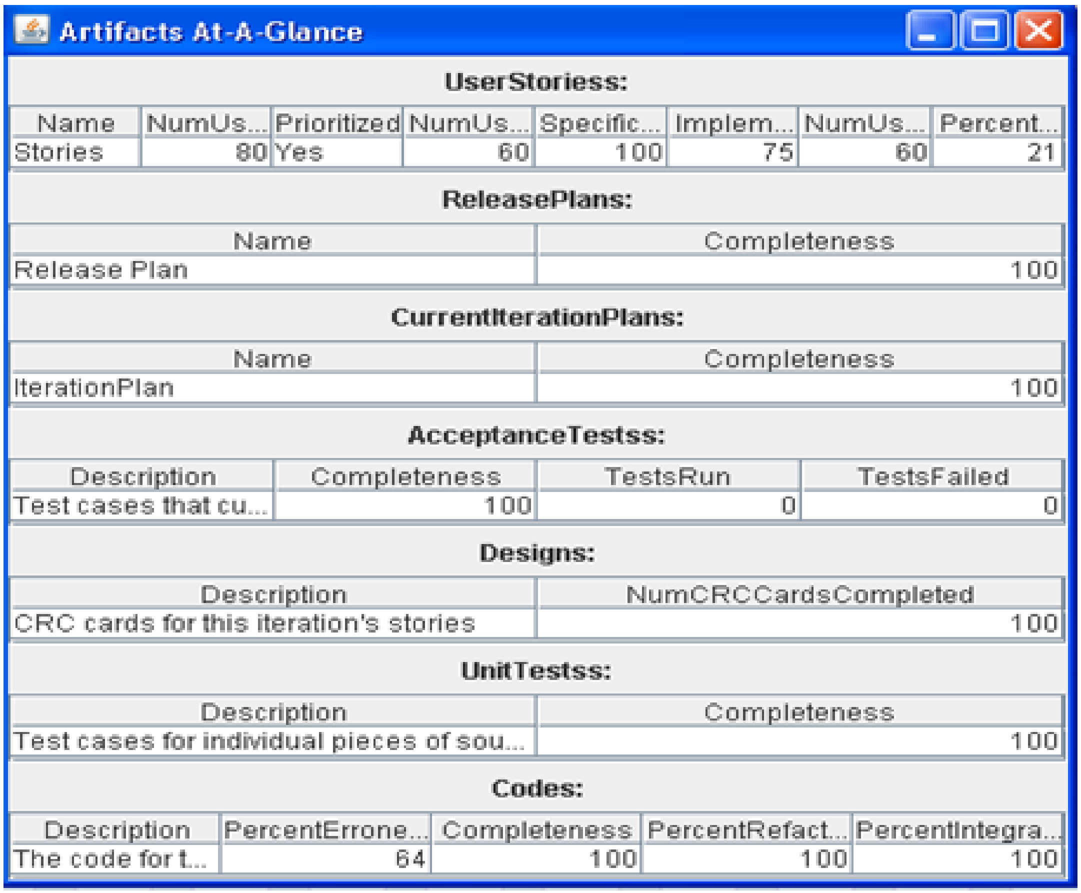
For player 6, there was an apparent improvement from game 1 to game 2. In game 1, the user was able to finish two iterations and was in iteration 3 when the game ended whereas in game 2, the user was able to finish three iterations and was in iteration 4 when the game ended. In game 1, the implementation completeness was 50 and the total number of user stories integrated was 40. In game 2, the implementation completeness was 78 and the total number of user stories integrated was 60. In game 1, the total number of times the customer complained was 15 whereas in game 2, it was just 7.

4.6.2.2.2 Player 7

Player 7 rated their PM experience as high and their Agile/XP experience as high. For Game1, release 1 was completed at clock tick 706 and release 2 at 1276. Customer complained 8 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game1 - Normal XP game	706	1276	-	-

Table 20: Clock ticks representing the completion of releases for game1


Artifacts At-A-Glance

UserStoriess:

Name	NumUs...	Prioritized	NumUs...	Specific...	Implem...	NumUs...	Percent...
Stories	80	Yes	60	100	75	60	21

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that cu...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of sou...	100

Codes:

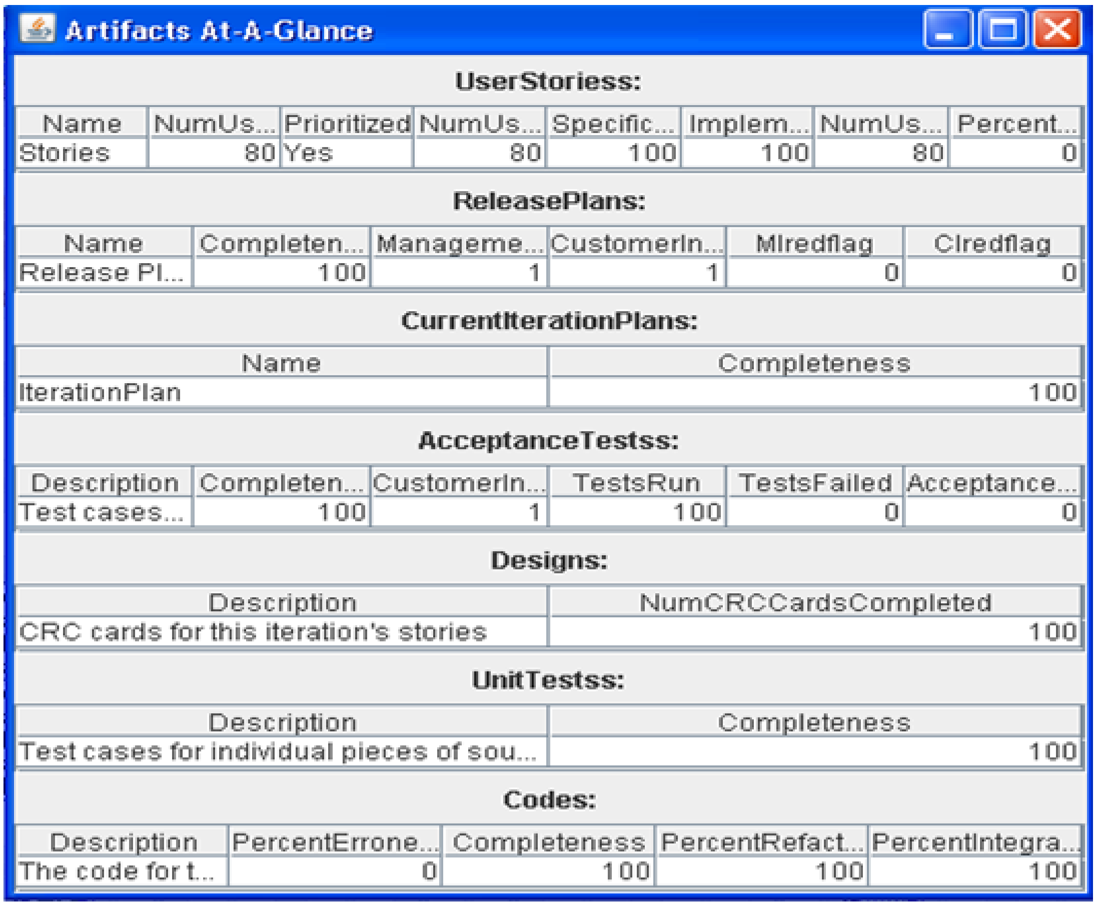
Description	PercentErrone...	Completeness	PercentRefact...	PercentIntegra...
The code for t...	64	100	100	100

Figure 72: Final state of the artefacts for game1

For Game2, release 1 was completed at clock tick 540; release 2 at 958; release 3 at 1376 and release 4 at 1790. Customer never complained throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game2 - Integrated EWS XP game	540	958	1376	1790

Table 21: Clock ticks representing the completion of releases for game2



UserStoriess:

Name	NumUs...	Prioritized	NumUs...	Specific...	Implem...	NumUs...	Percent...
Stories	80	Yes	80	100	100	80	0

ReleasePlans:

Name	Completen...	Manageme...	CustomerIn...	MIredflag	CIredflag
Release Pl...	100	1	1	0	0

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completen...	CustomerIn...	TestsRun	TestsFailed	Acceptance...
Test cases...	100	1	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of sou...	100

Codes:

Description	PercentErrone...	Completeness	PercentRefact...	PercentIntegra...
The code for t...	0	100	100	100

Figure 73: Final state of the artefacts for game2

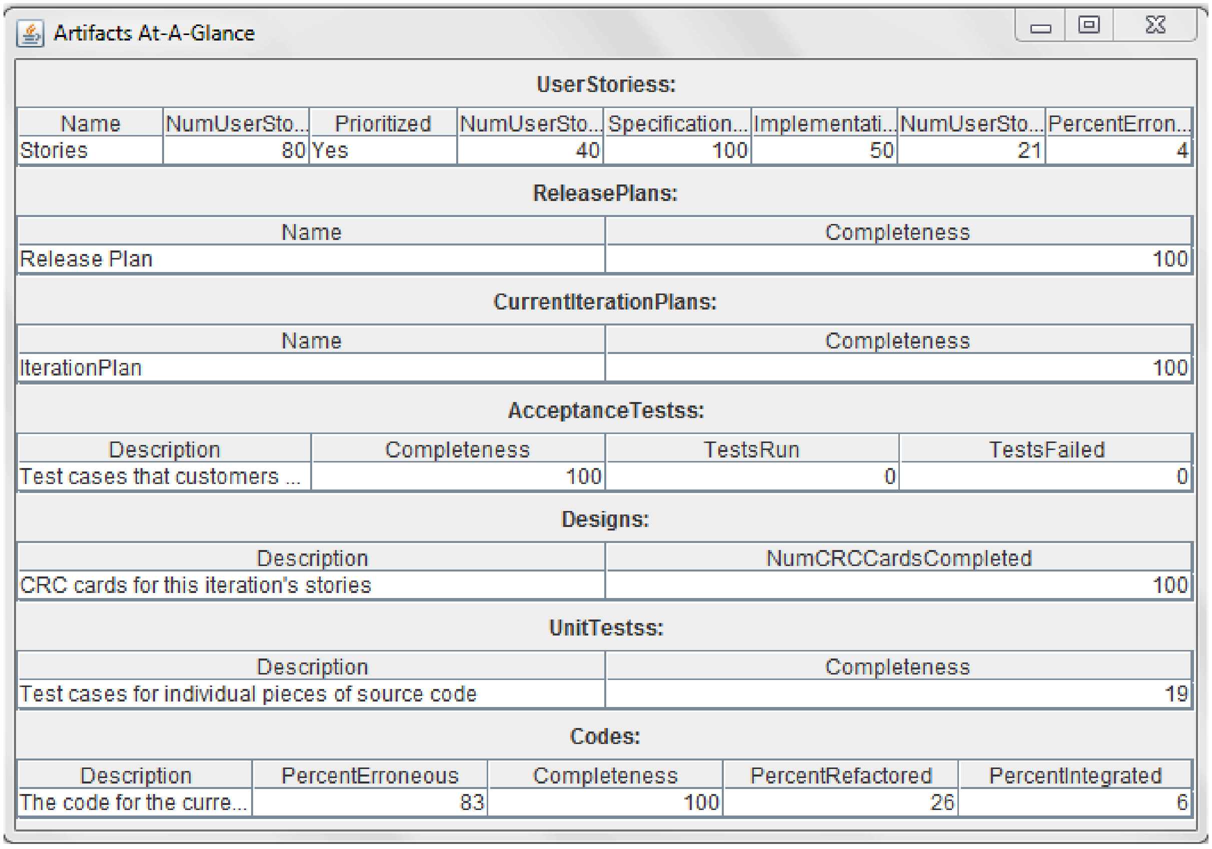
For player 7, there was an apparent improvement from game 1 to game 2. In game 1, the user was able to finish two iterations and was in iteration 3 when the game ended whereas in game 2, the user finished all the four iterations. The fourth release was finished at clock tick 1790 which was before the deadline of 1800 clock ticks. In game 1, the implementation completeness was 75 and the total number of user stories integrated was 60. In game 2, the implementation completeness was 100 and the total number of user stories integrated was 80. Also, the percent erroneous code dropped to 0% in game 2 from 21% in game 1. In game 1, the total number of times the customer complained was 8 whereas in game 2, the customer never complained. Game 2 was similar to the best case as defined in earlier sections.

4.6.2.2.3 Player 8

Player 8 rated their PM and Agile/XP experience as low. For Game1, release 1 was completed at clock tick 1349. Customer complained 17 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game1 - Normal XP game	1349	-	-	-

Table 22: Clock ticks representing the completion of releases for game1


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSto...	Prioritized	NumUserSto...	Specification...	Implementati...	NumUserSto...	PercentErron...
Stories	80	Yes	40	100	50	21	4

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers ...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	19

Codes:

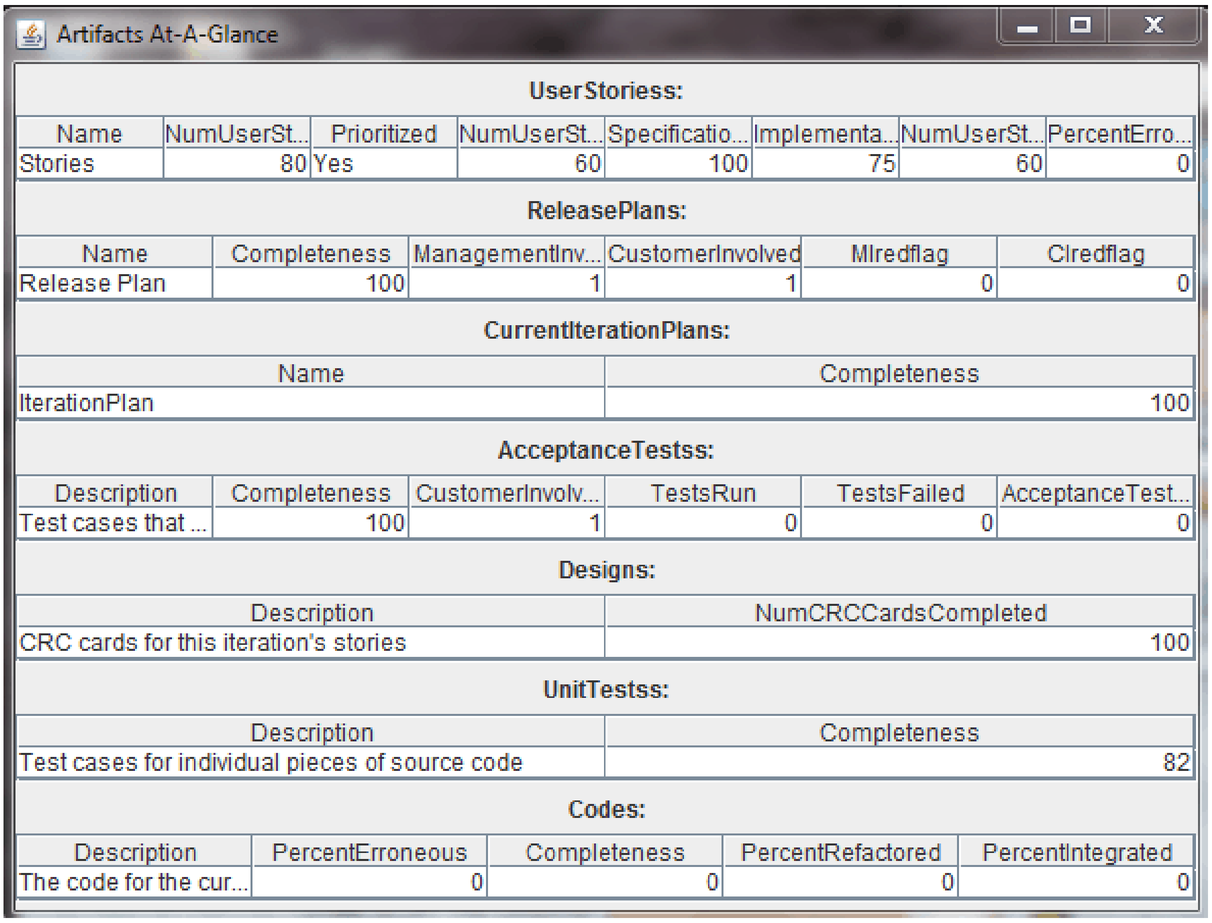
Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curre...	83	100	26	6

Figure 74: Final state of the artefacts for game1

For Game2, release 1 was completed at clock tick 666; release 2 at 1154 and release 3 at 1642. Customer complained 7 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game2 - Integrated EWS XP game	666	1154	1642	-

Table 23: Clock ticks representing the completion of releases for game2



UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementa...	NumUserSt...	PercentErro...
Stories	80	Yes	60	100	75	60	0

ReleasePlans:

Name	Completeness	ManagementInv...	CustomerInvolved	MIredflag	CIredflag
Release Plan	100	1	1	0	0

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	CustomerInvolv...	TestsRun	TestsFailed	AcceptanceTest...
Test cases that ...	100	1	0	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	82

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the cur...	0	0	0	0

Figure 75: Final state of the artefacts for game2

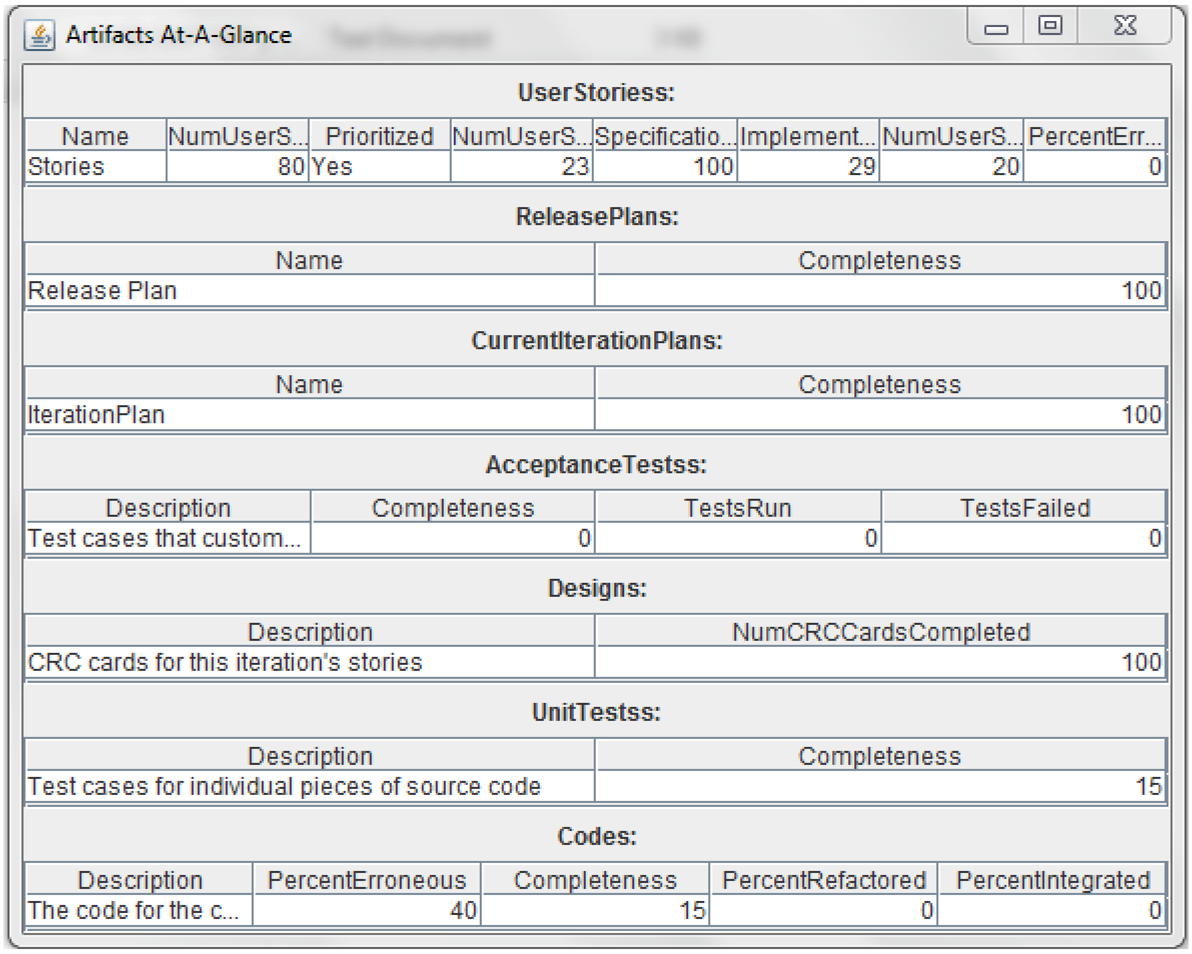
For player 8, there was an apparent improvement from game 1 to game 2. In game 1, the user was able to finish first iteration and was in iteration 2 when the game ended whereas in game 2, the user was able to finish three iterations and was in iteration 4 when the game ended. In game 1, the implementation completeness was 50 and the total number of user stories integrated was 21. In game 2, the implementation completeness was 75 and the total number of user stories integrated was 60. Further, in game 2 the percent erroneous of code dropped to 0% (from 4% in game 1). In game 1, the total number of times the customer complained was 17 whereas in game 2, it was just 7.

4.6.2.2.4 Player 9

Player 9 rated their PM and Agile/XP experience as low. For Game1, release 1 was completed at clock tick 1557. Customer complained 20 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game1 - Normal XP game	1557	-	-	-

Table 24: Clock ticks representing the completion of releases for game1



Artifacts At-A-Glance

UserStoriess:

Name	NumUserS...	Prioritized	NumUserS...	Specificatio...	Implement...	NumUserS...	PercentErr...
Stories	80	Yes	23	100	29	20	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that custom...	0	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	15

Codes:

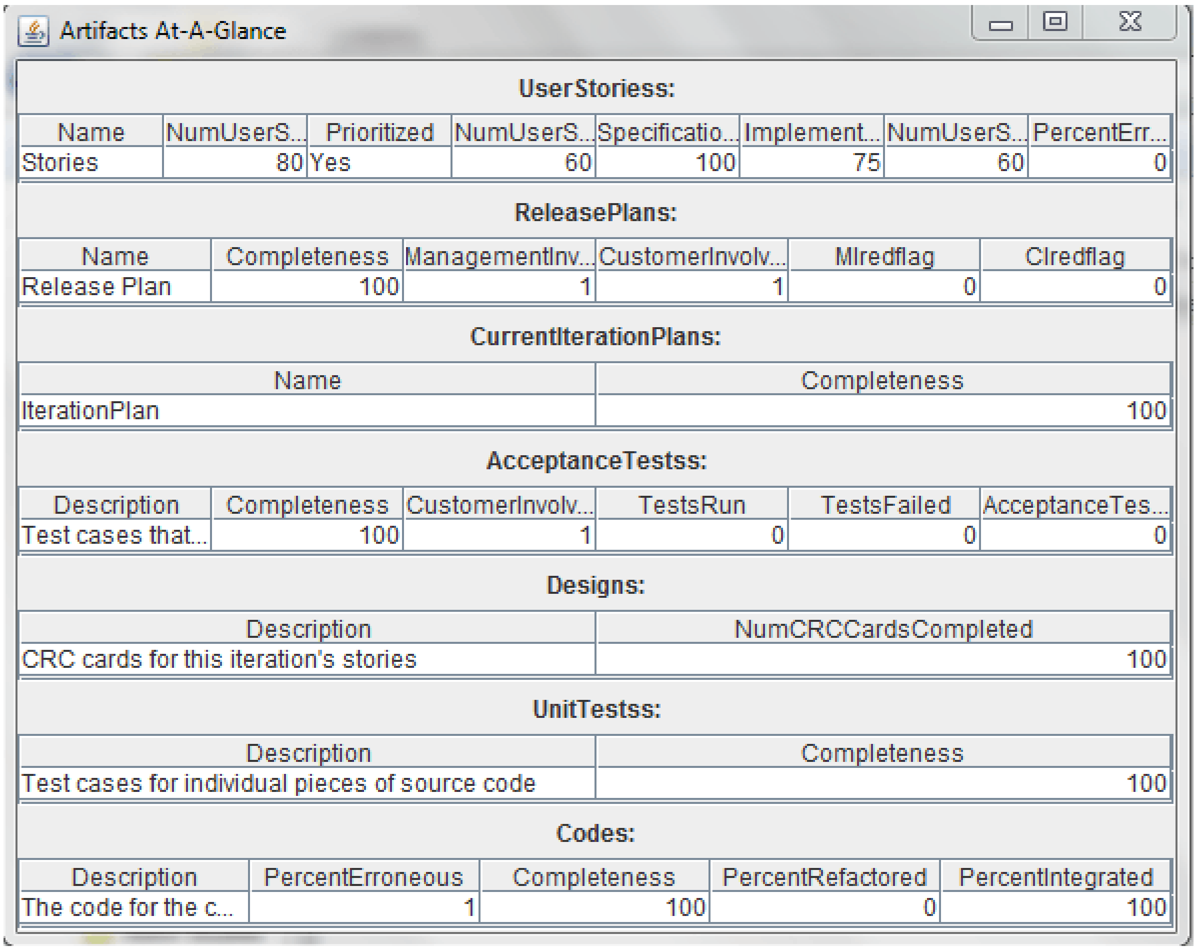
Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the c...	40	15	0	0

Figure 76: Final state of the artefacts for game1

For Game2, release 1 was completed at clock tick 835 and release 2 at 1322. Customer complained 9 times throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game2 – Integrated EWS XP game	835	1322	-	-

Table 25: Clock ticks representing the completion of releases for game2


Artifacts At-A-Glance

UserStoriess:

Name	NumUserS...	Prioritized	NumUserS...	Specificatio...	Implement...	NumUserS...	PercentErr...
Stories	80	Yes	60	100	75	60	0

ReleasePlans:

Name	Completeness	ManagementInv...	CustomerInvolv...	MIredflag	CIredflag
Release Plan	100	1	1	0	0

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	CustomerInvolv...	TestsRun	TestsFailed	AcceptanceTes...
Test cases that...	100	1	0	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the c...	1	100	0	100

Figure 77: Final state of the artefacts for game2

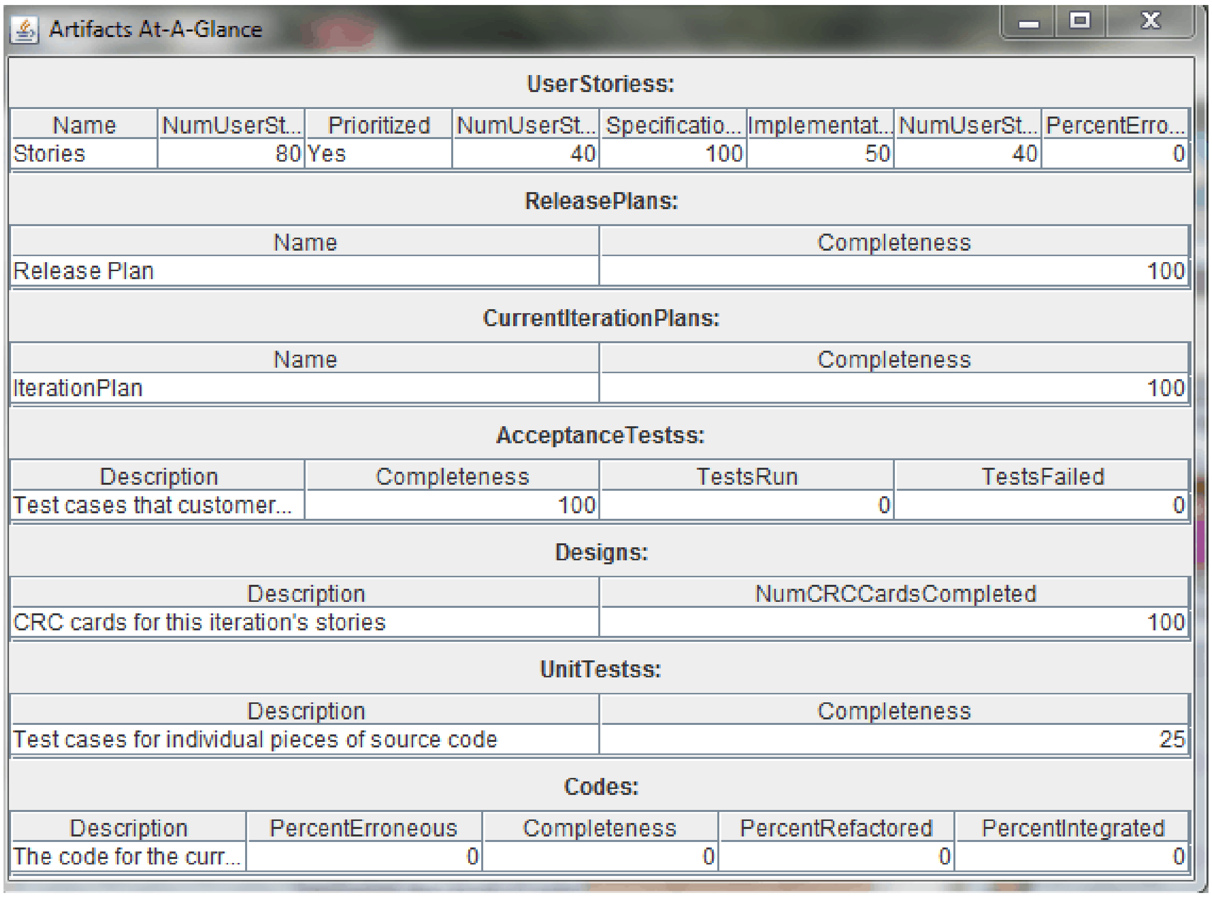
For player 9, there was an apparent improvement from game 1 to game 2. In game 1, the user was able to finish first iteration and was in iteration 2 when the game ended whereas in game 2, the user was able to finish two iterations and was in iteration 3 when the game ended. In game 1, the implementation completeness was 29% and the total number of user stories integrated was 20. In game 2, the implementation completeness was 75% and the total number of user stories integrated was 60. In game 1, the total number of times the customer complained was 20 whereas in game 2, it was just 9.

4.6.2.2.5 Player 10

Player 10 rated their PM experience as low and their Agile/XP experience as medium. For Game1, release 1 was completed at clock tick 1180 and release 2 at 1667. Customer complained 14 times throughout the game.

	Release 1	**Release 2**	**Release 3**	**Release 4**
Game1 - Normal XP game	1180	1667	-	-

Table 26: Clock ticks representing the completion of releases for game1


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementat...	NumUserSt...	PercentErro...
Stories	80	Yes	40	100	50	40	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customer...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	25

Codes:

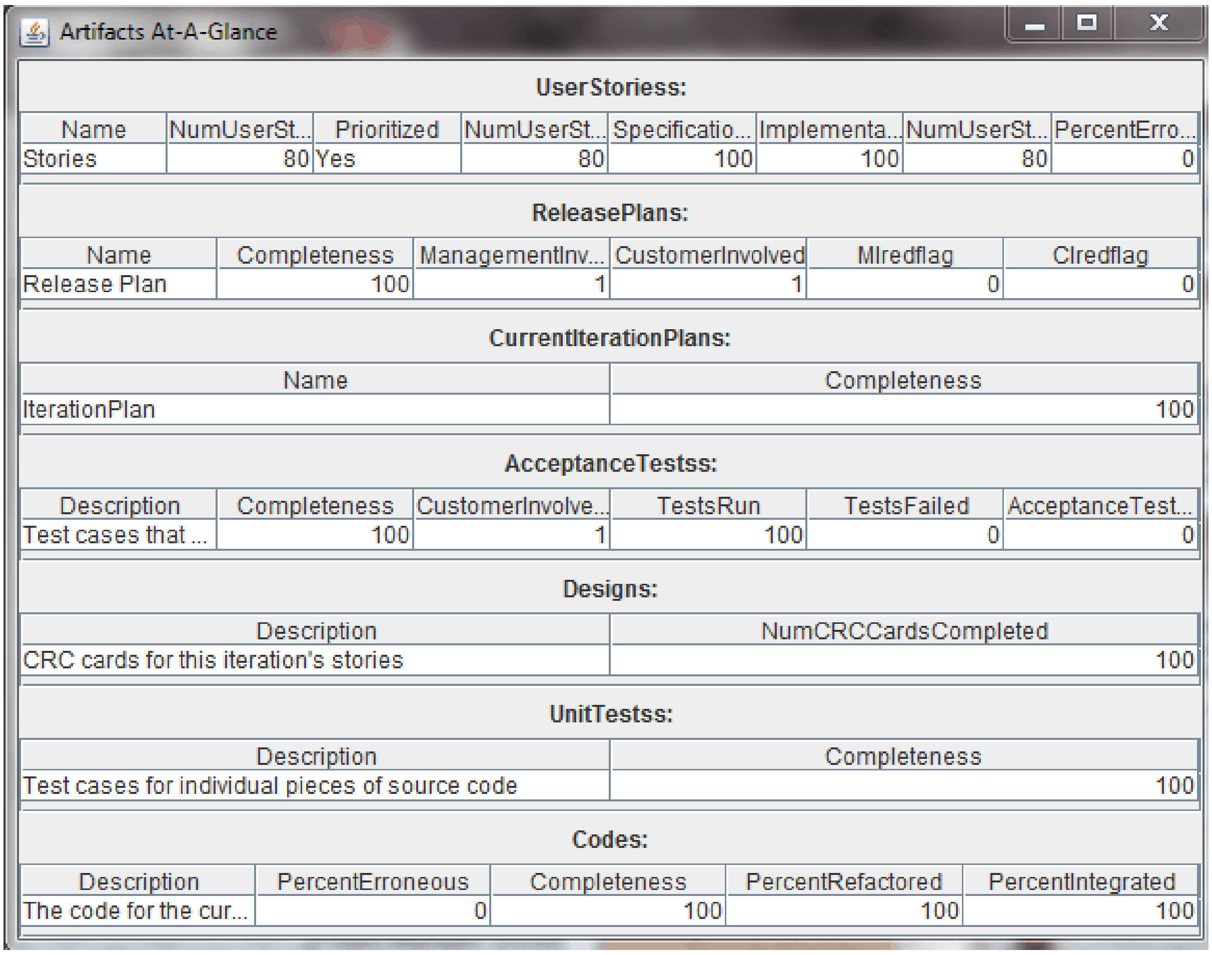
Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curr...	0	0	0	0

Figure 78: Final state of the artefacts for game1

For Game2, release 1 was completed at clock tick 533; release 2 at 946; release 3 at 1359 and release 4 at 1772. Customer never complained throughout the game.

	Release 1	Release 2	Release 3	Release 4
Game2 – Integrated EWS XP game	533	946	1359	1772

Table 27: Clock ticks representing the completion of releases for game2



UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementa...	NumUserSt...	PercentErro...
Stories	80	Yes	80	100	100	80	0

ReleasePlans:

Name	Completeness	ManagementInv...	CustomerInvolved	MIredflag	CIredflag
Release Plan	100	1	1	0	0

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	CustomerInvolve...	TestsRun	TestsFailed	AcceptanceTest...
Test cases that ...	100	1	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the cur...	0	100	100	100

Figure 79: Final state of the artefacts for game2

For player 10, there was an apparent improvement from game 1 to game 2. In game 1, the user was able to finish two iterations and was in iteration 3 when the game ended whereas in game 2, the user finished all the four iterations. The fourth release was finished at clock tick 1772 which was before the deadline of

1800 clock ticks. In game 1, the implementation completeness was 50% and the total number of user stories integrated was 40. In game 2, the implementation completeness was 100% and the total number of user stories integrated was 80. In game 1, the total number of times the customer complained was 14 whereas in game 2, the customer never complained. Game 2 was similar to the best case as defined in earlier sections.

4.6.2.3 Analysis and Result

Now having discussed the individual player's results, this section will compare and analyse the results between control group and EWS group and summarise the findings. The tables below provide the summary of the time taken by different control and EWS group players for their releases.

	Release 1	Release 2	Release 3	Release 4
Player 1 - Game 1	685	1098	1639	-
Player 1 - Game 2	552	994	1416	-
Player 2 - Game 1	1270	-	-	-
Player 2 - Game 2	1020	1478	-	-
Player 3 - Game 1	738	1267	1742	-
Player 3 - Game 2	725	1328	-	-
Player 4 - Game 1	1329	-	-	-
Player 4 - Game 2	1159	-	-	-
Player 5 - Game 1	1138	-	-	-
Player 5 - Game 2	1113	-	-	-

Table 28: Summary of time for releases by control group players

	Release 1	Release 2	Release 3	Release 4
Player 6 - Game 1	830	1708	-	-
Player 6 - Game 2	739	1231	1672	-
Player 7 - Game 1	706	1276	-	-
Player 7 - Game 2	540	958	1376	1790
Player 8 - Game 1	1349	-	-	-
Player 8 - Game 2	666	1154	1642	-

Player 9 - Game 1	1557	-	-	-
Player 9 - Game 2	835	1322	-	-
Player 10 - Game 1	1180	1667	-	-
Player 10 - Game 2	533	946	1359	1772

Table 29: Summary of time for releases by EWS group players

The average improvement in the number of iterations/releases completed between the two games by the control group players is as shown in Table 30:

	Number of iterations/releases completed in Game 1	Number of iterations/releases completed in Game 2	Improvement from Game 1 to Game 2
Player 1	3	3	0
Player 2	1	2	+1
Player 3	3	2	-1
Player 4	1	1	0
Player 5	1	1	0

Table 30: Improvement in the number of iterations/releases completed between the two games by the Control Group Players

Average improvement = (0 + 1 – 1 + 0 + 0) / 5 = **0**

That means for control group players on an average there is no improvement in the number of iterations/releases completed between the two games.

The average improvement in the number of iterations/releases completed between the two games by the EWS group players is as shown in Table 31:

	Number of iterations/releases completed in Game 1	Number of iterations/releases completed in Game 2	Improvement from Game 1 to Game 2
Player 6	2	3	+1
Player 7	2	4	+2
Player 8	1	3	+2
Player 9	1	2	+1
Player 10	2	4	+2

Table 31: Improvement in the number of iterations/releases completed between the two games by the EWS Group Players

Average improvement = (1 + 2 + 2 + 1 + 2) / 5 = **1.6**

That means for EWS group players on an average there is an improvement of 1.6 iterations/releases completed between the two games.

This is showing that on average, the players in the EWS group successfully finished **1.6** more iterations/releases than players in the control group.

The second point of comparison is the completeness and correctness of the whole code (or product). The final state of the artefacts for each game represents the completeness and correctness of the product. The code or product is considered to be fully complete if all the 4 iterations are fully completed. A typical illustration of it is the game 2 of player 10 where the entire code is correctly completed and the product is delivered to the customer (that too before the deadline).

In terms of statistics there are many useful attributes from the list of artefacts which can be used to compare the improvement of control group and EWS group. One such useful absolute attribute which can be used as a measure is "Number of User Stories Integrated". The "Percent Erroneous" attribute in the UserStories artifact has also been taken into account in conjunction with "Number of User Stories Integrated" attribute to address both "completeness" and "correctness" of the code. The following formula has been used:

Score = ((100 – PercentErroneous) * NumUserStoriesIntegrated)/100

For the control group players, the change in number of correctly integrated user stories taking into account the percentage error between the two games is as shown in the following tables:

	User Stories Integrated in Game 1	**Percent Erroneous in Game 1**	**User Stories Integrated in Game 2**	**Percent Erroneous in Game 2**
Player 1	60	0	80	8
Player 2	40	10	40	0

Player 3	60	0	60	1
Player 4	40	49	20	0
Player 5	20	0	20	0

Table 32: Comparison of User Stories Integrated and Percent Erroneous for the Control Group Players

	Score (Game 1)	Score (Game 2)	Improvement in the number of User Stories Integrated correctly	% Improvement in the number of User Stories Integrated correctly
Player 1	60	73.6	+13.6	22.7%
Player 2	36	40	+4	11.1%
Player 3	60	59.4	-0.6	-1.0%
Player 4	20.4	20	-0.4	-2.0%
Player 5	20	20	0	0.0%

Table 33: Improvement in the number of User Stories Integrated correctly between the two games for the Control Group Players

Average improvement = (13.6 + 4 – 0.6 – 0.4 + 0) / 5 = **3.3**

That means for control group players on an average there is increase of 3.3 correctly integrated user stories from game 1 to game 2.

For EWS group players, the change in number of correctly integrated user stories taking into account the percentage error between the two games is as shown in the following tables:

	User Stories Integrated in Game 1	Percent Erroneous in Game 1	User Stories Integrated in Game 2	Percent Erroneous in Game 2
Player 6	40	0	60	0
Player 7	60	21	80	0
Player 8	21	4	60	0
Player 9	20	0	60	0
Player 10	40	0	80	0

Table 34: Comparison of User Stories Integrated and Percent Erroneous for the EWS Group Players

	Score (Game 1)	Score (Game 2)	Improvement in the number of User Stories Integrated correctly	% Improvement in the number of User Stories Integrated correctly
Player 6	40	60	+20	50%
Player 7	47.40	80	+32.6	68.8%
Player 8	20.16	60	+39.84	197.6%
Player 9	20	60	+40	200.0%
Player 10	40	80	+40	100.0%

Table 35: Improvement in the number of User Stories Integrated correctly between the two games for the EWS Group Players

Average improvement = (20 + 32.6 + 39.84 + 40 + 40) / 5 = **34.5**

That means for EWS group players on an average there is increase of 34.5 correctly integrated user stories from game 1 to game 2.

This is showing that on average, the players in the EWS group successfully correctly integrated **31.2** more user stories than players in the control group.

A similar statistical analysis can also be done using other attributes. However, it should be taken in to account that unlike NumUserStoriesIntegrated, some of these attributes are relative and is based on current iteration. That means, for example there can’t be a direct comparison between the UnitTest.Completeness for iteration 2 and UnitTest.Completeness for iteration 3 - there is a gap of an entire iteration between them; unless the change in the iteration has been somehow included in the statistical analysis. If the primary measure of user stories integrated is inconclusive then these attributes can help to assess the progress in the incomplete iteration. However, the primary measure in this case (user stories integrated in conjunction with percent erroneous) is very conclusive and hence the detailed analysis of these attributes has not been shown.

The third point of comparison is the number of times the customer complains.

	Customer Complains
Player 1 - Game 1	6
Player 1 - Game 2	5
Player 2 - Game 1	16
Player 2 - Game 2	11
Player 3 - Game 1	7
Player 3 - Game 2	9
Player 4 - Game 1	17
Player 4 - Game 2	18
Player 5 - Game 1	16
Player 5 - Game 2	16

Table 36: Summary of number of the customer complains for control group players

	Customer Complains
Player 6 - Game 1	15
Player 6 - Game 2	7
Player 7 - Game 1	8
Player 7 - Game 2	-
Player 8 - Game 1	17
Player 8 - Game 2	7
Player 9 - Game 1	20
Player 9 - Game 2	9
Player 10 - Game 1	14
Player 10 - Game 2	-

Table 37: Summary of number of the customer complains for EWS group players

The average improvement in the number of customer complains between the two games by the control group players are as follows:

	Customer complains in Game 1	Customer complains in Game 1	Improvement from Game 1 to Game 2
Player 1	6	5	+1
Player 2	16	11	+5
Player 3	7	9	-2

Player 4	17	18	-1
Player 5	16	16	0

Table 38: Customer complains in the two games for the Control Group Players

Average improvement = (1 + 5- 2 - 1 + 0) / 5 = **0.6**

That means for control group players on an average there is an improvement of 0.6 number of customer complains between the two games.

The average improvement in the number of customer complains between the two games by the EWS group players are as follows:

	Customer complains in Game 1	Customer complains in Game 1	Improvement from Game 1 to Game 2
Player 6	15	7	+8
Player 7	8	0	+8
Player 8	17	7	+10
Player 9	20	9	+11
Player 10	14	0	+14

Table 39: Customer complains in the two games for the EWS Group Players

Average improvement = (8 + 8 + 10 + 11 + 14) / 5 = **10.2**

That means for EWS group players on an average there is an improvement of 10.2 number of customer complains between the two games.

This is showing that on average, the players in the EWS group got **9.6** fewer customer complains than players in the control group.

To summarise the findings:

- For the control group players there wasn't always a clear progression from game 1 to game 2. For players 1, 2 and 5 the performance improved slightly in their second game whereas the performance of players 3 and 4 slightly deteriorated in their second game. I would tend to say that for all the control group players their performance in second game either marginally improved or marginally deteriorated if compared to game 1. Player 2 was just about to finish iteration 2 in game 1. He got a slight push and hence managed to finish iteration 2 and got into iteration 3 in game 2. Evidence of progression or improvement is not conclusive for the control group players.

- For all the EWS group players there was a very clear progression from game 1 to game 2. Their performance improvement was evident. In fact player 7 and player 10 were able to complete all the four iterations before the deadline of 1800 clock ticks. This improvement pattern can also be noticed in the total number of customer complains. It appears that EWS system has really added value and has positive impact on the project outcomes.

Further, on the basis of the statistical analysis it has been shown that the EWS group showed a greater improvement than the control group. Just to re-iterate, the control group played the normal XP game twice whereas the EWS group played the normal XP game once followed by the integrated EWS XP game. The purpose of the control group is therefore to provide a baseline of comparison from which any implicit bias that can be attributed to playing the game (and "learning the rules") is removed so that any benefit of the EWS approach can be appropriately quantified. In the first evaluation parameter, for the control group players on an average there was no improvement in the number of iterations/releases completed between the two games whereas for the EWS group players the average was 1.6. This is showing that on average, the players in the EWS group successfully finished 1.6 more iterations/releases than players in the control group. In the second evaluation parameter, for the control group players the average change in the number of correctly integrated user stories taking into account the percentage error between the two games was 3.3 whereas for the EWS group players it was 34.5 (which is 31.2 more correctly integrated user stories than players in the control group). Similarly, in the third evaluation parameter, for the control group players the average improvement in the number of customer complains between the two games was 0.6 compared 10.2 for the EWS group players. On an average the players in the EWS group got 9.6 fewer customer complains than players in the control group.

Having said that, author acknowledges the limitations of the use of simple statistics. One of the original research goals of this study is to simply look for the evidence to support further study. Given the goal, this simple approach is appropriate and hence the author hasn’t looked too much into the statistical significance. To add more, the author also acknowledges the limitations of the sample size. The sample size is really not large enough to make any generalisations. Also, as said earlier all the early warning signs were not addressed by the game (only 6 of them were addressed). So the obvious "further work" comments are to do with developing some form of recommender system outside of the SimSE environment and testing it alongside a real project. But these results could definitely act as a conceptual proof.

4.7 Summary

This section concludes the results and analysis chapter which has presented the early warning signs implemented in this research. Information about SimSE simulation tool was also provided. SimSE is an interactive and graphical computer based environment which allows the creation and simulation of software engineering processes (Navarro and Hoek, 2004). It allows simulating realistic software engineering environment comprised of real world components like people, budget, large complex projects, planning, deadlines, unexpected events and others (Navarro and Hoek, 2007). It is designed specifically for providing a platform to experience different aspects of software engineering in a practical manner. CT and controlled experimental setup was explicated in detail followed by their results. Finally, the results were analysed, interpreted and evaluated. The following chapter will conclude the thesis and discuss the research findings and answers the research questions. It will also talk about the limitations of the study.

Chapter V
Discussion and Conclusion

5.1 Introduction

This thesis has demonstrated the application of a new and innovative concept known as early warning signs in the context of software project management. Employing this concept in project management field could improve the project success rate. This chapter will conclude the thesis and discuss the research findings and answers the research questions. Further, it also outlines the limitations of the study and recommendations for future research.

5.2 Answers to the Research Questions

As stated in chapter 1, following are the research questions for this thesis:

- Does the implementation of early warning phenomenon have positive effects on the project outcomes?
- Is there positive impact on the project outcomes if the corrective actions are taken early?

To answer these questions SDRM methodology was employed. SimSE simulation tool was used to design the models which were tested using two experimental techniques: Individual EWS testing and Controlled Experimental Study. Experiments were conducted which contributed data to this aspect. These data was analysed and evaluated. The results were then discussed and the answers for the research questions were acquired.

For research question one:

Does the implementation of early warning phenomenon have positive effects on the project outcomes?
From the results of controlled experimental study we have seen that for all the EWS group players, performance has improved in their second game. In fact their performance improvement was marked. For all the control group players this wasn’t true. In some of the cases their performance deteriorated in their second game. It appears that EWS system has really added value and has positive effects on the project outcomes. Hence the answer to the research question is: **Yes**, the implementation of early warning phenomenon has positive effects on the project outcomes.

For research question two:

Is there positive impact on the project outcomes if the corrective actions are taken early?
From the results of individual EWS testing it is evident that sooner we apply corrective actions the better it is for the projects. This was true for all the models (apart from last model (EWS 6) where this test was not applicable and we had no data generated). Hence the answer to the research question is: **Yes**, there is a positive impact on the project outcomes if the corrective actions are applied early. However, it is important to recognise the limitations of the approach used to generate the notification. In a real world problem it is unlikely that there will be a clear indication that something has gone wrong. Rather than a binary change in a project element, the early warning sign is likely to grow more

apparent over time. Therefore it is important to not overstate the contribution of this research.

5.3 Limitations and Recommendations

This research is a stepping stone to head towards a proactive project management style. It is preliminary and exploratory in nature. Even though the findings of the research are encouraging there were some limitations. But these results could definitely act as a conceptual proof and renders as an important piece of research upon which further hypotheses can be based.

Not all the important early warning signs were addressed by the research mainly due to the limitations of the SimSE tool and time constraints. But the idea was that if we can demonstrate and prove the objective using 6 EWS then it should be scalable and also work with more EWS (though it might add some complexity). Implementation of more EWS is something which should be considered as future work for this research.

Further, the models evaluated in this research were preliminary models. A major limitation of the study is that it is based on the arbitrary rules in the SimSE game. There were few assumptions and known functionality limitations of SimSE (like for instance sometimes you can’t stop a task once it is allocated). One of the assumptions was that the SimSE XP model would ignore the budget and only focus on the schedule of the project. Hence the SimSE models used in the research may not completely represent the real-world project situation. There were many technical challenges during the implementation of the SimSE models. Also, the model was simulating XP software development methodology and few users had low experience in XP methodology. Therefore, the results of this research were constrained by these parameters.

As stated earlier, the author acknowledges the limitations of the sample size. The controlled experimental study was performed only using 10 volunteers. The sample size is really not large enough to make any generalisations. It would also be interesting to observe how helpful the early warning sign concept is to volunteers of different PM experience. Maybe volunteers with low PM experience would find this recommender system more useful than volunteers with high PM experience. However, that means the sample size should be large and diverse enough to make any generalisations (or smaller but targeted to a particular group of project managers).

Also, another obvious further work comments are to do with developing some form of recommender system outside of the SimSE environment and testing it alongside a real project.

Again as stated earlier, it is important to recognise the limitations of the approach used to generate the notification. In a real world problem it is unlikely that there will be a clear indication that something has gone wrong. Rather than a binary change in a project element, the early warning sign is likely to grow more apparent over time. Therefore it is important to not overstate the contribution of this research.

The author also acknowledges the use of simple statistical analysis like average while evaluating and comparing the improvement of the control group and the EWS group in this research.

This research is first of its kind as no-one has ever empirically demonstrated the concept of early warning signs in a typical IT project environment. Having said that, Nikander has done considerable work in this field but his work (or models) was not empirically demonstrated. This is a wide open research area and has many future work options. To name a few the decision support model of early warnings by Nikander (2002) needs to be empirically demonstrated. The current research has extended his decision support model but hasn’t demonstrated all the concepts presented by the model. A further, detailed empirical study is needed to demonstrate his decision support model.

Another future work option is to construct a framework so that early warning signs phenomenon can be embedded in to project risk management. We already have an unproven reference model by Nikander (2002) as shown in figure 80. Empirically proving this model would definitely be next stepping stone in this field as that would increase organisation's confidence on early warning signs phenomenon and would practically provide a way for organisations to implement this phenomenon with their risk management strategies.

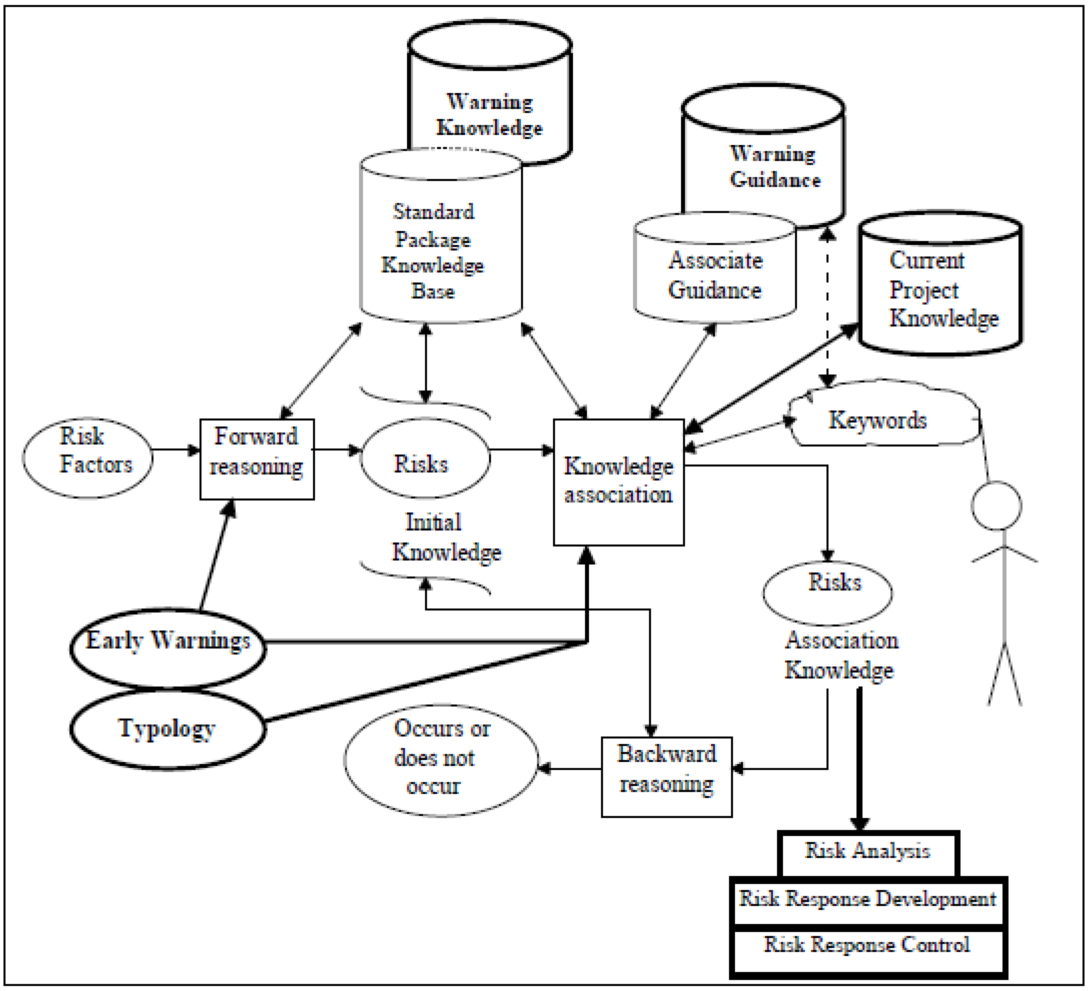


Figure 80: Embedding of Early Warning Sign phenomenon with Project Risk Management (Nikander, 2002)

Another future recommendation work would be, to provide definite mapping between early warning signs and their corrective actions. This could act as a recommender system. All the EWS implemented in this research had only one corrective action but in real project scenario this is unlikely. For example, for EWS "Functional, performance, and reliability requirements and scope are not documented", we could have following corrective actions (prescriptions) (Havelka and Rajkumar, 2006):

- Define the scope of the project and specify boundaries
- Review project specifications and requirements with sponsor and users, obtain signoffs
- Review and update project management artefacts e.g. the WBS
- Determine specific problem, if necessary
- Review project specifications and requirements with analysis, design, coding and testing personnel

Determining the priority of these corrective actions depending on their effectiveness and recommending to the user would be useful for a recommender system. This could then lead to a learning system which self-adapts for an individual company so that the appropriate prescriptions are tailored to their own work environment.

5.4 Conclusion

The software industry has been plagued by the staggering failure rate of projects, which have resulted in the loss of billions of dollars. The well known Chaos Report by the Standish Group declared that software projects are in chaos with only 16.2% of software projects actually being successful in the year 1994 (Eveleens and Verhoef, 2010). However, since that time it appears that IT project's success rate is improving, albeit slowly. According to the recent study by Standish group, 32% of the projects were successful in the year 2009 (Eveleens and Verhoef, 2010). Whilst the Chaos report has faced many critiques by several authors regarding the credibility, validity of figures, relevance and integrity (Eveleens and Verhoef, 2010; Robin and Goldsmith, 2007; Molokken, 2006, Glass, 2006), it provides some degree of evidence that developing software is still challenging today as it was at the time of writing of the Mythical Man Month (Brooks, 1995), though the reasons and their relative impact may have changed in the intervening period.

Managing software development projects is certainly challenging and difficult. They have inherent characteristics like human interaction, abstraction, complexity and uncertainty. There is no out-of-the-box or off-the-shelf solutions for real-life software development projects but one way to save projects is to deal with uncertainties.

In this research, we have identified different approaches to deal with uncertainties starting from requirements engineering, software process management to project risk management. Project management strategies are critical to the enterprise success. This study has proposed new and innovative concept of early warning signs which can be embedded in to project risk management. The project risk management theories are not closely integrated with the early warning phenomenon but this can apparently be utilised as a tool in project risk management (Nikander, 2002).

The early warning signs concept is grounded on the thought that failures do not happen overnight and that well run software projects root out problems early. A warning sign is an indication or an event that predicts or alerts impending problem(s) (Kappelman et al., 2006). Early warning sign provide an indication of manifesting risks.

But the flipside is that the whole concept of early warnings phenomenon is complex and its scope is very broad. It has multifarious dimensions. These early warnings could emerge in numerous forms and can be detected in several different situations and can be interpreted in various ways (Nikander, 2002). Further, it is influenced by many internal and external factors. Also, it is fully dependent on the observer/receiver who is interpreting the information and hence the results/decision making depends completely on him.

In spite of these challenges the idea looks promising. As of yet there was no empirical evidence that proves that implementing early warnings phenomenon may have positive effects on the project outcomes. The aim of this research work was to provide some empirical evidence that there is a value in following and implementing early warnings phenomenon. A further aim of this research work was to provide some empirical evidence that there is a value in terms of undertaking corrective actions early by demonstrating that if we delay corrective actions then it has a negative impact on the project outcomes. These research objectives led to the following two research questions which were answered within this research:

- Does the implementation of early warning phenomenon have positive effects on the project outcomes?
- Is there positive impact on the project outcomes if the corrective actions are taken early?

To answer these questions SDRM methodology was employed. SimSE simulation tool was used to design the models which were tested using two experimental techniques: Individual EWS testing and Controlled Experimental Study. Experiments were conducted which contributed data to this aspect. These data was analysed and evaluated. The results were then discussed and positive answers for the research questions were acquired.

The study found that the implementation of early warning phenomenon has positive effects on the project outcomes. Also, there is a positive impact on the project outcomes if the corrective actions are taken early.

This study is just one step in this direction and has introduced a new concept of early warning signs to the research arena. This is a wide domain and needs to be built on in several directions. This research has contributed in software engineering field by introducing early warning phenomenon to a typical IT project environment. In spite of many challenges, the author has empirically proved that the implementation of early warning phenomenon has positive effects on the project outcomes. It is believed that this head start in introducing early warning concept to project environment will provide the basis for next generation proactive project risk management strategies.

Finally, the author would like to conclude with the statement that "*Paying attention to early warning signs can improve the chances of project's success. Just as we notice the warning lights and gauges on the dashboards of our* *automobiles, paying attention to early warning signs during our project journey can help us to avoid problems and successfully reach our destinations*" (Kappelman et al., 2006).

References

Aalst, V. D. (1998). The Application of Petri Nets to Workflow Management. *Journal of Circuits Systems and Computers*.

Aberg, L. (1989). *The Facet Theory of Communication,* University of Helsinki, Department of Communications, Helsinki, Finland.

Abrahamsson, P., Warsta, J., Siponen, M. T., &Ronkainen, J. (2003). *New directions on agile methods: a comparative analysis.* Paper presented at the Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon.

Aguilar-Ruiz, J. S., Riquelme, J. C., Rodríguez, D., & Ramos, I. (2002). *Generation of Management Rules through System Dynamics and Evolutionary Computation.* Paper presented at the Proceedings of the 4th International Conference on Product Focused Software Process Improvement, Rovaniemi, Finland.

Al-Emran, A., Pfahl, D., &Ruhe, G. (2008). A method for re-planning of software releases using discrete-event simulation. *Software Process: Improvement and Practice, 13*(1), 19 - 33.

Anderson, D. J. (2004). The Four Roles of Agile Management. *Cutter IT Journal.*

Ansoff, I. H. (1975). Managing Strategic Surprise by Response to Weak Signals. *California Management Review, 17*(2).

Ansoff, I. H. (1984). *Implanting Strategic Management*. New York, USA: Prentice/Hall International Inc.

Ashley, D. B. (1989). *Project Risk Identification Using Inference Subjective Expert Assessment and Historical Data.* Paper presented at the Proceedings of the International Project Management Association.

Balandis, O., &Laurinskaite, L. (2005). Software Process Improvement in Lithuania - UAB Sintagma Case Study. *Information Technology and Control, 34*(2A), 195 - 201.

Betts, R. K. (1982). *Surprise Attack, Lessons for Defense Planning*. Washington D.C., USA: The Brookings Institution.

Beznosov, K., &Kruchten, P. (2004). *Towards agile security assurance.* Paper presented at the Proceedings of the New Security Paradigms Workshop, Nova Scotia, Canada.

Bishop, M. (2009). 2009 Standish Group CHAOS Report: Worst Project Failure Rate in a Decade. Retrieved March 3, 2010, from http://www.irise.com/blog/index.php/tag/chaos-report/

Bloodgood, J. M., & Salisbury, W. D. (2001). Understanding the influence of organizational change strategies on information technology and knowledge management strategies. *Decision Support Systems, 31*(1), 55 - 69.

Boehm, B. (2002). Get Ready for Agile Methods, with Care. *Computer, 35*(1), 64 - 69.

Boehm, B., Abts, C., &Chulani, S. (2000). Software development cost estimation approaches - A survey. *Annals of Software Engineering, 10*, 177-205.

Boehm, B., & Turner, R. (2003). Using risk to balance agile and plan-driven methods. *IEEE Journal, 36*(6), 57 - 66.

Bose, I. (2008). Lessons Learned from Distributed Agile Software Projects: A Case-Based Analysis. *Communications of the Association for Information Systems, 23*(1), 619 - 632.

Brooks, F.P. (1995). *The Mythical Man-Month: Essays on Software Engineering.* Boston, MA, USA: Addison Wesley Longman.

By, R. T. (2005). Organisational change management: A critical review. *Journal of Change Management, 5*(4), 369 - 380.

Chang, C.K., Jiang, H., Di, Y., Zhu, D., &Ge, Y. (2008). Time-line based model for software project scheduling with genetic algorithms. *Information and Software Technology, 50,* 1142-1154.

Charette, R. N. (2005). Why software fails. *IEEE Spectrum, 42*(9), 42-49.

Christie, A. M. (1999). Simulation in support of CMM-based process improvement. *The Journal of Systems and Software, 46*, 107 - 112.

Clarkson, P. J., & Hamilton, J. R. (2000). ‘Signposting’, A Parameter-driven Task-based Model of the Design Process. *Research in Engineering Design, 12*(1), 18-38.

Clarkson, P. J., Melo, A., & Connor, A. M. (2000). *Signposting for Design Process Improvement: A Dynamic Approach to Design Process Planning.* Paper presented at the Aritificial Intelligence in Design. 333 - 353.

Clarkson, P. J., Melo, A., & Eckert, C. (2000). *Visualization of Routes in Design Process Planning.* Paper presented at the Fourth International Conference on Information Visualisation.

Connor, A. M. (2007). Probabilistic estimation of software project duration. *New Zealand Journal of Applied Computing & Information Technology, 11*(1), 11-22.

Connor, A., &MacDonell, S. (2006). *Using historical data in stochastic estimation of software project duration*. Paper presented at the 19th Annual Conference of the National Advisory Committee on Computing Qualifications.

Conrow, E.H., &Shishido, P.S. (1997). Implementing Risk Management on Software Intensive Projects. *IEEE Software,* 83-89.

Creasey, T. (2007). *Defining change management*: Prosci and the Change Management Learning Center.

Crosby, D. A. K. (2007). *Project Risk Management in Smaller Software Teams.* Unpublished Master's thesis, Auckland University of Technology, Auckland, New Zealand.

Dominguez, J. (2009). The Curious Case of the CHAOS Report 2009. Retrieved from http://www.projectsmart.co.uk/the-curious-case-of-the-chaos-report-2009.html

Eppinger, S. D., Whitney, D. E., Smith, R. P., &Gebala, D. A. (1994). A model-based method for organizing tasks in product development. *Research in Engineering Design, 6*(1), 1-13.

Eveleens, J. L., &Verhoef, C. (2010). The Rise and Fall of the Chaos Report Figures. *IEEE Software, 27*(1), 30 - 36.

Ferreira, S., Collofello, J., Shunk, D., &Mackulak, G. (2009). Utilization of Process Modeling and Simulation in Understanding the Effects of Requirements Volatility in Software Development. *Journal of Systems and Software, 82*(10), 1568 -1577.

Finkelstein, A., Kramer, J., &Nuseibeh, B. (1994). *Software Process Modelling and Technology*. New York, USA: John Wiley & Sons, Inc.

Garvey, P. R., Phair, D. J., & Wilson, J. A. (1997). An Information Architecture for Risk Assessment and Management. *IEEE Software, 14*(3), 25 - 37.

Glass, R. L. (2006). The Standish Report: Does It Really Describe a Software Crisis? *COMMUNICATIONS OF THE ACM, 49*(8), 15 - 16.

Halvorsen, C. P., &Conradi, R. (2001). *A Taxonomy to Compare SPI Frameworks.* Paper presented at the Proceedings of the 8th European Workshop on Software Process Technology, Witten, Germany.

Havelka, D., &Rajkumar, T. M. (2006). Using the Troubled Project Recovery Framework: Problem Recognition and Decision to Recover. *e-Service Journal*.

Heeks, R. (2003). *Most e-Government-for-Development Projects Fail How Can Risks be Reduced?* Manchester, UK: Institute for Development Policy and Management.

Heemstra, F. J. (1990). *Software cost estimation models.* Paper presented at the Jerusalem Conference on Information Technology, Jerusalem, Israel.

Henderson, P., Howard, Y.M., &Walters,R.J. (2001). A tool for evaluation of the software development process. *The Journal of Systems and Software, 59,* 355-362.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in InformationSystems Research. *MIS Quarterly, 28*(1), 75-105.

Highsmith, J. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems.* New York: Dorset House Publishing

Hiltunen, E. (2008). The future sign and its three dimensions. *Futures, 40*(3), 247- 260.

Hofmann, H. F., &Lehner, F. (2001). Requirements Engineering as a Success Factor in Software Projects. *IEEE Software, 18*(4), 58 - 66.

Ilamola, L., &Katsola-Mustonen, A. (2003). Filters in the Strategy Formulation Process. *Journal of Universal Computer Science, 9*(6), 481 - 490.

Ilmola, L., &Kotsalo-Mustonen, A. (2004). Filters of weak signals hinder foresight: Monitoring weak signals efficiently in corporate decision-making. *Futures*.

Jiang, J., & Klein, G. (2000). Software development risks to project effectiveness. *Journal of Systems and Software, 52*(1), 3 - 10.

Jones, C. (2004). Software Project Management Practices: Failure Versus Success. *Project Management*.

Jørgensen, M. (2004). A review of studies on expert estimation of software development effort. *Journal of Systems and Software, 70*(1-2), 37-60.

Jørgensen, M., &Moløkken, K. (2006). How Large Are Software Cost Overruns? A Review of the 1994 CHAOS Report. *Information and Software Technology, 48*(4).

Jörgensen, M., &Shepperd, M. (2007). A Systematic Review of Software Development Cost Estimation Studies. *IEEE Transactions on Software Engineering, 33*(1), 33-53.

Juran, J.M. (1995). *Managerial Breakthrough. The Classic Book on Improving Management Performance* (Rev. ed.). USA: McGraw-Hill, Inc.

Jurison, J. (1999). Software Project Management: The Manager's view. *Communications of the Association for Information Systems, 2*(17), 1-57.

Kappelam, L.A., McKeeman, R and Zhang L. (2006), EARLY WARNING SIGNS OF IT PROJECT FAILURE: THE DOMINANT DOZEN, *Information Systems Management*, 31-36.

Kasanen, E., Lukka, K., &Siitonen, A. (1993). The constructive approach in management accounting research. *Journal of Management Accounting Research, 5*, 243.

Keil, M., Tiwana, A., & Bush, A. (2002). Reconciling user and project manager perceptions of IT project risk: a Delphi study. *Information Systems Journal, 12*, 103 -119.

Kellner, M. I., Madachy, R. J., &Raffo, D. M. (1999). Software Process Simulation Modeling: Why? What? How? *Journal of Systems and Software, 46*(2/3).

Kirk, D., &MacDonell, S. (2009). *A Simulation Framework to Support Software Project (Re)Planning.* Paper presented at the Proceedings of the 2009 35th Euromicro Conference on Software Engineering and Advanced Applications, Patras, Greece.

Kirk, D., &Tempero, E. (2006). *Identifying Risks in XP Projects through Process Modeling.* Paper presented at the Australian Software Engineering Conference, Sydney, Australia.

Kohavi, R., Henne, R. M., &Sommerfield, D. (2007). *Practical Guide to Controlled Experiments on the Web: Listen to Your Customers not to the HiPPO.* Paper presented at the Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, New York, USA.

Kuppuswami, S., Vivekanandan, K., Ramaswamy, P., & Rodrigues, P. (2003). The effects of individual XP practices on software development effort. *ACM SIGSOFT Software Engineering, 28*(6), 1 - 6.

Larman, C., &Basili, V. R. (2003). Iterative and Incremental Development: A Brief History. *IEEE Computer Society*, 47 - 56.

Lederer, A. L., & Prasad, J. (1993). SYSTEMS DEVELOPMENT AND COST ESTIMATING. *Information Systems Management, 10*(4), 37 - 41.

Leidecker, J. K., & Bruno, A. V. (1987). Critical Success Factor Analysis and the Strategy Development Process. *Strategic Planning and Management Handbook*, 333 - 351.

Limbu, D. K. (2008). *Contextual Information Retrieval from the WWW.* Unpublished Doctoral dissertation, Auckland University of Technology, Auckland, New Zealand.

Linberg, K. R. (1999). Software developer perceptions about software project failure: a case study. *The Journal of Systems and Software, 49*(2), 177 - 192.

Lonchamp, J. (1993). *A Structured Conceptual and Terminological Framework for Software Process Engineering.* Paper presented at the Second Int'l Conference on the Software Process, Berlin.

Long, L.D., &Ohsato, A. (2008). Fuzzy critical chain method for project scheduling under resource constraints and uncertainty. *International Journal of Project Management, 26,* 688- 698.

Lukka, K. (2003). The Constructive Research Approach. In: L. Ojala& O.P. Himola (Eds.). *Case Study Research in Logistics* (pp. 83-101). Publications of the Turku School of Economics and Business Administration.

MacCormack, A., &Verganti, R. (2003). Managing the Sources of Uncertainty: Matching Process and Context in Software Development. *The Journal of Product Innovation Management, 20*, 217 - 232.

MacDonell, S. G. (1994). Comparative review of functional complexity assessment methods for effort estimation. *Software Engineering Journal, 9*, 107-116.

Makridakis, S., &Heáu, D. (1987). The Evolution of Strategic Planning and Management. *Strategic Planning and Management Handbook*, 3 - 20.

Mans, R. S., Russell, N. C., Aalst, V. D., Moleman, A. J., & Bakker, P. J. (2009). *Schedule-Aware Workflow Management Systems.* Paper presented at the Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE09), Paris, France.

Martin, R. H., &Raffo, D. (2000). A Model of the Software Development Process Using Both Continuous and Discrete Models. *Software Process Improvement and Practice, 5*, 147 - 157.

Mauerkirchner, M. (2000). Decision Based Adaptive Model for Managing Software Development Projects. *Project Management System,* 475-485.

McConnell, S. (1998). *Software Project Survival Guide*. Redmond, WA, USA: Microsoft Press.

Meng, J., Su, S. Y. W., Lam, H., Helal, A., Xian, J., Liu, X., et al. (2006). DynaFlow: a dynamic inter-organisational workflow management system. *International Journal of Business Process Integration and Management, 1*(2), 101 - 115.

Miller, K. D., & Waller, H. G. (2003). Scenarios, Real Options and Integrated Risk Management. *Long Range Planning, 36*(1), 93 - 107.

Mintzberg, H. (1994). The Fall and Rise of Strategic Planning. *Harvard Business Review*, 107 - 114.

Moløkken, K., &Jörgensen, M. (2003). *A Review of Software Surveys on Software Effort Estimation.* Paper presented at the International Symposium on Empirical Software Engineering, Rome, Italy.

Morris, R. C. (1997). *Early warning indicators of corporate failure: A critical review of previous research and further empirical evidence*. UK: Ashgate.

Navarro, E. (2006). *SimSE: A Software Engineering Simulation Environment for Software Process Education.* Unpublished Doctoral Dissertation, University of California, Irvine.

Navarro, E.O.andHoek, A. (2004). *SimSE: An interactive simulation game for software engineering education.* Paper presented at the Proceedings of the 7th IASTED International Conference on Computers and Advanced Technology in Education, Kauai, Hawaii, USA.

Navarro, E.O.andHoek, A. (2007). *Comprehensive Evaluation of an Educational Software Engineering Simulation Environment.* Paper presented at the Proceedings of the 20th Conference on Software Engineering Education & Training, Dublin, Ireland.

Ncube, C., & Dean, J. C. (2002). *The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Components.* Paper presented at the Proceedings of the 1st International Conference on COTS-Based Software System, Orlando, FL.

Niazi, A., Dai, J. S., Balabani, S., &Seneviratne, L. (2006). Product Cost Estimation: Technique Classification and Methodology Review. *Journal of Manufacturing Science and Engineering, 128*(2), 563-575.

Nikander, I. O. (2002). *Early Warnings - A Phenomenon in Project Management.* Unpublished Doctoral dissertation, Helsinki University of Technology, Espoo, Finland.

Nikander, I. O., &Eloranta, E. (1997). Preliminary signals and early warnings in industrial investment projects. *International Journal of Project Management, 15*(6), 371 - 376.

Nikander, I. O., &Eloranta, E. (2001). Project management by early warnings. *International Journal of Project Management, 19*, 385 - 399.

Nunamaker, J. F., & Chen, M. (1990). *Systems development in information systemsresearch.* Paper presented at the Twenty-Third Annual Hawaii InternationalConference on System Sciences, Kailua-Kona, HI, USA.

Nyugen, T.N. (2006). A decision model for managing software development projects. *Information & Management, 43,* 63-75.

Nyugen, L., &Swatman, P.A. (2003). Managing the requirements engineering process. *Requirements Eng, 8, 55-68.*

Paetsch, F., Eberlein, A., & Maurer, F. (2003). *Requirements Engineering and Agile Software Development.* Paper presented at the Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Washington, DC.

Paulk, M. C. (2002). *Agile Methodologies and Process Discipline*. Pittsburgh, PA: Institute for Software Research - Carnegie Mellon University.

Pesic, M., Schonenberg, M., Sidorova, N., & Aalst, V. D. (2007). Constraint-Based Workflow Models: Change Made Easy. In R. Meersman& Z. Tari (Eds.), *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS* (Vol. 4803, pp. 77-94): Springer Berlin / Heidelberg.

Pinto, J. K., &Slevin, D. P. (1992). *Project Implementation Profile*. Tuxedo, NY: Xicom, Inc.

Project ManangementInstitute. (2000). *A Guide to the Project Management Body of Knowledge (PMBOK Guide).* USA: Project Management Institute Inc.

Pressman, R. S. (1997). *Software Engineering - A practitioner's approach* (4th ed.). New York: McGraw-Hill.

Rai, A., & Al-Hindi, H. (2000). The effects of development process modeling and task uncertainty on development quality performance. *Information & Management, 37,* 335-346.

Rainer, A., &Shepperd, M. (1999). *Re-Planning for a Successful Project Schedule.* Paper presented at the Proceedings of the 6th International Symposium on Software Metrics, Boca Raton, FL, USA.

Reel, J. S. (1999). Critical Success Factors In Software Projects. *IEEE Software, 16*(3), 18 - 23.

Robin, F., Goldsmith, JD. (2007). *REAL CHAOS, Two Wrongs May Make a Right*. Retrieved October 5, 2008, from http://www.gopromanagement.com/REAL%20CHAOS,%20Two%20Wrongs%20May%20Make%20a%20Right.pdf

Sadiq, W., &Orlowska, M. (2000). Analyzing Process Models using Graph Reduction Techniques. *Information Systems, 25*(2), 117- 134.

Schackmann, H., Jansen, M., Lischkowitz, C., &Lichter, H. (2009). *QMetric - A Metric Tool Suite for the Evaluation of Software Process Data*. Paper presented at the 31st International Conference on Software Engineering, ICSE 2009.

Schmidt, R., Lyytinen, K., Keil, M., &Cule, P. (2001). Identifying Software Projects Risks: An International Delphi Study. *Journal of Management Information Systems, 17*(4), 5 - 36.

Schoemaker, P. J. H., & Day, G. S. (2009). How to Make Sense of Weak Signals. *MIT Slogan Management Review, 50*(3), 81 - 89.

Sharp, H., Robinson, H., & Segal, J. (2004). *Customer collaboration: successes and challenges in practice systems.* (Technical Report No. TR2004/10): Computing Department, The Open University.

Siringoringo, W. S. (2006). *Minimum Cost Polygon Overlay with Rectangular Shape Stock Panels.* Unpublished Master's thesis, Auckland University of Technology, Auckland, New Zealand.

Srinivasan, R. (2002). Planning is the Key in Software Development. *IT People evolve* Retrieved January 20, 2009, from http://www.expressitpeople.com/20020218/management4.shtml

Standish Group International. (1995). *The Chaos Report*. Retrieved October 1, 2008, from http://www.standishgroup.com/sample_research/PDFpages/chaos1994.pdf

Steward, D. V. (1981). The Design structure system: A method for managing the design of complex systems. *IEEE Transactions on Engineering Management, 28*, 71-74.

Talbot, A. (2010). *A survey of requirements engineering practice in New Zealand.* Unpublished Master's thesis, Auckland University of Technology, Auckland, New Zealand.

Taylor, A. (2000). IT projects: sink or swim. *The Computer Bulletin, 42*(1), 24 - 26.

Tiwana, A., &Keil, M. (2004). The one-minute risk assessment tool. *COMMUNICATIONS OF THE ACM, 47*(11), 73 - 77.

Uskali, T. (2005). Paying Attention to Weak Signals. *Innovation Journalism, 2*(11).

Wallace, L., &Keil, M. (2004). SOFTWARE PROJECT RISKS AND THEIR EFFECT ON OUTCOMES. *COMMUNICATIONS OF THE ACM, 47*(4), 68 - 73.

Wallace, L., Keil, M., &Rai, A. (2004). How Software Project Risk Affects Project Performance: An Investigation of the Dimensions of Risk and an Exploratory Model. *Design Sciences, 35*(2), 289 - 321.

Webb, J. R. (1987). *An Evaluation of Igor Ansoff´s Theory of Weak Signal Management by Means of an Investigation and Forecast of Future Development in the Ophthalmic Laser Environment.* Unpublished Doctoral thesis, University of Strathclyde, United Kingdom.

Weschke, H. R. (1994). Early Warning Signs of Risk Companies. *Economic Development Review*, 66 - 67.

Williams, T. (2005). Assessing and moving on from the dominant project management discourse in the light of project overruns. *Engineering Management: IEEE Journal, 52*(4), 497 - 508.

Workflow Management Coalition. (1996). *Workflow Management Coalition Announces Initial Publication of Workflow Interoperability Specification*. Retrieved June 17, 2009, from http://www.highbeam.com/doc/1G1-18854980.html

Yeo, K. T. (2002). Critical failure factors in information system projects. *International Journal of Project Management, 20*, 241 - 246.

Appendices

Appendix A: Supplementary Figures and Tables

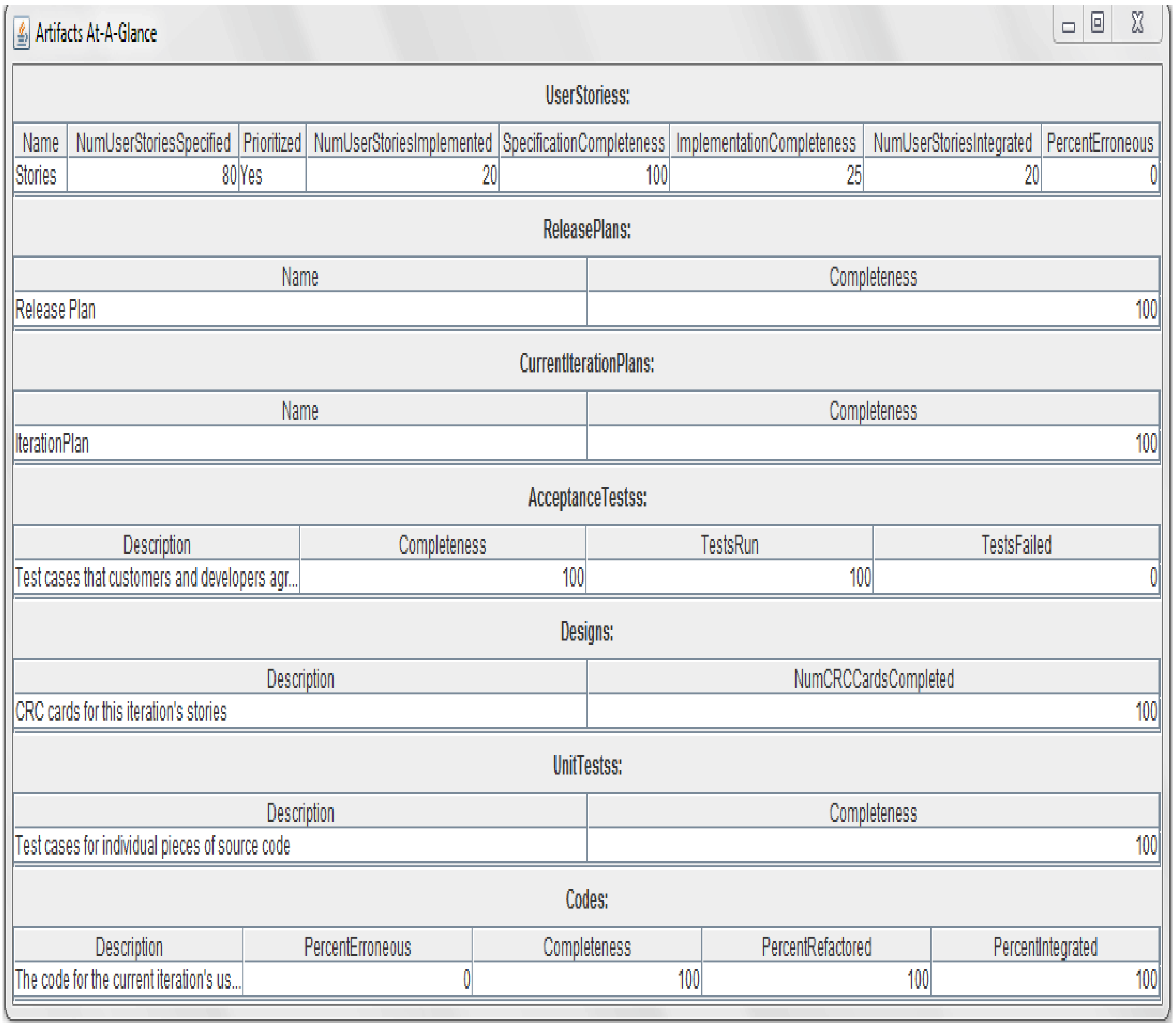
Appendix B: Experimental Results

Appendix C: Individual EWS Testing Results

Appendix A: Supplementary Figures and Tables

	Item Description	Source
1	Lack of top management support or commitment to the project	Schmidt et al., 2001
2	Functional, performance, and reliability requirements and scope are not documented	Winters, 2002
3	Project manager(s) cannot effectively lead the team and communicate with clients	Schmidt et al., 2001
4	No change control process	Schmidt et al., 2001
5	Project stakeholders have not been interviewed for project requirements	Ward, 2003
6	No documented milestone deliverables and due dates	
7	Undefined project success criteria	
8	Project team members have weak commitment to the project scope and schedule	Schmidt et al., 2001
9	Communication breakdown among project stakeholders	May, 1998
10	Key project stakeholders do not participate in major review meetings	
11	Project team members do not have required knowledge/skills	Barki et al., 2001
12	Project resources have been assigned to a higher priority project	Havelka et al., 2004
13	No business case for the project	Ward, 2003
14	No project status progress process	Havelka et al., 2004
15	Schedule deadline not reconciled to the project schedule	
16	Early project delays are ignored — no revision to the overall project schedule	McKeeman, 2001
17	Subject matter experts are overscheduled: retain all prior duties yet expected to provide substantial participation to the project	McKeeman, 2001
18	No planning and estimation documentation	Jones, 2004
19	Project managers have poor training	Schmidt et al., 2001
20	Key stakeholders do not review and sign off deliverables on a timely basis	
21	Project stakeholder decision delays have caused due dates to be missed	
22	No due diligence on vendor(s) and team members	McKeeman, 2001
23	No written commitment for the project outside of the project team	
24	Significant goal, scope, or schedule requirements change immediately after project kickoff	Boehm, 1991
25	Team members have undefined roles and responsibilities	Jiang et al., 2002
26	No project communications plan or resources devoted to managing and communicating project expectations	
27	Project team members are overscheduled	Schmidt et al., 2001
28	Users are not willing to cooperate	Schmidt et al., 2001
29	No team member experience with the chosen technology	Schmidt et al., 2001
30	No project management methodology	Schmidt et al., 2001
31	No project charter document at early stage of project	
32	No risk analysis documentation and process	McKeeman, 2001
33	Failure to gather requirement via joint application design	
34	No documented analysis of business strategy alignment	Winters, 2002
35	Major new risks are identified after the project kickoff	
36	No performance and reliability requirements metrics tracking process	Jones, 2004
37	Approved project budget less than budget estimated by the project team	
38	Budget, schedule, scope, and quality all mandated from outside the project team	
39	Project manager(s) have never managed a project of this scale before	McFarlan, 1982
40	Deliverable due dates missed during the first 10 percent of the project schedule	McKeeman, 2001
41	IT operations infrastructure and network infrastructure problems have major impact on project team productivity	
42	Difficulty in determining the input and output of the system	
43	Cultural conflict among organizations involved	Winters, 2002
44	No contingency budget for known risks and rate of changes	
45	Unstable organization environment (such as changes in senior management or restructuring)	Schmidt et al., 2001
46	Project team member(s) have low morale	McKeeman, 2001
47	Key team member turnover after project kickoff	Schmidt et al., 2001
48	Key stakeholders have not signed the project charter	
49	Large number of interfaces to other system required	Barki et al., 2001
50	Users cannot get involved because of lack of understanding of new system capabilities	McFarlan, 1982
51	Project involves implementing a custom or beta version of hardware or software	Schmidt et al., 2001
52	Users or technical support team feel threatened by a project to replace their legacy system	Jiang et al., 2002
53	Earned value systems not in place or used to control program	

Table A.1: Top 53 early warning signs (Kappelman, McKeeman and Zhang, 2006)



Artifacts At-A-Glance

UserStoriess:

Name	NumUserStoriesSpecified	Prioritized	NumUserStoriesImplemented	SpecificationCompleteness	ImplementationCompleteness	NumUserStoriesIntegrated	PercentErroneous
Stories	80	Yes	20	100	25	20	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers and developers agr...	100	100	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the current iteration's us...	0	100	100	100

Figure A.1: Artefacts at a glance

Artifact: User Stories

Attributes: Name, NumUserStoriesSpecified, Prioritized, NumUserStoriesImplemented,SpecificationCompleteness, ImplementationCompleteness, NumUserStoriesIntegrated and PercentErroneous

Artifact: Release Plans:

Attributes: Name and Completeness

Artifact: Current Iteration Plans:

Attributes: Name and Completeness

Artifact: Acceptance Tests

Attributes: Description, Completeness, TestsRun and TestsFailed

Artifact: Designs

Attributes: Description and NumCRCCardsCompleted

Artifact: Unit Tests

Attributes: Description and Completeness

Artifact: Codes

Attributes: Description, PercentErroneous, Completeness, PercentRefactored and PercentIntegrated

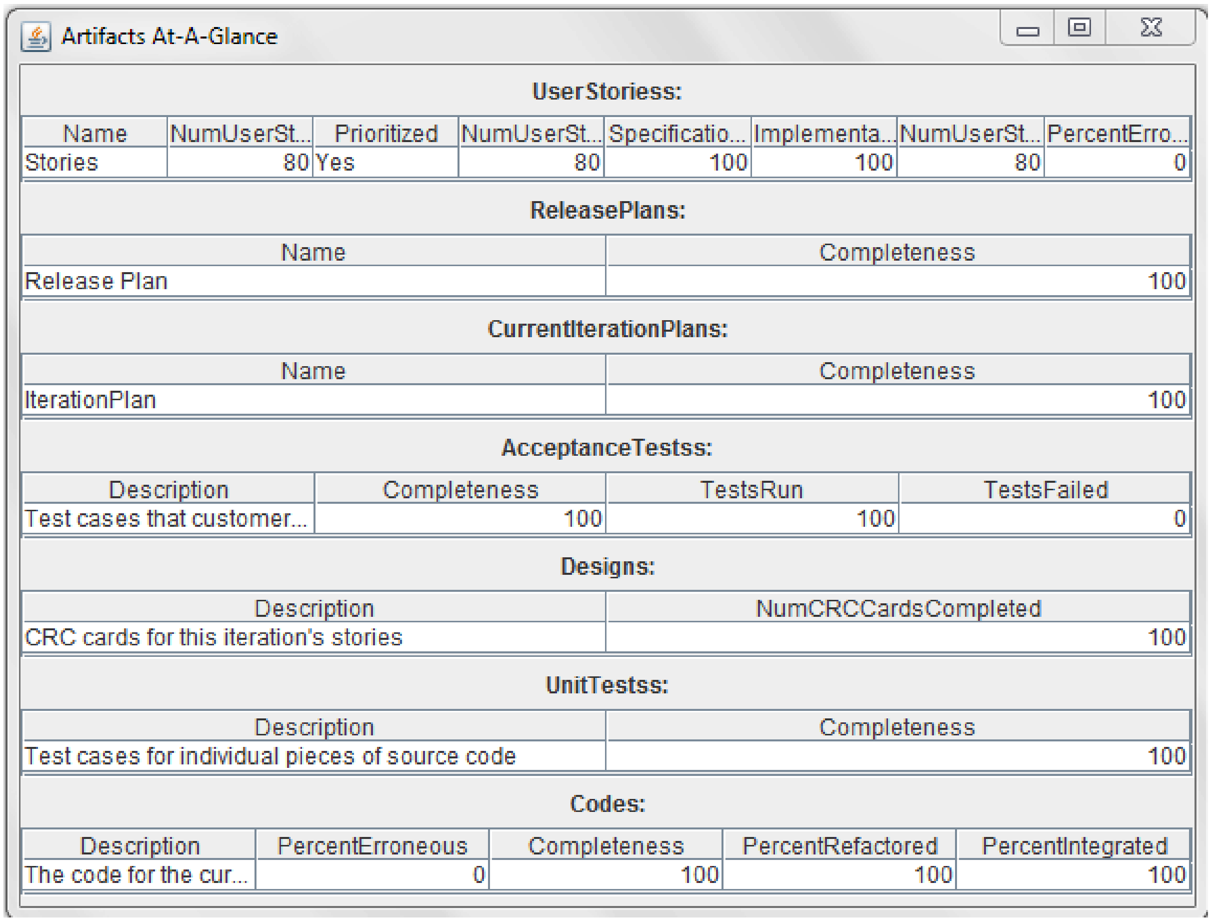
Appendix B: Experiment Results

B.1: Best Case

Best case represents an ideal project i.e. all the right actions are taken by the user and everything goes smoothly and the project is completed successfully before the deadline. As everything goes fine no warning message appears in the game. For the best case, release 1 (i.e. end of iteration 1 and implementation of first 20 out of 80 user stories) was completed at clock tick 543; release 2 at 956; release 3 at 1369 and release 4 at 1782 (each iteration implements 20 user stories). The implementation of the 80 user stories was completed before the deadline of 1800 clock ticks and customer never complained.

	Release 1	**Release 2**	**Release 3**	**Release 4**
Best Case	543	956	1369	1782

Table B.1: Clock ticks representing the completion of releases for best case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementa...	NumUserSt...	PercentErro...
Stories	80	Yes	80	100	100	80	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customer...	100	100	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

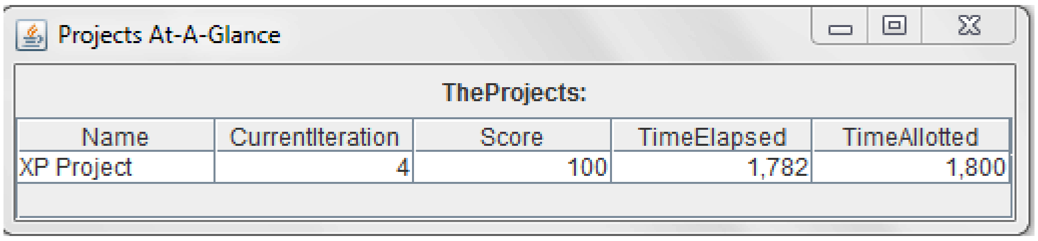
UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the cur...	0	100	100	100

Figure B.1: Final state of the artefacts for best case


Projects At-A-Glance

TheProjects:

Name	CurrentIteration	Score	TimeElapsed	TimeAllotted
XP Project	4	100	1,782	1,800

Figure B.2: Final project artefacts for best case

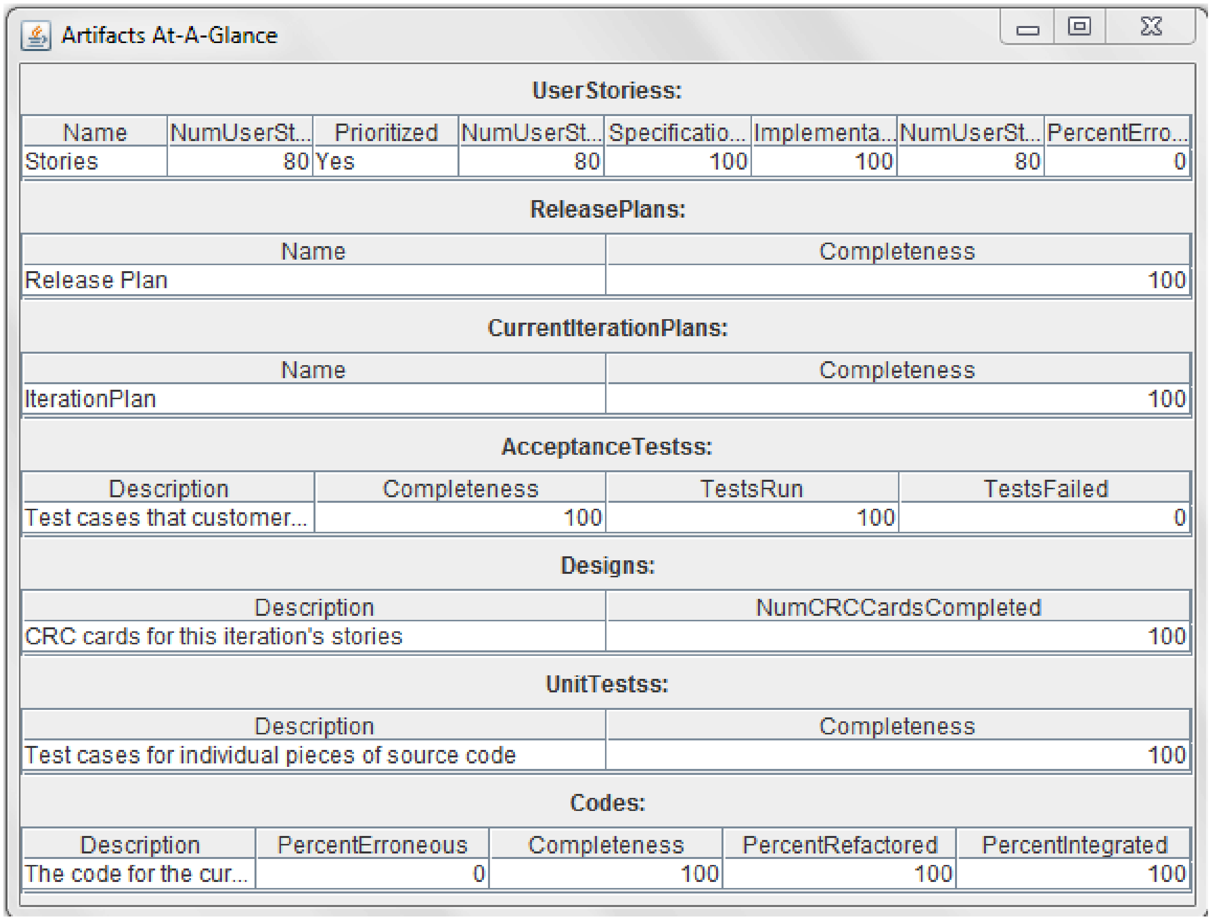
The CurrentIteration=4 means the above artefacts (i.e. code completeness, design completeness, unit test completeness, code erroneous and others) represent the values for the last iteration (as all these values are resetted once the iteration is completed, for the next iteration to start).

B.2: EWS Case Results

EWS case is the scenario where the user immediately applies corrective action on encountering warning message. The only difference between the best and the EWS case is that in the best case warning message never pops up whereas in EWS case the warning message appears once. The user reacts spontaneously to the warning message. For the EWS case, release 1 was completed at clock tick 543; release 2 at 956; release 3 at 1369 and release 4 at 1782. The implementation of the 80 user stories was completed before the deadline of 1800 clock ticks and customer never complained.

	Release 1	**Release 2**	**Release 3**	**Release 4**
EWS Case	543	956	1369	1782

Table B.2: Clock ticks representing the completion of releases for EWS case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementa...	NumUserSt...	PercentErro...
Stories	80	Yes	80	100	100	80	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customer...	100	100	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

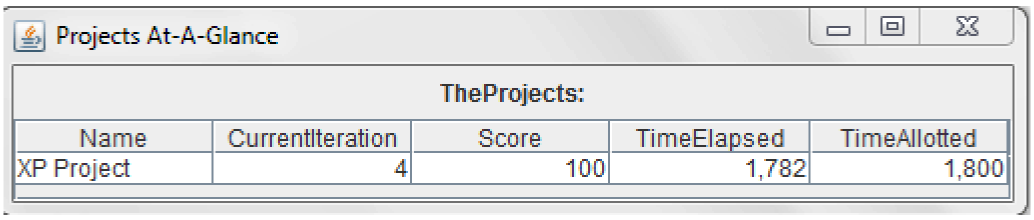
UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the cur...	0	100	100	100

Figure B.3: Final state of the artefacts for EWS case


Projects At-A-Glance

TheProjects:

Name	CurrentIteration	Score	TimeElapsed	TimeAllotted
XP Project	4	100	1,782	1,800

Figure B.4: Final project artefacts for EWS case

The CurrentIteration=4 means the above artefacts (i.e. code completeness, design completeness, unit test completeness, code erroneous and others) represent the values for the last iteration.

Appendix C: Individual EWS Testing Results

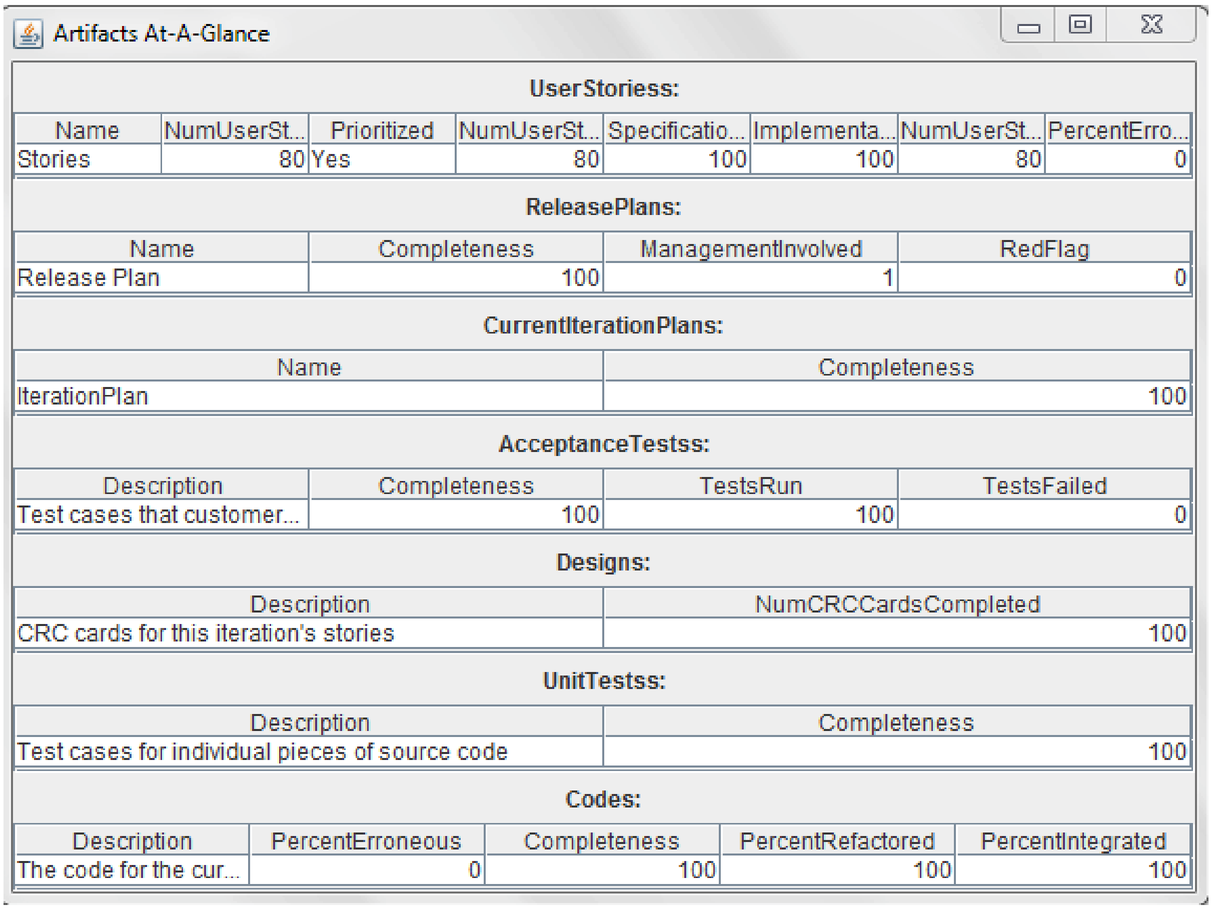
C1: EWS1 - Lack of top management support or commitment to the project

C1.1: Best Case

Best case is the scenario where the user selects the top management official (Manager Chang) in the release planning meeting at first go. Here no warning message appears as the user selects the management official in the meeting. Best case represents an ideal project i.e. all the right actions are taken by the user and everything goes smoothly and the project is completed successfully before the deadline. However, it is important to know that in all the simulation experiments conducted by this research budget is not considered (ignored) as a measure. The main reason for ignoring the budget is, the original XP SimSE model which was used as the initial prototype to develop this modified and intelligent early warning system ignored the budget and hence the resulting modified model also ignored the budget. The results are evaluated on the basis of the time (clock ticks) spent on each iteration (i.e. for each release) and the completeness and the quality of the delivered product. For the best case, release 1 (i.e. end of iteration 1 and implementation of first 20 out of 80 user stories) was completed at clock tick 543; release 2 at 956; release 3 at 1369 and release 4 at 1782 (each iteration implements 20 user stories). The implementation of the 80 user stories was completed before the deadline of 1800 clock ticks and customer never complained.

	Release 1	Release 2	Release 3	Release 4
Best Case	543	956	1369	1782

Table C.1: Clock ticks representing the completion of releases for best case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementa...	NumUserSt...	PercentErro...
Stories	80	Yes	80	100	100	80	0

ReleasePlans:

Name	Completeness	ManagementInvolved	RedFlag
Release Plan	100	1	0

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customer...	100	100	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

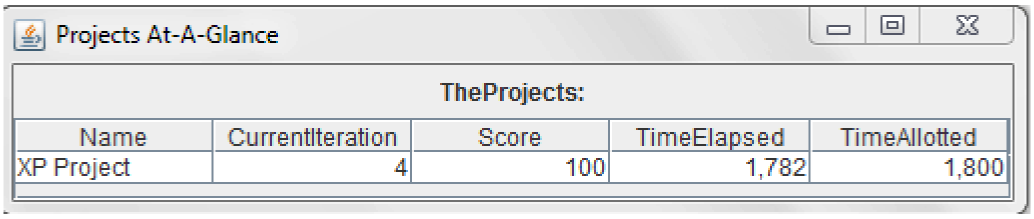
Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the cur...	0	100	100	100

Figure C.1: Final state of the artefacts for best case

Under ReleasePlans, the ManagementInvolved=1 indicate that the management official attended the release planning meeting. RedFlag=0 indicate that the warning message never appeared during the XP game play.



Projects At-A-Glance

TheProjects:

Name	CurrentIteration	Score	TimeElapsed	TimeAllotted
XP Project	4	100	1,782	1,800

Figure C.2: Final project artefacts for best case

The CurrentIteration=4 means the above artefacts (i.e. code completeness, design completeness, unit test completeness, code erroneous and others) represent the values for the last iteration (as all these values are resetted once the iteration is completed, for the next iteration to start). It is important to understand that the attribute ‘Score’ shouldn’t confuse and it’s not considered as a measure for evaluation by this research. The reason for ignoring the score attribute is because it is not considering the completeness and quality of the code for calculating the score. As this attribute was already existing in the initial XP prototype used by this modified XP system it was untouched but not of any importance in terms of evaluation in this research.

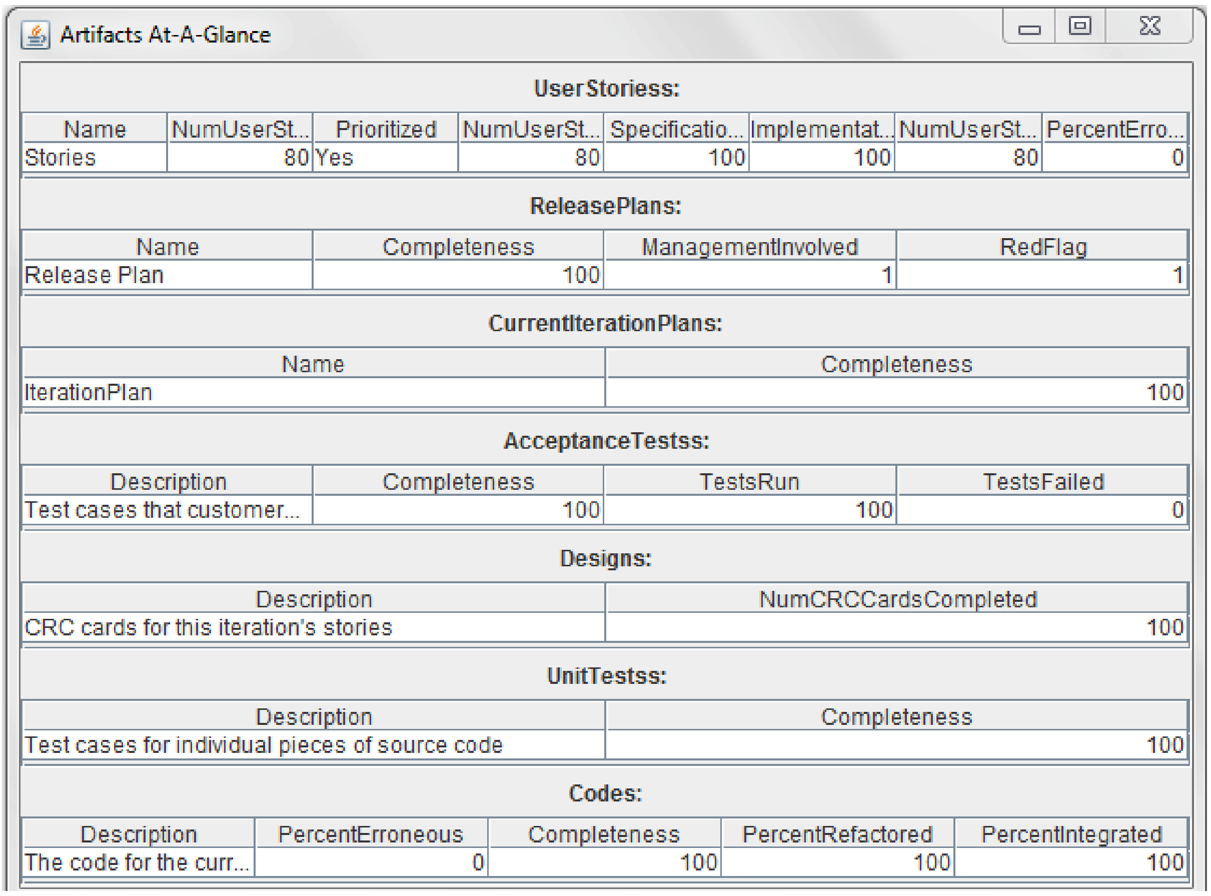
C1.2: Early Warning Sign (EWS) Case

EWS case is the scenario where the user does not select the management in the release planning meeting and the meeting starts but this immediately triggers the warning message. On encountering the early warning message the user reacts to it and selects the ‘Involve Management’ action immediately. Because the user acted so promptly the outcomes of this scenario are very similar to the best case. The only difference between the best and the EWS case is that in the best case warning message never pops up whereas in EWS case the warning message appears once (at the start of the release planning meeting). If compared to reality this scenario can be thought of as a concerned person identified just few minutes before (or at the start of) the meeting that the top management officials are missing and hence gives a quick call and requests them to attend the meeting. The author understands that this is sometimes practically difficult as the management officials are always lined up with work and may not be available in such a short notice. Also, it may be difficult for the management officials to be in the meeting on time, for instance if the meeting is scheduled onsite and the management is residing offsite. However, there are situations where management and meeting are at the same place (especially in big organisations). The author has personal experience where top level management were requested to attend a project meeting and they attended even though that wasn't planned by them. For the EWS case, release 1 was completed at clock tick 543; release 2 at 956; release 3 at 1369 and release 4 at 1782. The implementation of the 80 user stories was completed before the deadline of 1800 clock ticks and customer never complained.

	Release 1	**Release 2**	**Release 3**	**Release 4**
EWS Case	543	956	1369	1782

Table C.2: Clock ticks representing the completion of releases for EWS case

The final artefacts state is shown in the figure below. Under ReleasePlans, the ManagementInvolved=1 indicate that the management official attended the release planning meeting. RedFlag=1 indicate that the warning message appeared during the XP game play.


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementat...	NumUserSt...	PercentErro...
Stories	80	Yes	80	100	100	80	0

ReleasePlans:

Name	Completeness	ManagementInvolved	RedFlag
Release Plan	100	1	1

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customer...	100	100	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

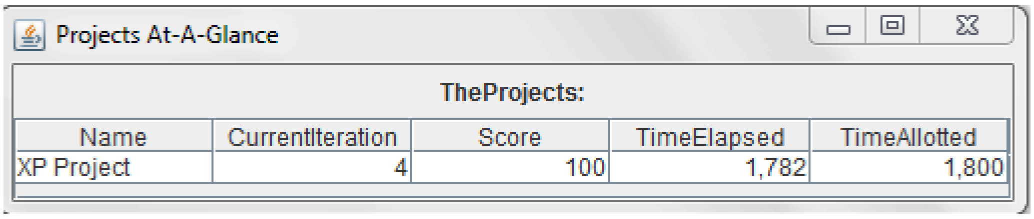
UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curr...	0	100	100	100

Figure C.3: Final state of the artefacts for EWS case


Projects At-A-Glance

TheProjects:

Name	CurrentIteration	Score	TimeElapsed	TimeAllotted
XP Project	4	100	1,782	1,800

Figure C.4: Final project artefacts for EWS case

The CurrentIteration=4 means the above artefacts (i.e. code completeness, design completeness, unit test completeness, code erroneous and others) represent the values for the last iteration (as all these values are resetted once the iteration is completed, for the next iteration to start).

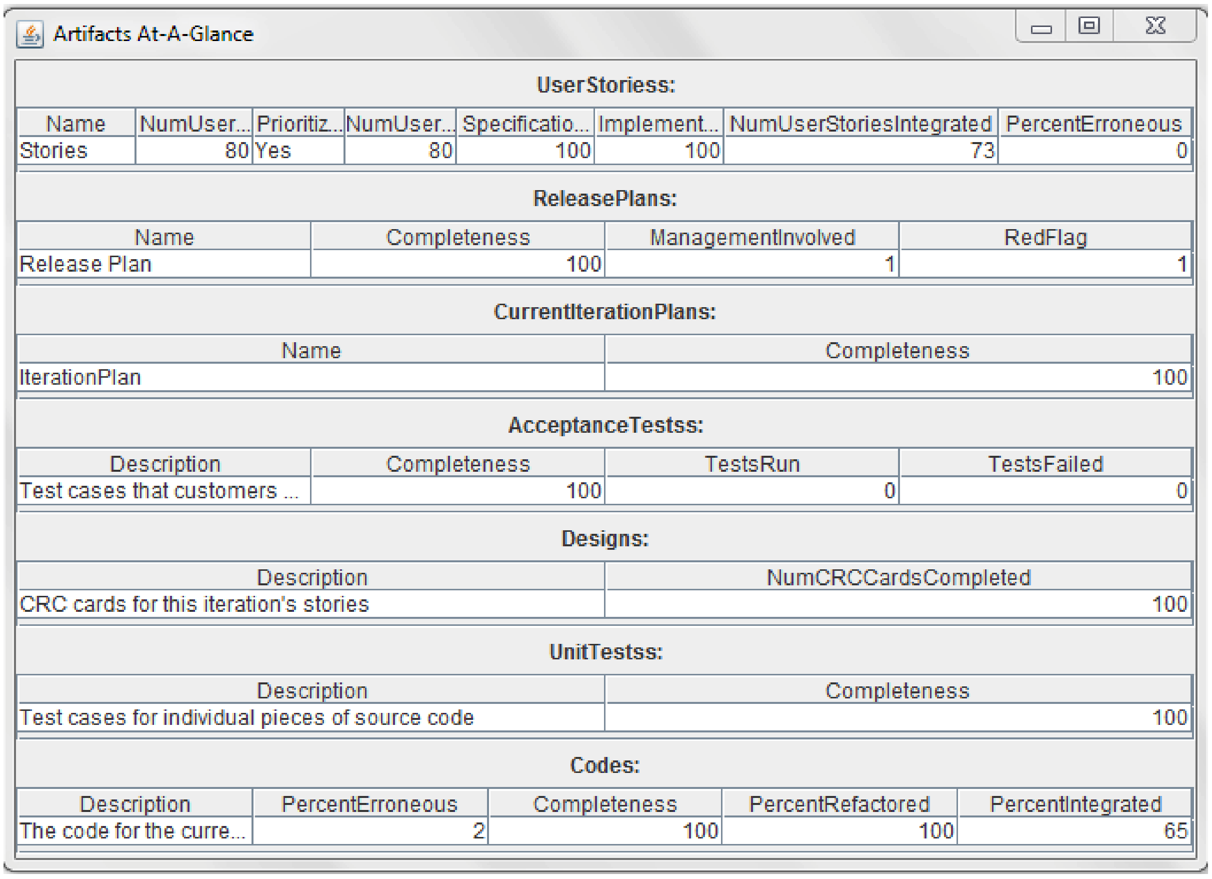
C1.3: First Iteration

This is the scenario where the user does not select the management in the release planning meeting and keeps on ignoring the warning sign and does not select the ‘Involve Management’ action until the start of first iteration. The XP model is developed in such a way that the warning message appears repetitively so that the user is warned constantly. At the start of first iteration, on encountering the warning message the user selects the ‘Involve Management’ action. However, due to delaying the select action slight damage has been done and some project time has been lost already. Hence for this case, release 1 was completed at 605 clock ticks, release 2 at 1018 and release 3 at 1431. Due to the deadline of 1800 clock ticks release 4 was never fully completed. Throughout the project, customer also complained once.

	Release 1	**Release 2**	**Release 3**	**Release 4**
First Iteration	605	1018	1431	-

Table C.3: Clock ticks representing the completion of releases for ‘First Iteration’ case

The final artefacts state is shown in the figure below. Under ReleasePlans, the ManagementInvolved=1 indicate that the management official attended the release planning meeting. RedFlag=1 indicate that the warning message appeared during the XP game play.



Artifacts At-A-Glance

UserStoriess:

Name	NumUser...	Prioritiz...	NumUser...	Specificatio...	Implement...	NumUserStoriesIntegrated	PercentErroneous
Stories	80	Yes	80	100	100	73	0

ReleasePlans:

Name	Completeness	ManagementInvolved	RedFlag
Release Plan	100	1	1

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers ...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

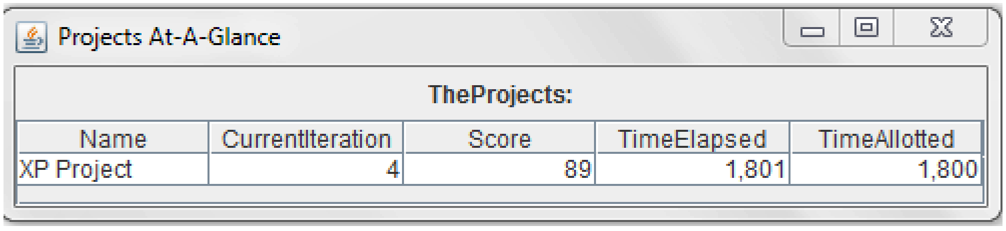
Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curre...	2	100	100	65

Figure C.5: Final state of the artefacts for ‘First Iteration’ case

As compared to the previous case, in addition to the final release not being completed the code is 2% erroneous. Only 65% of the code has been integrated in 4th iteration (hence overall only 73 user stories were integrated out of 80), which provides a measure of how close to completion the project was.



Projects At-A-Glance

TheProjects:

Name	CurrentIteration	Score	TimeElapsed	TimeAllotted
XP Project	4	89	1,801	1,800

Figure C.6: Final project artefacts for ‘First Iteration’ case

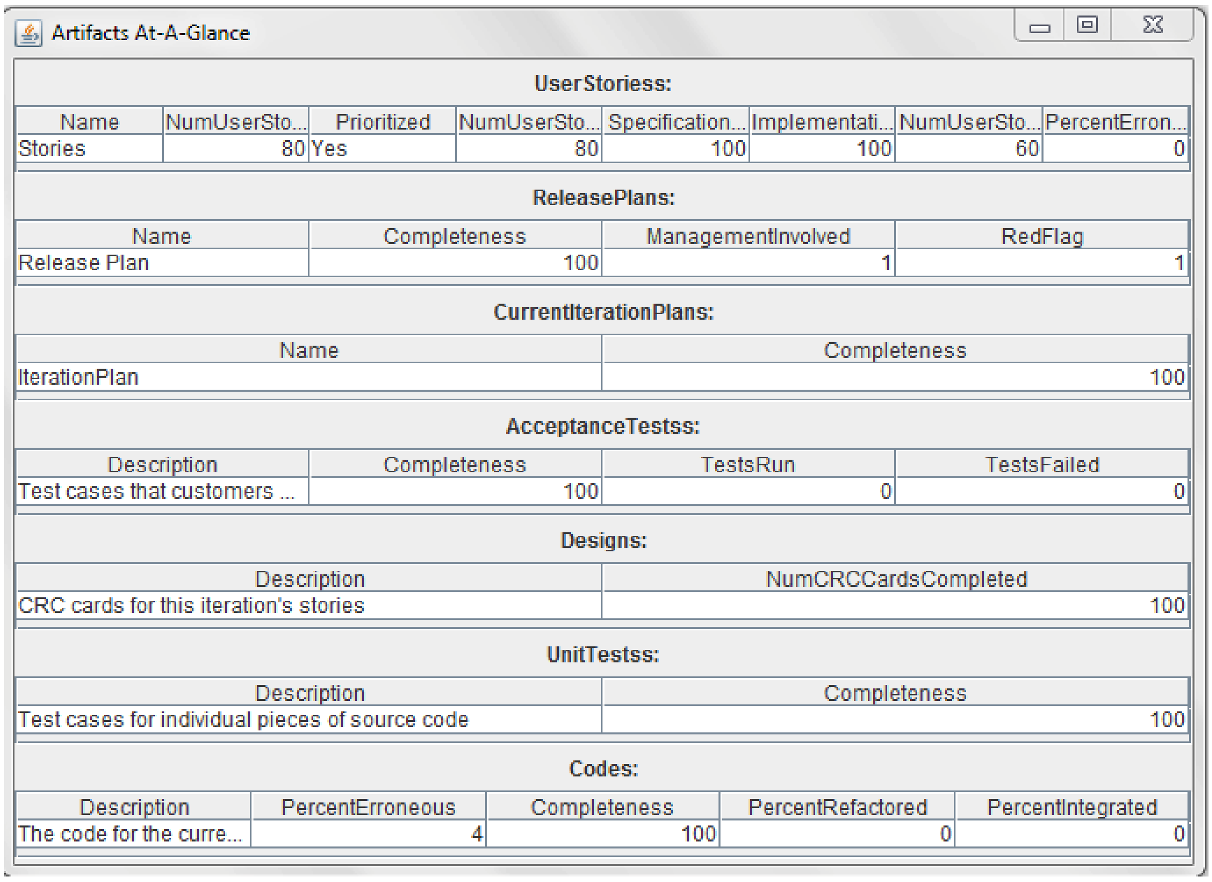
The CurrentIteration=4 means the above artefacts (i.e. code completeness, design completeness, unit test completeness, code erroneous and others) represent the values for the last iteration (as all these values are resetted once the iteration is completed, for the next iteration to start).

C1.4: Between 1st and 2nd Iteration

This is the scenario where the user does not select the management in the release planning meeting and keeps on ignoring the warning sign and does not select the 'Involve Management' action until the game reaches in the middle of the first iteration. The XP model is developed in such a way that the warning message appears repetitively so that the user is warned constantly. At the middle of first iteration, on encountering the warning message the user selects the 'Involve Management' action. However, due to delaying the select action slight damage has been done and some project time has been lost already. Hence for this case, release 1 was completed at 667 clock ticks, release 2 at 1080 and release 3 at 1493. Due to the deadline of 1800 clock ticks release 4 was never fully completed. Throughout the project, customer complained twice.

	Release 1	**Release 2**	**Release 3**	**Release 4**
Between 1st and 2nd Iteration	667	1080	1493	-

Table C.4: Clock ticks representing the completion of releases for 'Between 1st and 2nd Iteration' case



Artifacts At-A-Glance

UserStoriess:

Name	NumUserSto...	Prioritized	NumUserSto...	Specification...	Implementati...	NumUserSto...	PercentErron...
Stories	80	Yes	80	100	100	60	0

ReleasePlans:

Name	Completeness	ManagementInvolved	RedFlag
Release Plan	100	1	1

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers ...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

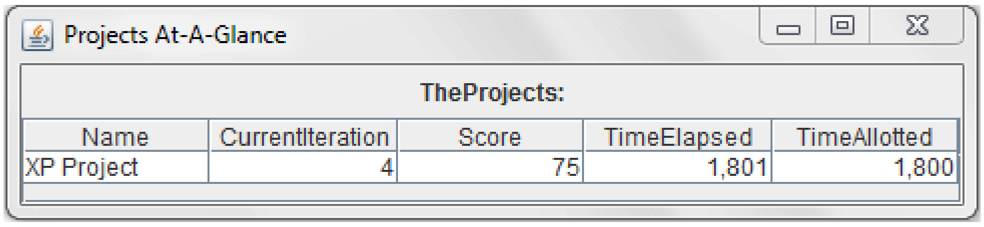
Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curre...	4	100	0	0

Figure C.7: Final state of the artefacts for ‘Between 1st and 2nd Iteration’ case

As compared to the previous case, only 60 user stories were integrated. The 4th iteration’s code was not refactored and integrated at all and it was 4% erroneous. This represents a worse end situation than was achieved in the previous case.



Projects At-A-Glance

TheProjects:

Name	CurrentIteration	Score	TimeElapsed	TimeAllotted
XP Project	4	75	1,801	1,800

Figure C.8: Final project artefacts for ‘Between 1st and 2nd Iteration’ case

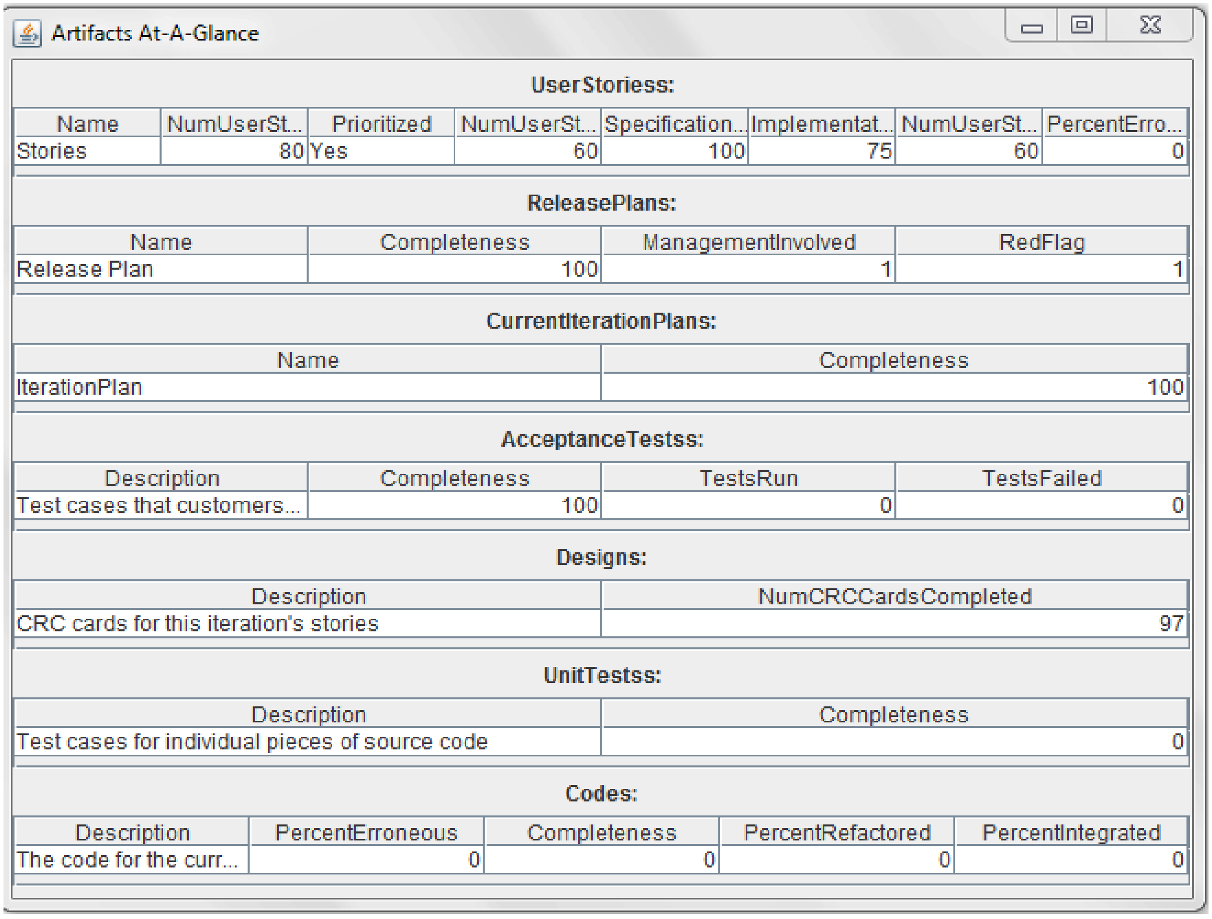
The CurrentIteration=4 means the above artefacts (i.e. code completeness, design completeness, unit test completeness, code erroneous and others) represent the values for the last iteration.

C1.5: Second Iteration

This is the scenario where the user does not select the management in the release planning meeting and keeps on ignoring the warning sign and does not select the 'Involve Management' action until the start of second iteration. At the start of the second iteration, on encountering the warning message the user selects the 'Involve Management' action. However, some time has been lost already. Hence for this case, release 1 was completed at 791 clock ticks, release 2 at 1266 and release 3 at 1679. Due to the deadline of 1800 clock ticks release 4 was never fully completed. Throughout the project, customer complained six times.

	Release 1	**Release 2**	**Release 3**	**Release 4**
Second Iteration	791	1266	1679	-

Table C.5: Clock ticks representing the completion of releases for 'Second Iteration' case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specification...	Implementat...	NumUserSt...	PercentErro...
Stories	80	Yes	60	100	75	60	0

ReleasePlans:

Name	Completeness	ManagementInvolved	RedFlag
Release Plan	100	1	1

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	97

UnitTestss:

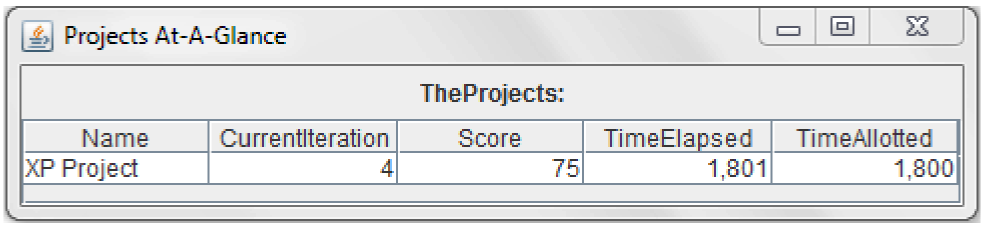
Description	Completeness
Test cases for individual pieces of source code	0

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curr...	0	0	0	0

Figure C.9: Final state of the artefacts for 'Second Iteration' case

As compared to the previous case, the design is only 97% complete for 4th iteration. The implementation completeness is only 75% (compared to 100% in previous case). The unit tests creation and coding were not even started. It is important to analyse the end results completely, here the code is not erroneous because it has not been started. This is a worse ending situation than the previous example.


Projects At-A-Glance

TheProjects:

Name	CurrentIteration	Score	TimeElapsed	TimeAllotted
XP Project	4	75	1,801	1,800

Figure C.10: Final project artefacts for ‘Second Iteration’ case

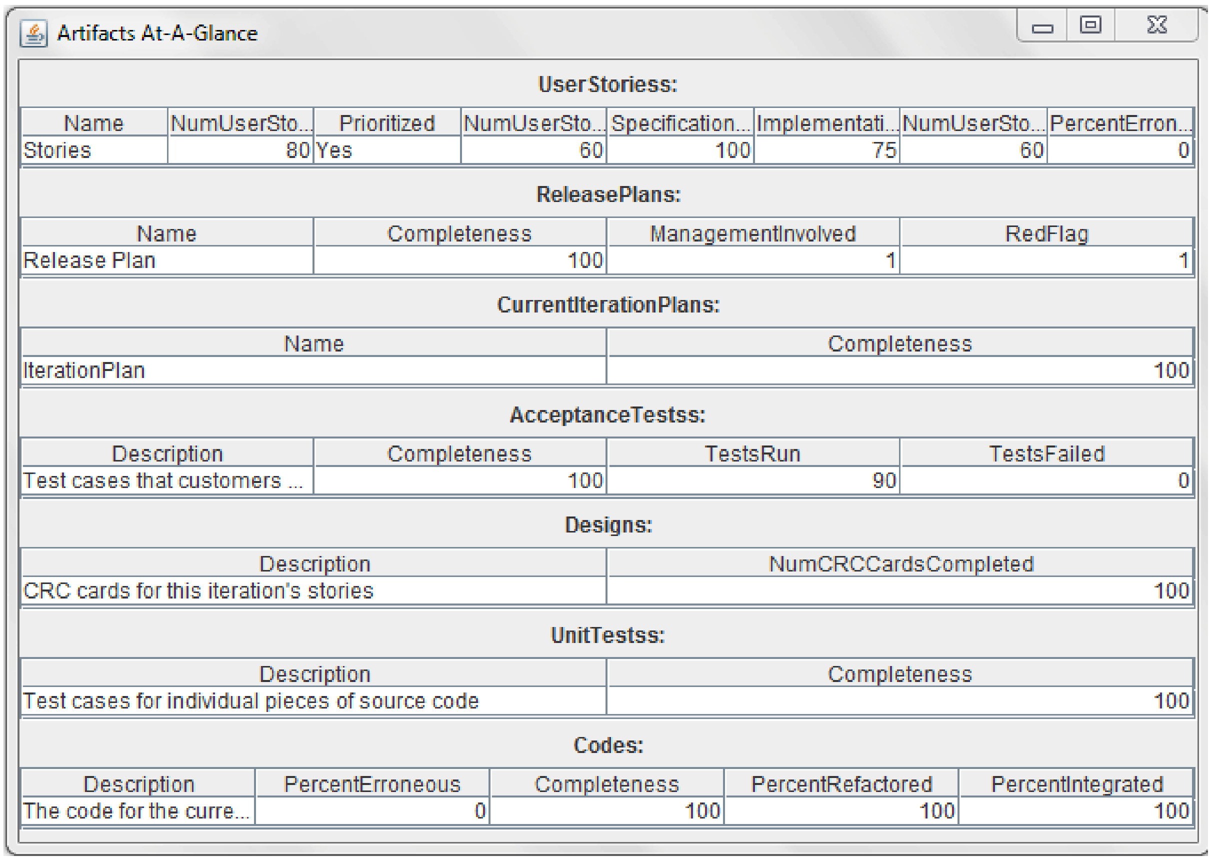
The CurrentIteration=4 means the above artefacts (i.e. code completeness, design completeness, unit test completeness, code erroneous and others) represent the values for the last iteration.

C1.6: Between 2nd and 3rd Iteration

This is the scenario where the user does not select the management in the release planning meeting and keeps on ignoring the warning sign and does not select the ‘Involve Management’ action until the game reaches in the middle of the second iteration. At the middle of the second iteration, on encountering the warning message the user selects the ‘Involve Management’ action. However, due to delay in the selection of ‘Involve Management’ action some project time has been lost already. Hence for this case, release 1 was completed at 791 clock ticks and release 2 at 1389. Release 3 was not completed (and that means release 4 was not even started). Throughout the project, customer complained nine times.

	Release 1	Release 2	Release 3	Release 4
Between 2nd and 3rd Iteration	791	1389	-	-

Table C.6: Clock ticks representing the completion of releases for ‘Between 2nd and 3rd Iteration’ case



Artifacts At-A-Glance

UserStoriess:

Name	NumUserSto...	Prioritized	NumUserSto...	Specification...	Implementati...	NumUserSto...	PercentErron...
Stories	80	Yes	60	100	75	60	0

ReleasePlans:

Name	Completeness	ManagementInvolved	RedFlag
Release Plan	100	1	1

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers ...	100	90	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

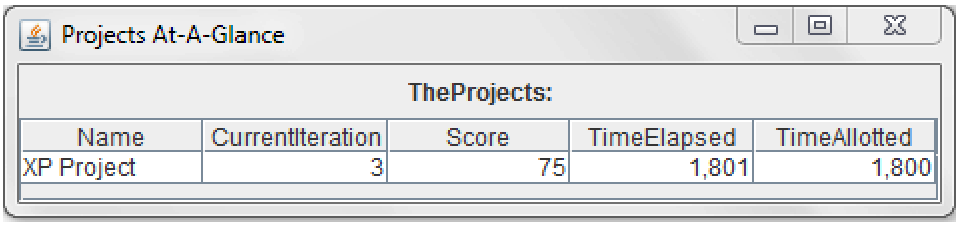
Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curre...	0	100	100	100

Figure C.11: Final state of the artefacts for ‘Between 2nd and 3rd Iteration’ case

As compared to the previous case, this case was still in its 3rd iteration. The project was in its last phase in 3rd iteration with 90% acceptance tests completed. Again, it is important to reiterate that to make a direct comparison with previous cases that all artefact values need to be considered. In this scenario, the project has not completed the third iteration and therefore is a worse ending situation than the previous case.



Projects At-A-Glance

TheProjects:

Name	CurrentIteration	Score	TimeElapsed	TimeAllotted
XP Project	3	75	1,801	1,800

Figure C.12: Final Project artefacts for ‘Between 2nd and 3rd Iteration’ case

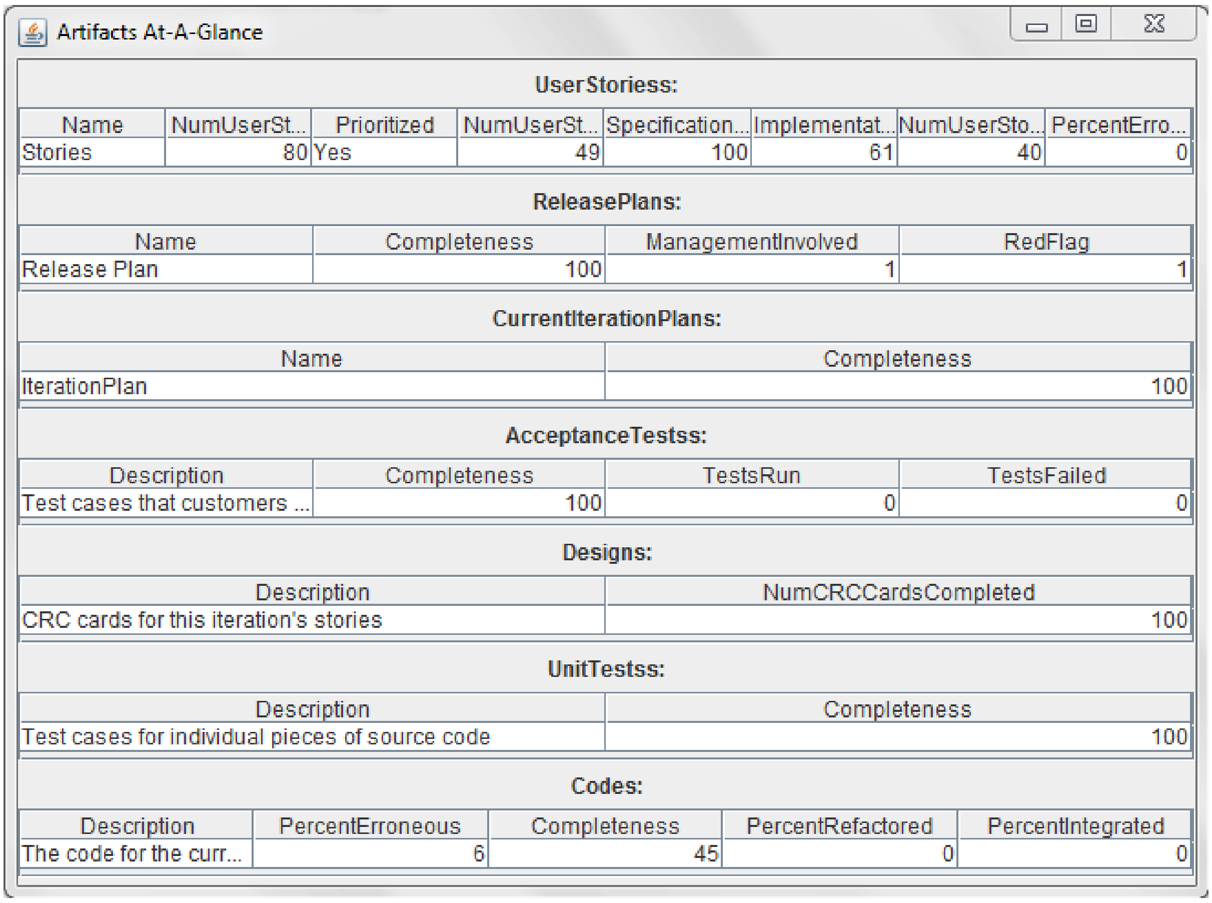
The CurrentIteration=3 means the above artefacts (i.e. code completeness, design completeness, unit test completeness, code erroneous and others) represent the values for the iteration 3.

C1.7: Third Iteration

This is the scenario where the user does not select the management in the release planning meeting and keeps on ignoring the warning sign and does not select the 'Involve Management' action until the start of third iteration. At the start of the third iteration, on encountering the warning message the user selects the 'Involve Management' action. However, some time has been lost already. Hence for this case, release 1 was completed at 791 clock ticks and release 2 at 1514. Release 3 was not completed (and that means release 4 was not even started). Throughout the project, customer complained 11 times.

	Release 1	**Release 2**	**Release 3**	**Release 4**
Third Iteration	791	1514	-	-

Table C.7: Clock ticks representing the completion of releases for 'Third Iteration' case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specification...	Implementat...	NumUserSto...	PercentErro...
Stories	80	Yes	49	100	61	40	0

ReleasePlans:

Name	Completeness	ManagementInvolved	RedFlag
Release Plan	100	1	1

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers ...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

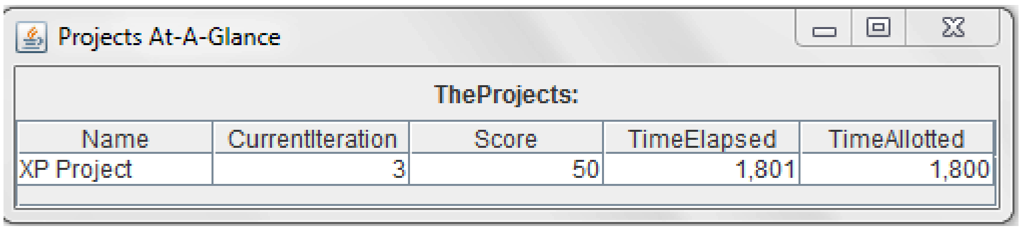
Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curr...	6	45	0	0

Figure C.13: Final state of the artefacts for 'Third Iteration' case

As compared to the previous case, only 45% of the code was complete in 3rd iteration with 6% erroneous. Further, implementation completeness was only 61%; number of user stories implemented was 49 and the number of user stories integrated was only 40 (compared to 75%, 60 and 60 in previous case). This is a worse ending situation than the previous example.


Projects At-A-Glance

TheProjects:

Name	CurrentIteration	Score	TimeElapsed	TimeAllotted
XP Project	3	50	1,801	1,800

Figure C.14: Final project artefacts for 'Third Iteration' case

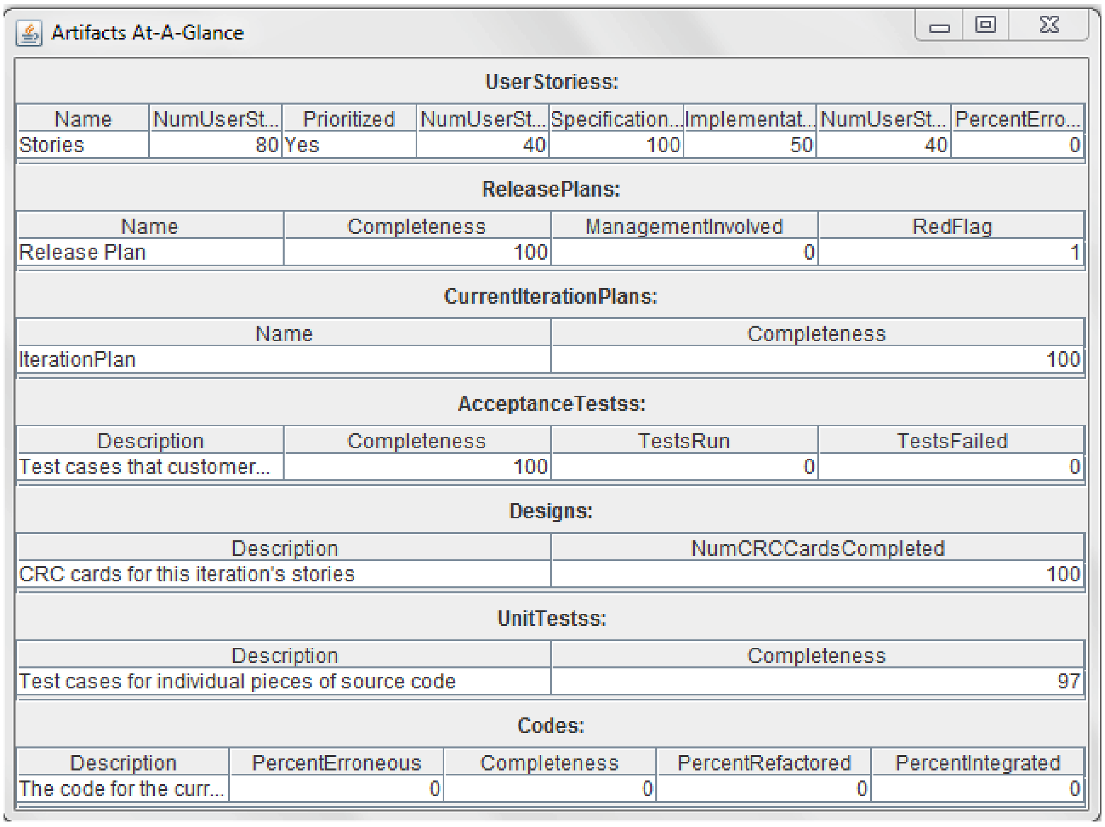
The CurrentIteration=3 means the above artefacts (i.e. code completeness, design completeness, unit test completeness, code erroneous and others) represent the values for the iteration 3.

C1.8: Worst Case

This is the scenario where the user does not select the management in the release planning meeting and keeps on ignoring all the warning sign and does not select the 'Involve Management' action throughout the game. Hence for this case, release 1 was completed at 791 clock ticks and release 2 at 1514. Release 3 was not completed (and that means release 4 was not even started). Throughout the project, customer complained 11 times. It is important to note that the time taken for releases 1 and 2 by the 'third iteration' case and worst case is the same but there is a difference between the completeness of the code.

	Release 1	**Release 2**	**Release 3**	**Release 4**
Worst Case	791	1514	-	-

Table C.8: Clock ticks representing the completion of releases for 'Worst case'


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specification...	Implementat...	NumUserSt...	PercentErro...
Stories	80	Yes	40	100	50	40	0

ReleasePlans:

Name	Completeness	ManagementInvolved	RedFlag
Release Plan	100	0	1

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customer...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	97

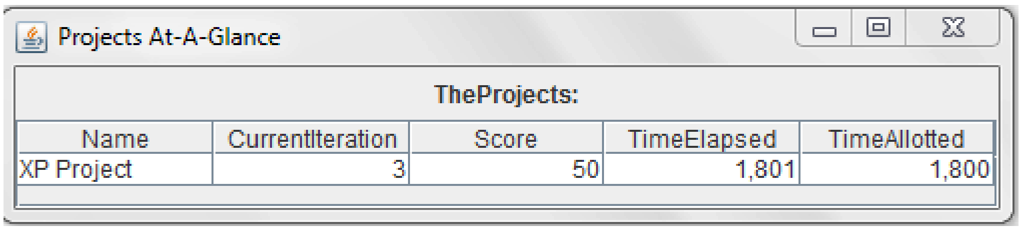
Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curr...	0	0	0	0

Figure C.15: Final state of the artefacts for ‘Worst case’

Under ReleasePlans, the ManagementInvolved=0 indicate that the management official never attended the release planning meeting of the project and their support was never acquired. RedFlag=1 indicate that the warning message appeared during the XP game play.

As compared to the previous case, the implementation completeness was only 50% and number of user stories implemented was only 40 (compared to 61% and 49 in previous case). Only 97% of the unit tests were created and coding was not even stated in 3rd iteration. This is a worse ending situation than the previous example.


Projects At-A-Glance

TheProjects:

Name	CurrentIteration	Score	TimeElapsed	TimeAllotted
XP Project	3	50	1,801	1,800

Figure C.16: Final project artefacts for 'Worst case'

The CurrentIteration=3 means the above artefacts (i.e. code completeness, design completeness, unit test completeness, code erroneous and others) represent the values for the iteration 3.

C2: EWS2 - Project stakeholders have not been interviewed for project requirements

C2.1: Best Case

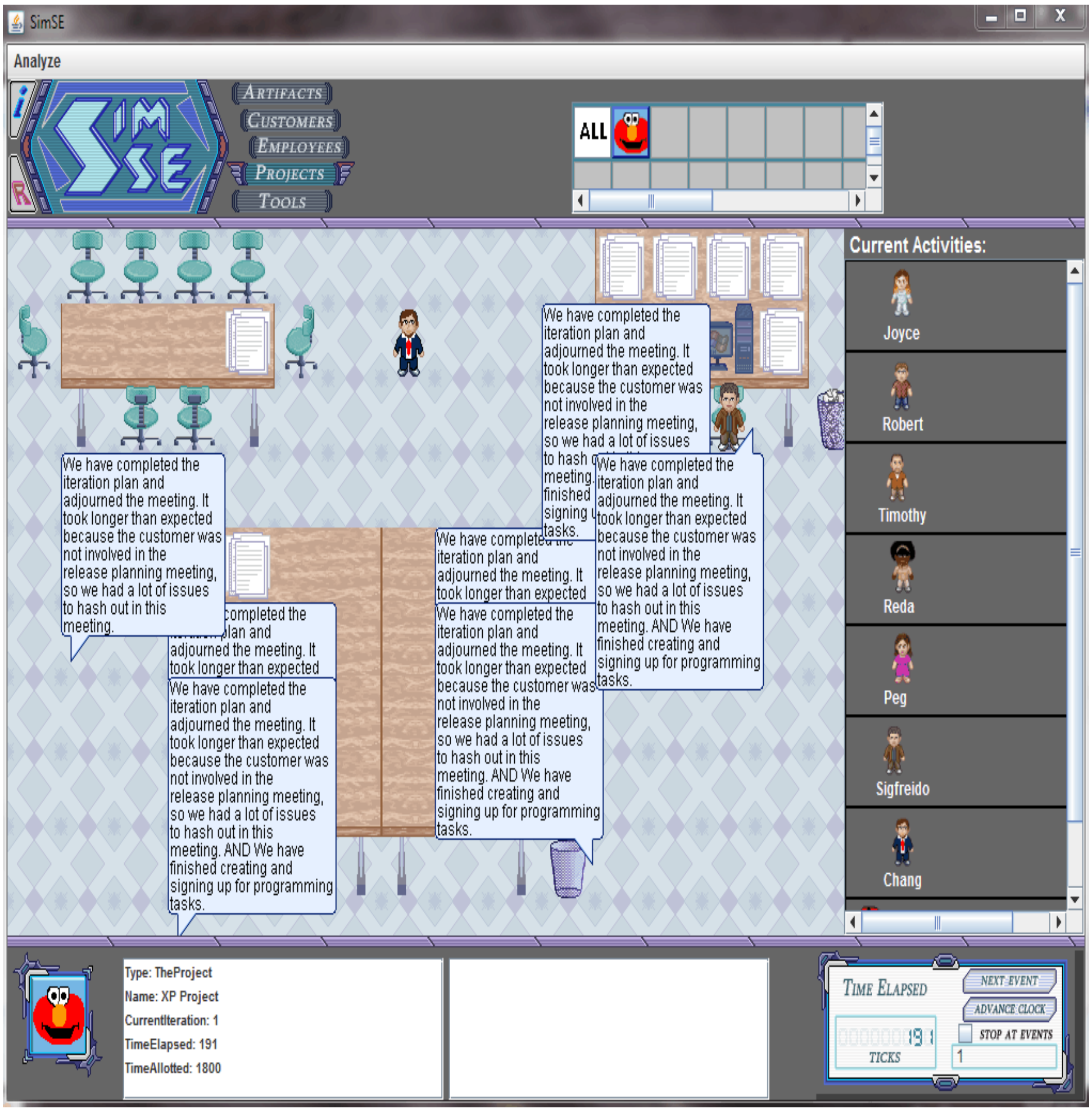
Best case is the scenario where the user selects the customer representative (customer Wayne) in the release planning meeting at first go. Here no warning message appears as the user selects the customer in the meeting. Best case represents an ideal project i.e. all the right actions are taken by the user and everything goes smoothly and the project is completed successfully before the deadline. The detail of the best case is exactly same as explained earlier. In order to remove the repetition the author is not including the same details here. Please refer to appendix C1.1 for inspection of detailed results.

C2.2 EWS Case

EWS case is the scenario where the user does not select the customer in the release planning meeting and the meeting starts but this immediately triggers the warning message. On encountering the early warning message the user reacts to it and selects the 'Involve Customer' action immediately. Because the user acted so spontaneously the outcomes of this scenario is exactly similar to the best case. The only difference between the best and the EWS case is that in the best case warning message never pops up whereas in EWS case the warning message appears once (at the start of the release planning meeting). The detail of the EWS case is exactly same as explained earlier. In order to remove the repetition the author is not including the same details here. Please refer to appendix C1.2 for inspection of detailed results.

C2.3: Second Iteration

This is the scenario where the user does not select the customer in the release planning meeting and keeps on ignoring the warning sign and does not select the 'Involve Customer' action until the start of second iteration. At the start of the second iteration, on encountering the warning message the user selects the 'Involve Customer' action. But there were many problems encountered during iteration 1 (due to the absence of customer in the release planning meeting) and the activities were taking longer time than expected. The iteration planning meeting for iteration 1 also took longer and displayed following message as shown in figure C.17.



We have completed the iteration plan and adjourned the meeting. It took longer than expected because the customer was not involved in the release planning meeting, so we had a lot of issues to hash out in this meeting.

completed the plan and adjourned the meeting. It took longer than expected

We have completed the iteration plan and adjourned the meeting. It took longer than expected because the customer was not involved in the release planning meeting, so we had a lot of issues to hash out in this meeting. AND We have finished creating and signing up for programming tasks.

We have completed the iteration plan and adjourned the meeting. It took longer than expected

We have completed the iteration plan and adjourned the meeting. It took longer than expected because the customer was not involved in the release planning meeting, so we had a lot of issues to hash out in this meeting. AND We have finished creating and signing up for programming tasks.

We have completed the iteration plan and adjourned the meeting. It took longer than expected because the customer was not involved in the release planning meeting, so we had a lot of issues to hash out in this meeting. finished signing tasks.

We have completed the iteration plan and adjourned the meeting. It took longer than expected because the customer was not involved in the release planning meeting, so we had a lot of issues to hash out in this meeting. AND We have finished creating and signing up for programming tasks.

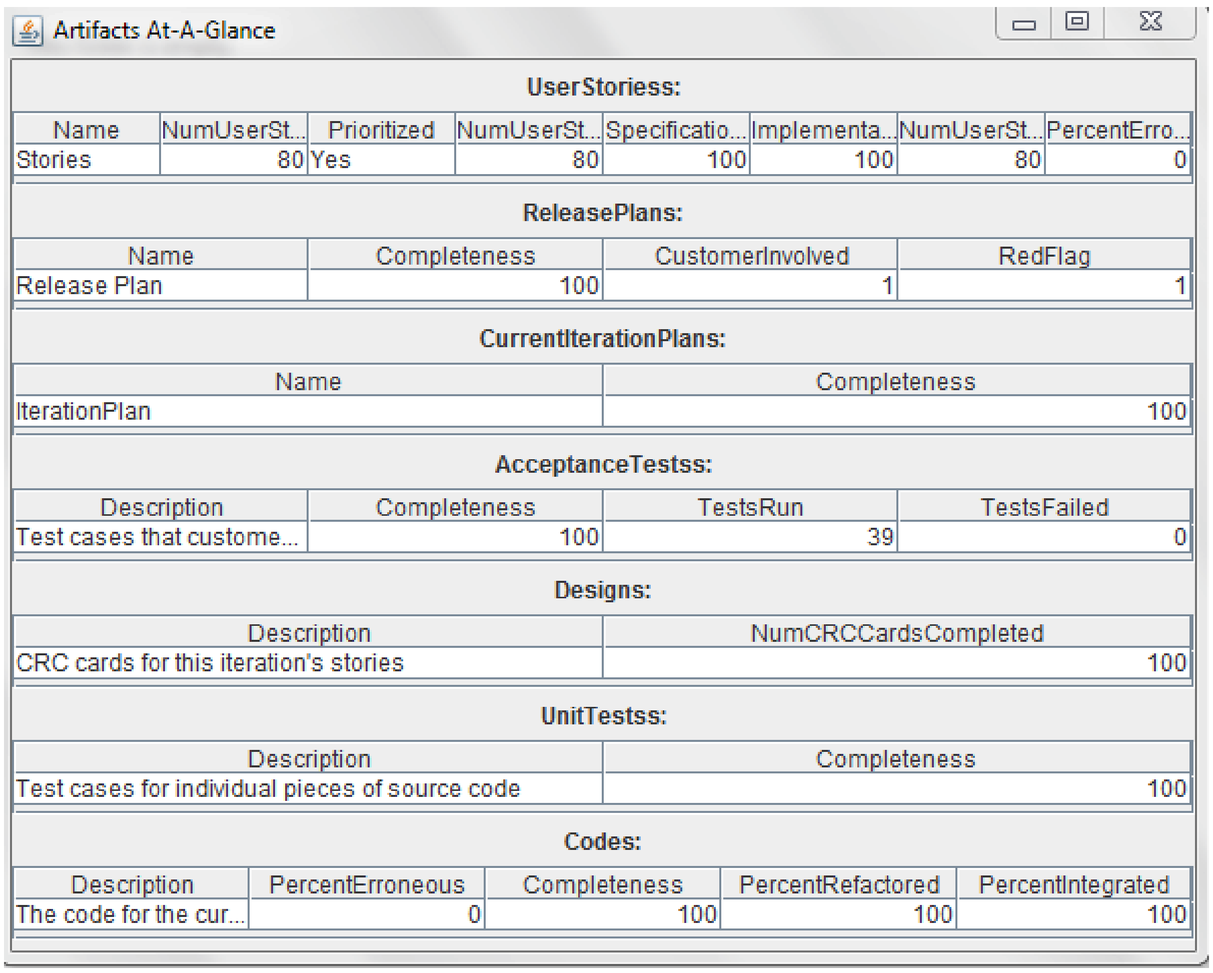
Type: TheProject
Name: XP Project
CurrentIteration: 1
TimeElapsed: 191
TimeAllotted: 1800

Figure C.17: Message displayed at the end of iteration planning meeting of iteration 1

This has had an impact on the project schedule. Hence for this case, release 1 was completed at 573 clock ticks, release 2 at 986 and release 3 at 1399. Due to the deadline of 1800 clock ticks release 4 was never fully completed. Customer complained once throughout the project.

	Release 1	Release 2	Release 3	Release 4
Second Iteration	573	986	1399	-

Table C.9: Clock ticks representing the completion of releases for 'Second Iteration' case



Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementa...	NumUserSt...	PercentErro...
Stories	80	Yes	80	100	100	80	0

ReleasePlans:

Name	Completeness	CustomerInvolved	RedFlag
Release Plan	100	1	1

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that custome...	100	39	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the cur...	0	100	100	100

Figure C.18: Final state of the artefacts for 'Second Iteration' case

In the 4th iteration, acceptance tests run have dropped to 39% from the previous 100%. This is a worse ending situation than the previous example. Under ReleasePlans, the CustomerInvolved=1 indicate that the customer representative attended the release planning meeting. RedFlag=1 indicate that the warning message appeared during the XP game play.

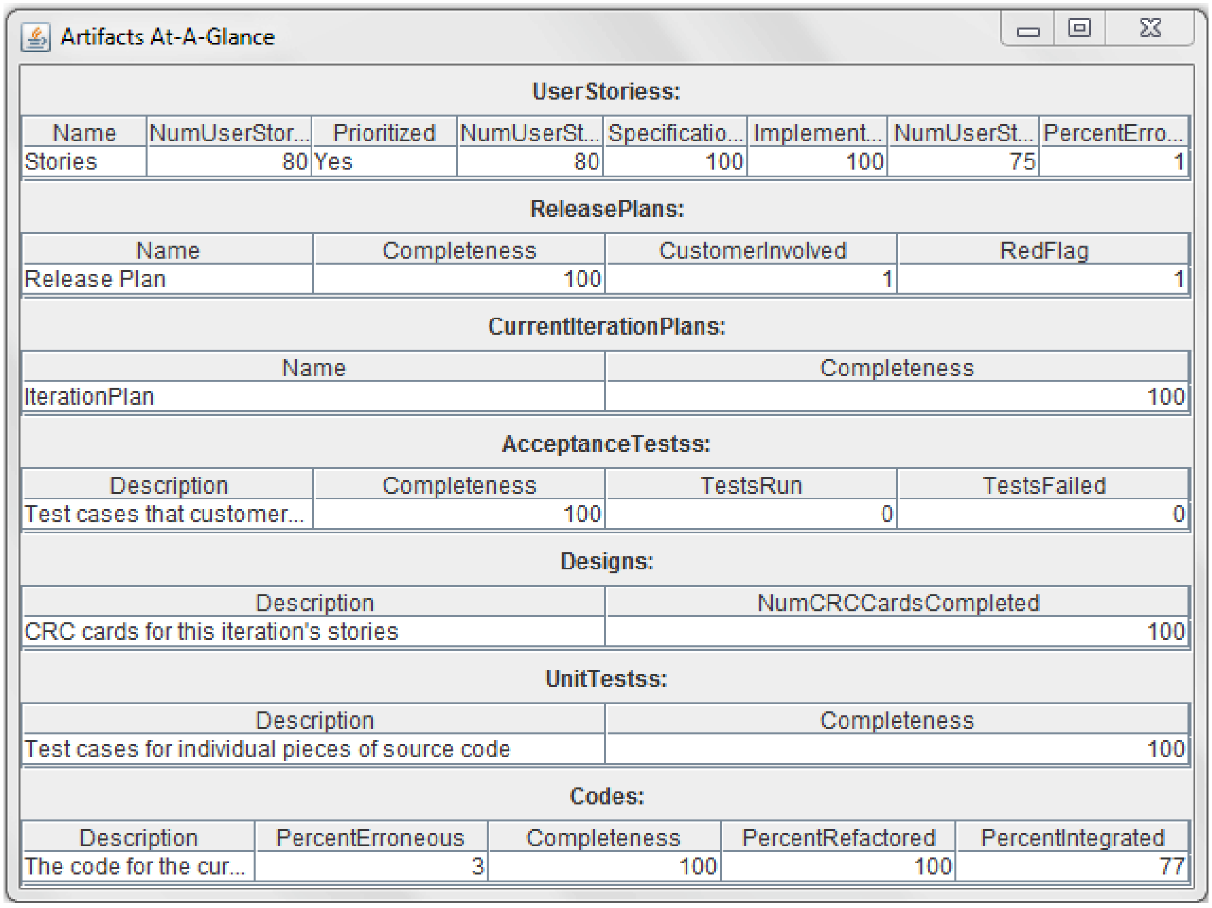
C2.4: Third Iteration

This is the scenario where the user does not select the customer in the release planning meeting and keeps on ignoring the warning sign and does not select the 'Involve Customer' action until the start of third iteration. At the start of the third iteration, on encountering the warning message the user selects the

‘Involve Customer’ action. But some time has been lost already. Hence for this case, release 1 was completed at 573 clock ticks, release 2 at 1016 and release 3 at 1429. Due to the deadline of 1800 clock ticks release 4 was never fully completed. Customer complained twice throughout the project.

	Release 1	Release 2	Release 3	Release 4
Third Iteration	573	1016	1429	-

Table C.10: Clock ticks representing the completion of releases for ‘Third Iteration’ case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserStor...	Prioritized	NumUserSt...	Specificatio...	Implement...	NumUserSt...	PercentErro...
Stories	80	Yes	80	100	100	75	1

ReleasePlans:

Name	Completeness	CustomerInvolved	RedFlag
Release Plan	100	1	1

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customer...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the cur...	3	100	100	77

Figure C.19: Final state of the artefacts for ‘Third Iteration’ case

Compared to the previous case, the percent erroneous has gone up to 3% and the percent integrated has dropped to 77% from 100% in 4th iteration. The total number of user stories integrated has fallen to 75 from 80. This is a worse ending situation than the previous example. Under ReleasePlans, the

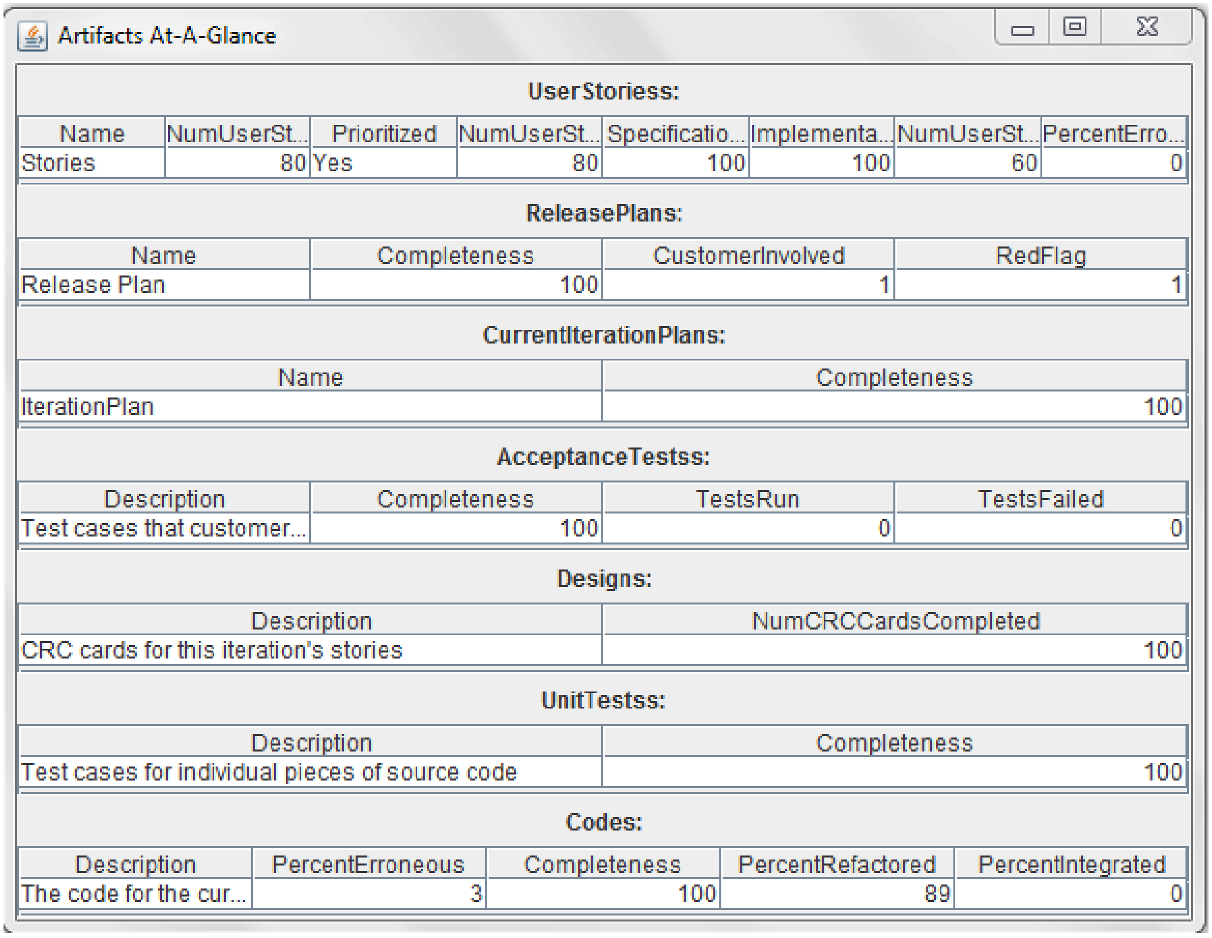
CustomerInvolved=1 indicate that the customer representative attended the release planning meeting. RedFlag=1 indicate that the warning message appeared during the XP game play.

C2.5: Fourth Iteration

This is the scenario where the user does not select the customer in the release planning meeting and keeps on ignoring the warning sign and does not select the 'Involve Customer' action until the start of the fourth iteration. At the start of the fourth iteration, on encountering the warning message the user selects the 'Involve Customer' action. But some time has already been lost. Hence for this case, release 1 was completed at 573 clock ticks, release 2 at 1016 and release 3 at 1459. Due to the deadline of 1800 clock ticks release 4 was never fully completed. Customer complained thrice throughout the project.

	Release 1	**Release 2**	**Release 3**	**Release 4**
Fourth Iteration	573	1016	1459	-

Table C.11: Clock ticks representing the completion of releases for 'Fourth Iteration' case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementa...	NumUserSt...	PercentErro...
Stories	80	Yes	80	100	100	60	0

ReleasePlans:

Name	Completeness	CustomerInvolved	RedFlag
Release Plan	100	1	1

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customer...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the cur...	3	100	89	0

Figure C.20: Final state of the artefacts for ‘Fourth Iteration’ case

Again in comparison with the previous case, in iteration four the percent refactored dropped to 89% from 100% and percent integrated dropped to 0. Also, the total number of user stories integrated has gone down to 60 from 75. This is a worse ending situation than the previous example. Under ReleasePlans, the CustomerInvolved=1 indicate that the customer representative attended the release planning meeting. RedFlag=1 indicate that the warning message appeared during the XP game play.

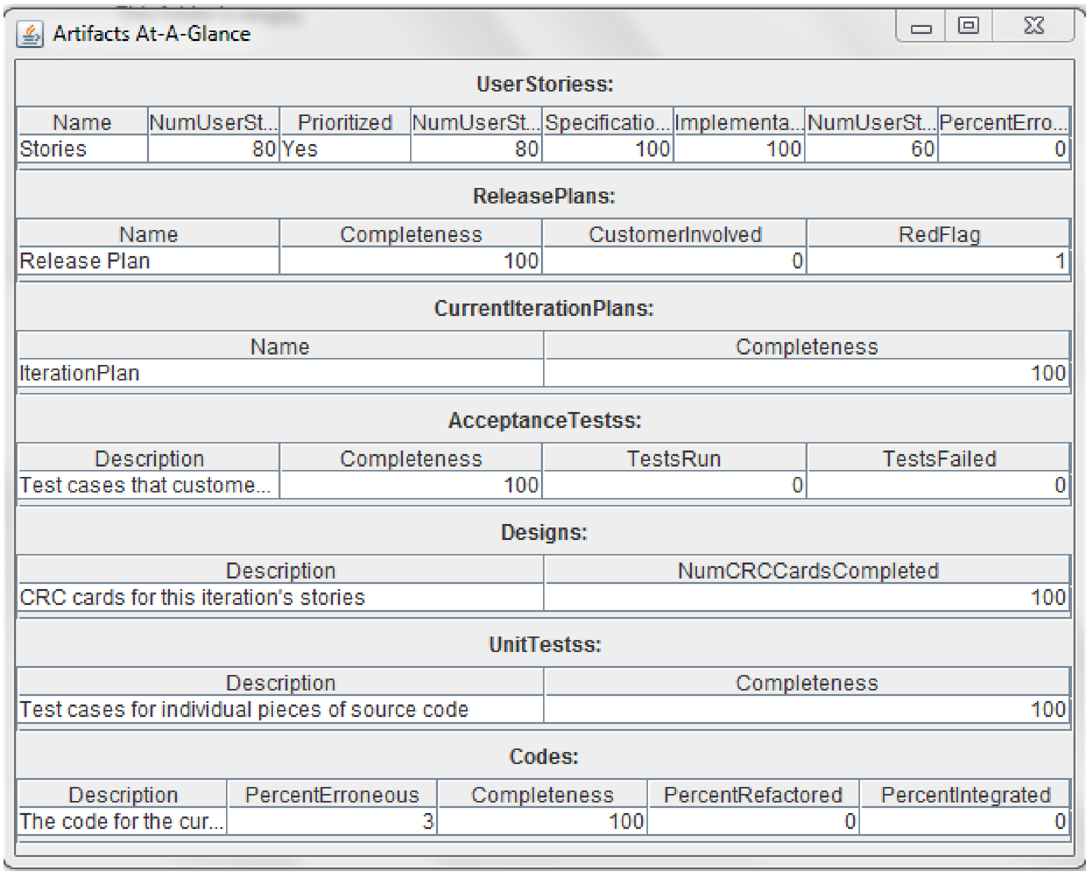
C2.6: Worst Case

This is the scenario where the user does not select the customer in the release planning meeting and keeps on ignoring the warning sign and does not select the ‘Involve Customer’ action throughout the game. Hence for this case, release 1 was completed at 573 clock ticks, release 2 at 1016 and release 3 at 1459.

Due to the deadline of 1800 clock ticks release 4 was never fully completed. It is important to note that the time taken for releases 1, 2 and 3 by the 'fourth iteration' case and worst case is the same but there is a difference between the completeness and correctness of the code (or product). Customer complained thrice throughout the project.

	Release 1	**Release 2**	**Release 3**	**Release 4**
Worst Case	573	1016	1459	-

Table C.12: Clock ticks representing the completion of releases for 'Worst case'


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementa...	NumUserSt...	PercentErro...
Stories	80	Yes	80	100	100	60	0

ReleasePlans:

Name	Completeness	CustomerInvolved	RedFlag
Release Plan	100	0	1

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that custome...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the cur...	3	100	0	0

Figure C.21: Final state of the artefacts for 'Worst case'

Here the percent refactored in fourth iteration has dropped to 0 compared to 89 in the previous case. This is a worse ending situation than the previous example. Under ReleasePlans, the CustomerInvolved=0 indicate that the customer representative was never involved in the release planning meeting.

RedFlag=1 indicate that the warning message appeared during the XP game play.

C3: EWS3 - Project team members do not have required knowledge/skills

C3.1: Best Case

Best case is the scenario where the developers are made aware of the coding standard immediately (before the actual coding starts). This is done by selecting the ‘Learn coding standard’ action for all the developers. Here no warning message appears as the users selects ‘Learn coding standard’ action immediately (being a proactive user). Best case represents an ideal project where everything goes smoothly and the project is completed successfully before the deadline. The detail of the best case is exactly same as explained earlier. In order to remove the repetition the author is not including the same details here. Please refer to appendix C1.1 for inspection of detailed results.

C3.2: EWS Case

EWS case is the scenario where on encountering the warning message the user selects ‘Learn coding standard’ action. Because the user acted so spontaneously the outcomes of this scenario is exactly similar to the best case. The only difference between the best and the EWS case is that in the best case warning message never pops up whereas in EWS case the warning message appears once. The detail of the EWS case is exactly same as explained earlier. In order to remove the repetition the author is not including the same details here. Please refer to appendix C1.2 for inspection of detailed results.

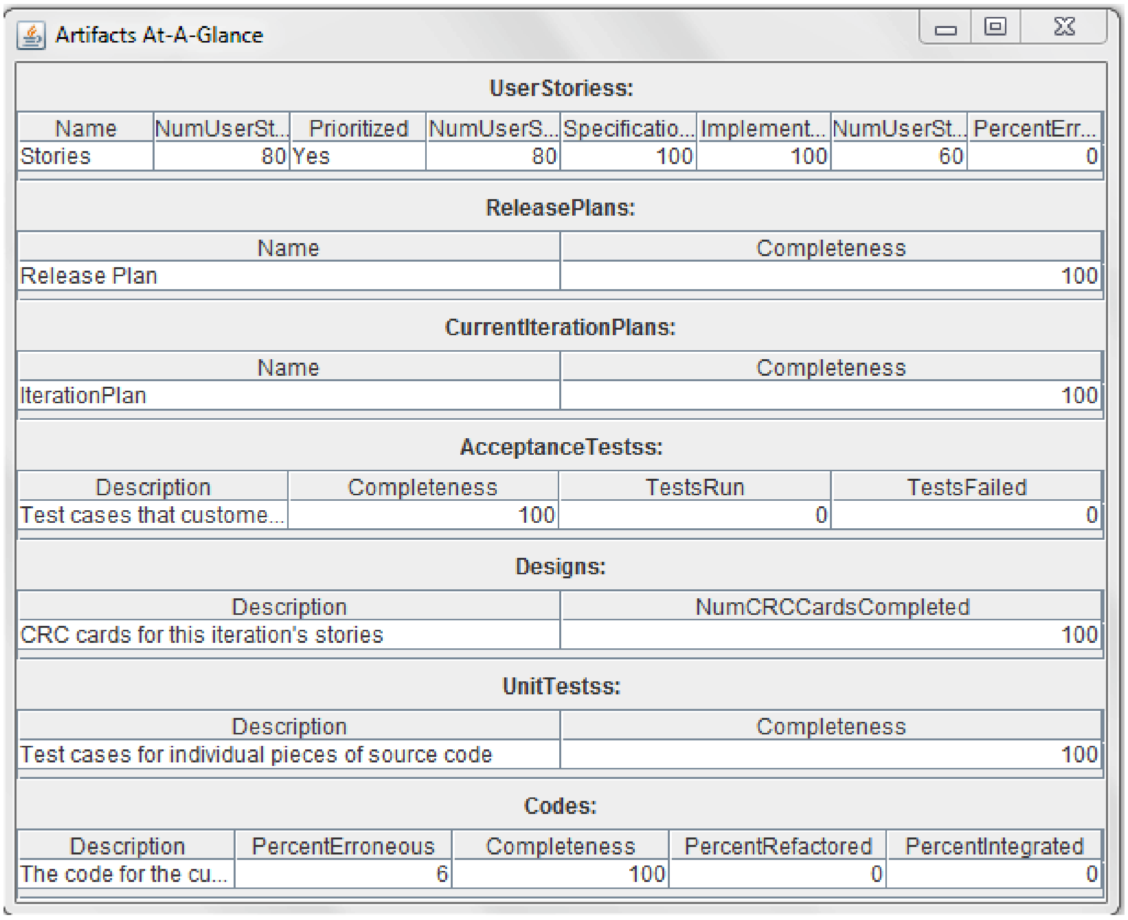
C3.3: Second Iteration

This is the scenario where the user keeps on ignoring the warning message and selects the ‘Learn coding standard’ option only at the start of the second iteration. But some time has already been lost by the project. Hence for this case, release 1 was completed at 655 clock ticks, release 2 at 1088 and release

3 at 1501. Due to the deadline of 1800 clock ticks release 4 was never fully completed. Customer complained thrice throughout the project.

	Release 1	Release 2	Release 3	Release 4
Second Iteration	655	1088	1501	-

Table C.13: Clock ticks representing the completion of releases for ‘Second Iteration’ case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserS...	Specificatio...	Implement...	NumUserSt...	PercentErr...
Stories	80	Yes	80	100	100	60	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that custome...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the cu...	6	100	0	0

Figure C.22: Final state of the artefacts for ‘Second Iteration’ case

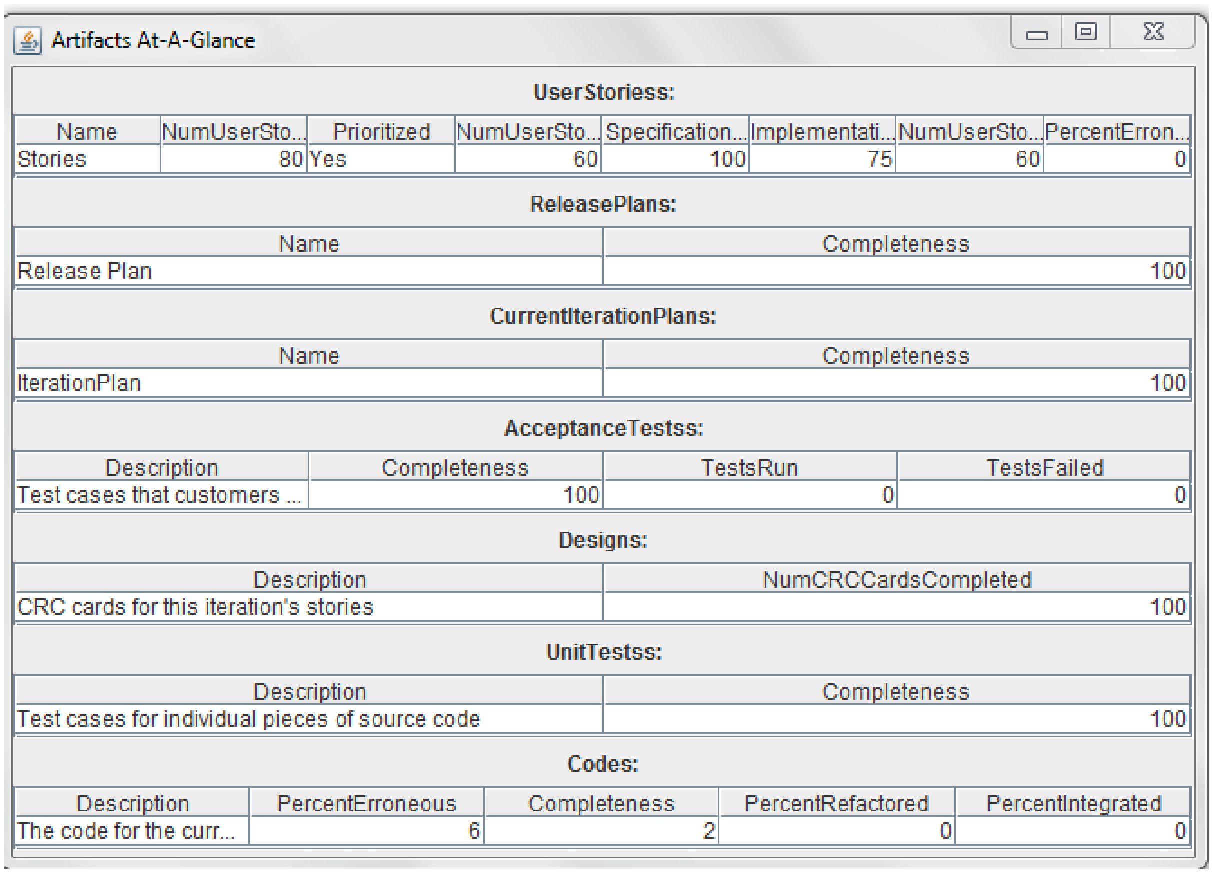
You can observe that percent erroneous in the 4th iteration has crept to 6% from 0%. Also, the percent refactored and percent integrated was 0% compared to 100% in the previous case. Further, the total number of user stories integrated was only 60 (compared to 80 in the previous case). Also, acceptance testing was not even started (TestsRun is 0). This is a worse ending situation than the previous example.

C3.4: Third Iteration

This is the scenario where the user keeps on ignoring the warning message and selects the ‘Learn coding standard’ option only at the start of the third iteration. But some time has already been lost by the project. Hence for this case, release 1 was finished at 655 clock ticks, release 2 at 1200 and release 3 at 1633. Due to the deadline of 1800 clock ticks release 4 was never fully completed. Customer complained six times throughout the project.

	Release 1	**Release 2**	**Release 3**	**Release 4**
Third Iteration	655	1200	1633	-

Table C.14: Clock ticks representing the completion of releases for ‘Third Iteration’ case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSto...	Prioritized	NumUserSto...	Specification...	Implementati...	NumUserSto...	PercentErron...
Stories	80	Yes	60	100	75	60	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers ...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curr...	6	2	0	0

Figure C.23: Final state of the artefacts for ‘Third Iteration’ case

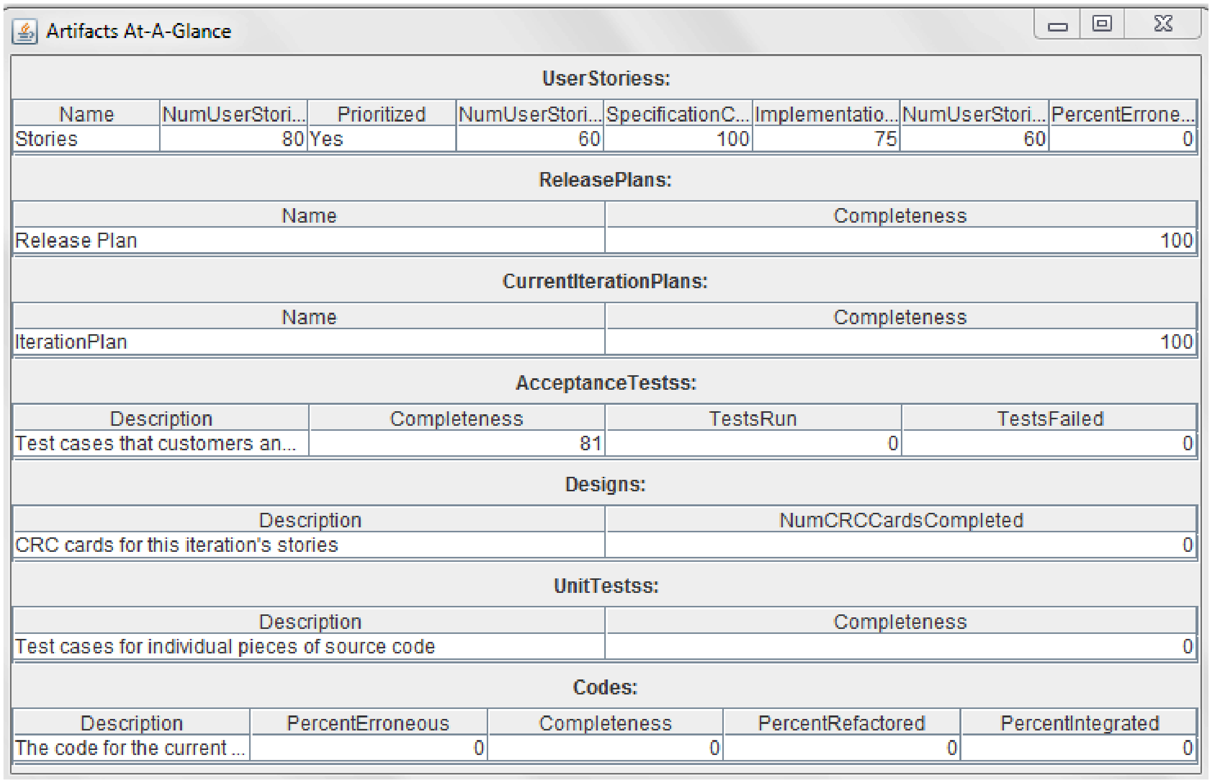
Here the code completeness is only 2% (compared to 100% in the previous case). Also, as the coding was not completed the implementation completeness is only 75% compared to 100%. This is a worse ending situation than the previous example.

C3.5: Worst Case

This is the scenario where the user keeps on ignoring the warning message and never selects the ‘Learn coding standard’ option. This means that the developers have done the coding for the project without knowing the coding standard. Hence for this case, release 1 was finished at 655 clock ticks, release 2 at 1200 and release 3 at 1745. Due to the deadline of 1800 clock ticks release 4 was never fully completed. Customer complained nine times throughout the project.

	Release 1	Release 2	Release 3	Release 4
Worst Case	655	1200	1745	-

Table C.15: Clock ticks representing the completion of releases for 'Worst case'


Artifacts At-A-Glance

UserStoriess:

Name	NumUserStori...	Prioritized	NumUserStori...	SpecificationC...	Implementatio...	NumUserStori...	PercentErrone...
Stories	80	Yes	60	100	75	60	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers an...	81	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	0

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	0

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the current ...	0	0	0	0

Figure C.24: Final state of the artefacts for 'Worst case'

If we compare with the previous case, we can observe that acceptance tests creation completeness is only 81% and designing and unit tests creation were not even started. This is a worse ending situation than the previous example.

C4: EWS4 - Key project stakeholders do not participate in major review meetings

C4.1: Best Case

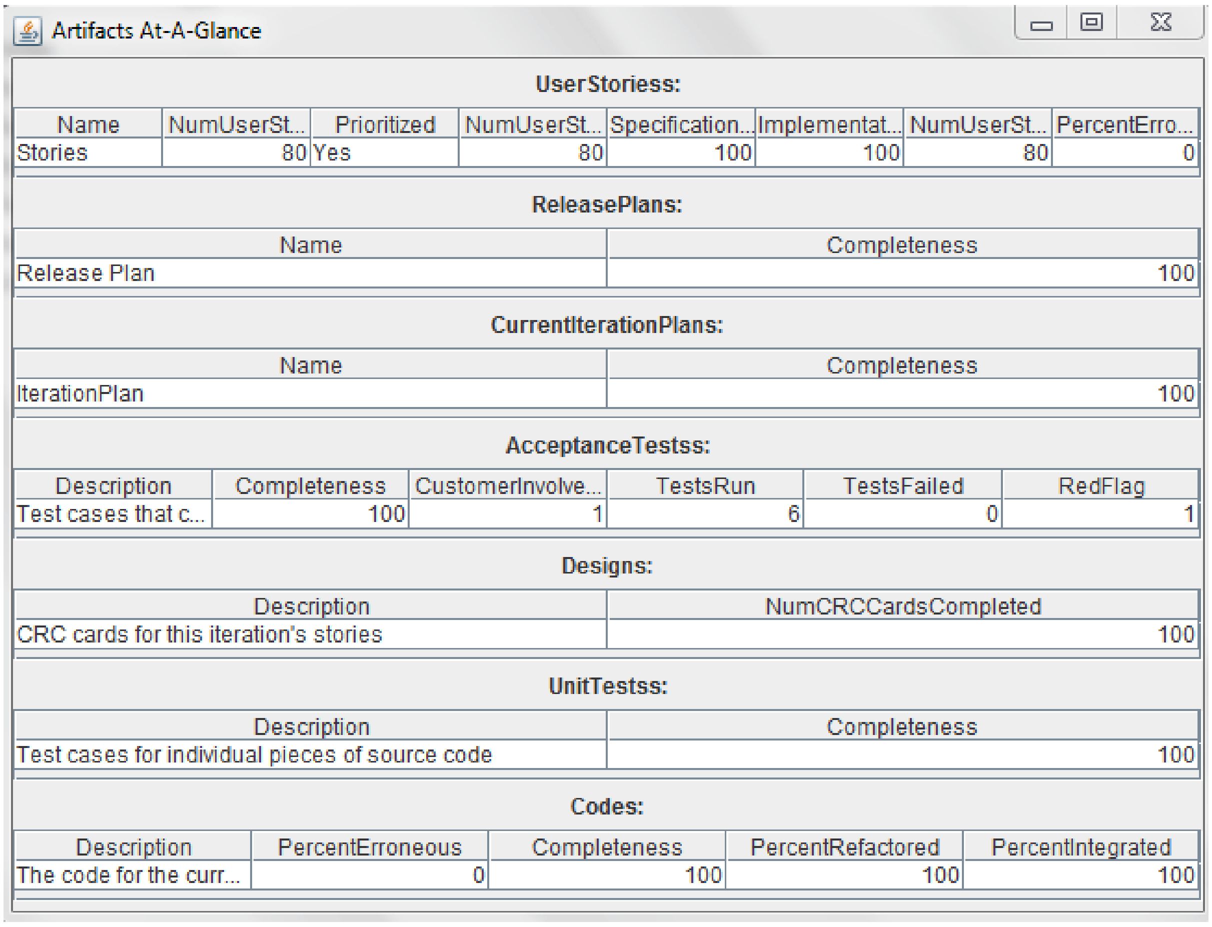
As already discussed in the previous cases, best case is the scenario where the user involves the customer representative (customer Wayne) in the creation of acceptance tests at first go. Here no warning message appears. Best case represents an ideal project. The detail of the best case is exactly same as explained earlier. In order to remove the repetition the author is not including the same details here. Please refer to appendix C1.1 for inspection of detailed results.

C4.2: EWS Case

EWS case is the scenario where the user does not involve customer in the creation of the acceptance tests but on encountering the warning message, immediately selects 'Involve Customer' action. The user responds to the first warning message. For the EWS case, release 1 was finished at clock tick 552; release 2 at 974 and release 3 at 1396. Release 4 was not fully completed. Customer complained 5 times throughout the project.

	Release 1	**Release 2**	**Release 3**	**Release 4**
EWS Case	552	974	1396	-

Table C.16: Clock ticks representing the completion of releases for EWS case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specification...	Implementat...	NumUserSt...	PercentErro...
Stories	80	Yes	80	100	100	80	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	CustomerInvolve...	TestsRun	TestsFailed	RedFlag
Test cases that c...	100	1	6	0	1

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curr...	0	100	100	100

Figure C.25: Final state of the artefacts for EWS case

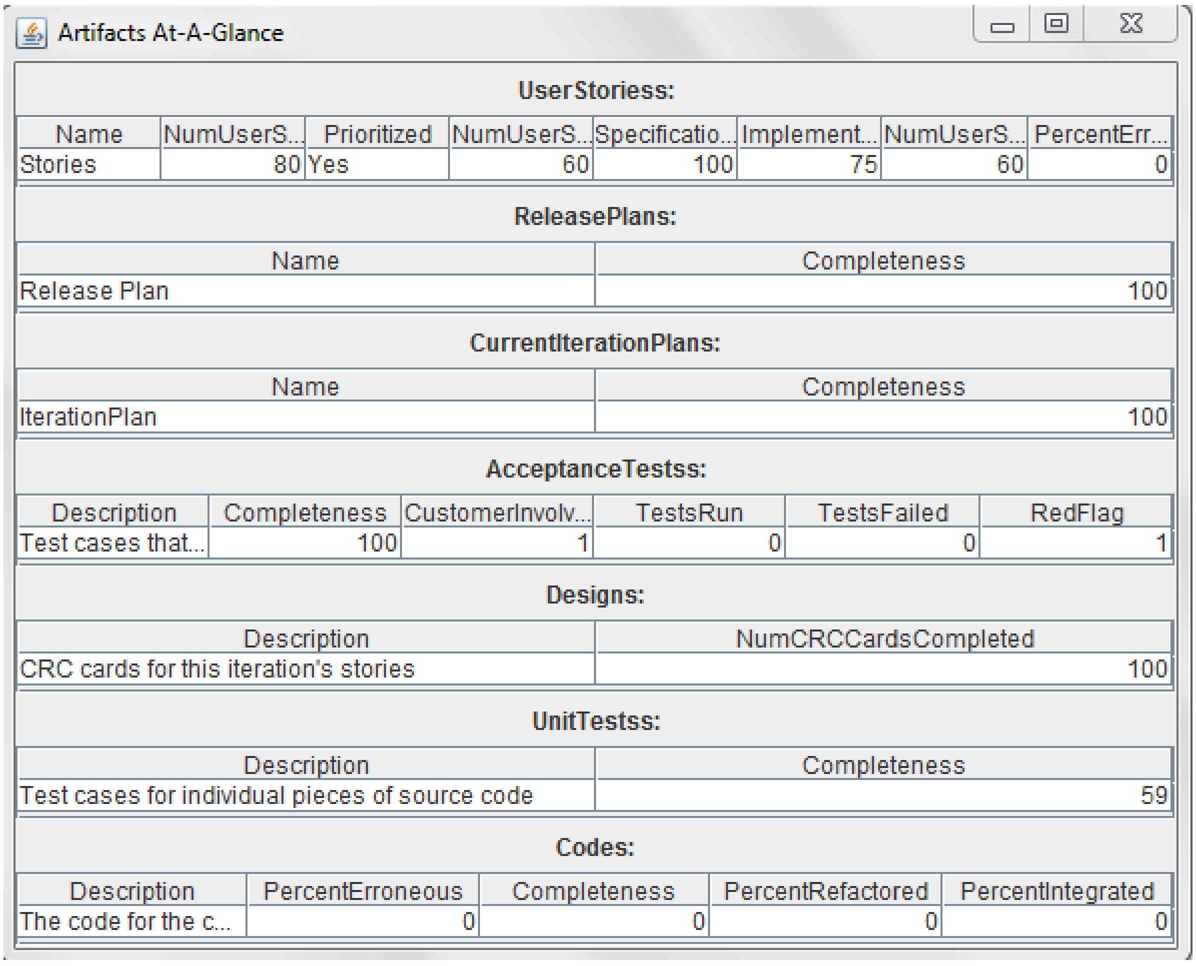
It appears that percent refactored and percent integrated are 100% and acceptance tests run is 6. The 4th iteration was almost about to end was just waiting to finish all the acceptance tests. Under AcceptanceTests, the CustomerInvolvedInCreation=1 indicates that the customer representative was involved in the creation of acceptance tests. RedFlag=1 indicate that the warning message appeared during the XP game play.

C4.3: Second Iteration

This is the scenario where the user does not involve customer in the creation of the acceptance tests until the second iteration. For this case, release 1 was finished at clock tick 800; release 2 at 1222 and release 3 at 1644. Release 4 was not fully completed. Customer complained 5 times throughout the project.

	Release 1	Release 2	Release 3	Release 4
Second Iteration	800	1222	1644	-

Table C.17: Clock ticks representing the completion of releases for 'Second Iteration' case



Artifacts At-A-Glance

UserStoriess:

Name	NumUserS...	Prioritized	NumUserS...	Specificatio...	Implement...	NumUserS...	PercentErr...
Stories	80	Yes	60	100	75	60	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	CustomerInvolv...	TestsRun	TestsFailed	RedFlag
Test cases that...	100	1	0	0	1

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	59

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the c...	0	0	0	0

Figure C.26: Final state of the artefacts for 'Second Iteration' case

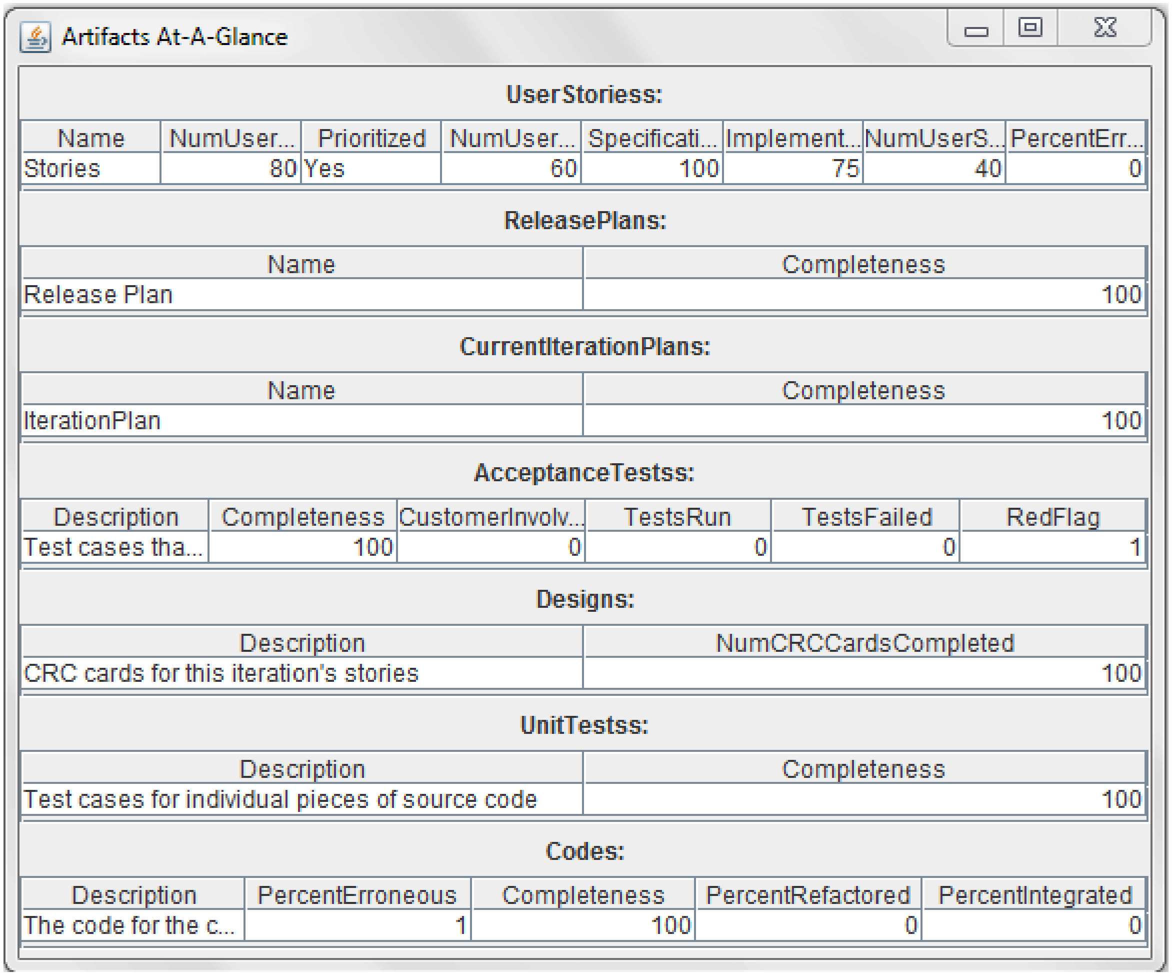
Compared to the previous case, the unit tests creation completeness in the 4th iteration dropped to 59% from 100%. The coding wasn't even started and hence the implementation completeness and number of user stories integrated dropped from 100 and 80 to 75 and 60 respectively. This is a worse ending situation than the previous example. Under AcceptanceTests, the CustomerInvolvedInCreation=1 indicates that the customer representative was involved in the creation of acceptance tests. RedFlag=1 indicate that the warning message appeared during the XP game play.

C4.4: Worst Case

This is the scenario where the user does not involve customer in the creation of the acceptance tests at all. The user repetitively ignores the warning message. For this case, release 1 was finished at clock tick 800 and release 2 at 1470. Release 3 was not fully completed. Customer complained 10 times throughout the project.

	Release 1	Release 2	Release 3	Release 4
Worst Case	800	1470	-	-

Table C.18: Clock ticks representing the completion of releases for ‘Worst case’


Artifacts At-A-Glance

UserStoriess:

Name	NumUser...	Prioritized	NumUser...	Specificati...	Implement...	NumUserS...	PercentErr...
Stories	80	Yes	60	100	75	40	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	CustomerInvolv...	TestsRun	TestsFailed	RedFlag
Test cases tha...	100	0	0	0	1

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the c...	1	100	0	0

Figure C.27: Final state of the artefacts for ‘Worst case’

In the previous case, the user managed to finish three iterations and was on the 4th iteration when the game ended but here the user only managed to finish first two iterations and was on the 3rd iteration when the game ended. Hence the above attribute values represent for iteration 3. The total number of user stories integrated was 40 (compared to 60 in the previous case). (Please note - the unit tests creation in this case is 100% complete compared to 59% in the previous case but that was for 4th iteration and this one is for 3rd iteration. These values are interpreted in light of the total number of iterations completed). This is a worse ending situation than the previous example. Under AcceptanceTests, the CustomerInvolvedInCreation=0 indicates that the customer representative was never involved in the creation of acceptance tests. RedFlag=1 indicate that the warning message appeared during the XP game play.

C5:EWS5 - Project manager(s) cannot effectively lead the team

C5.1: Best Case

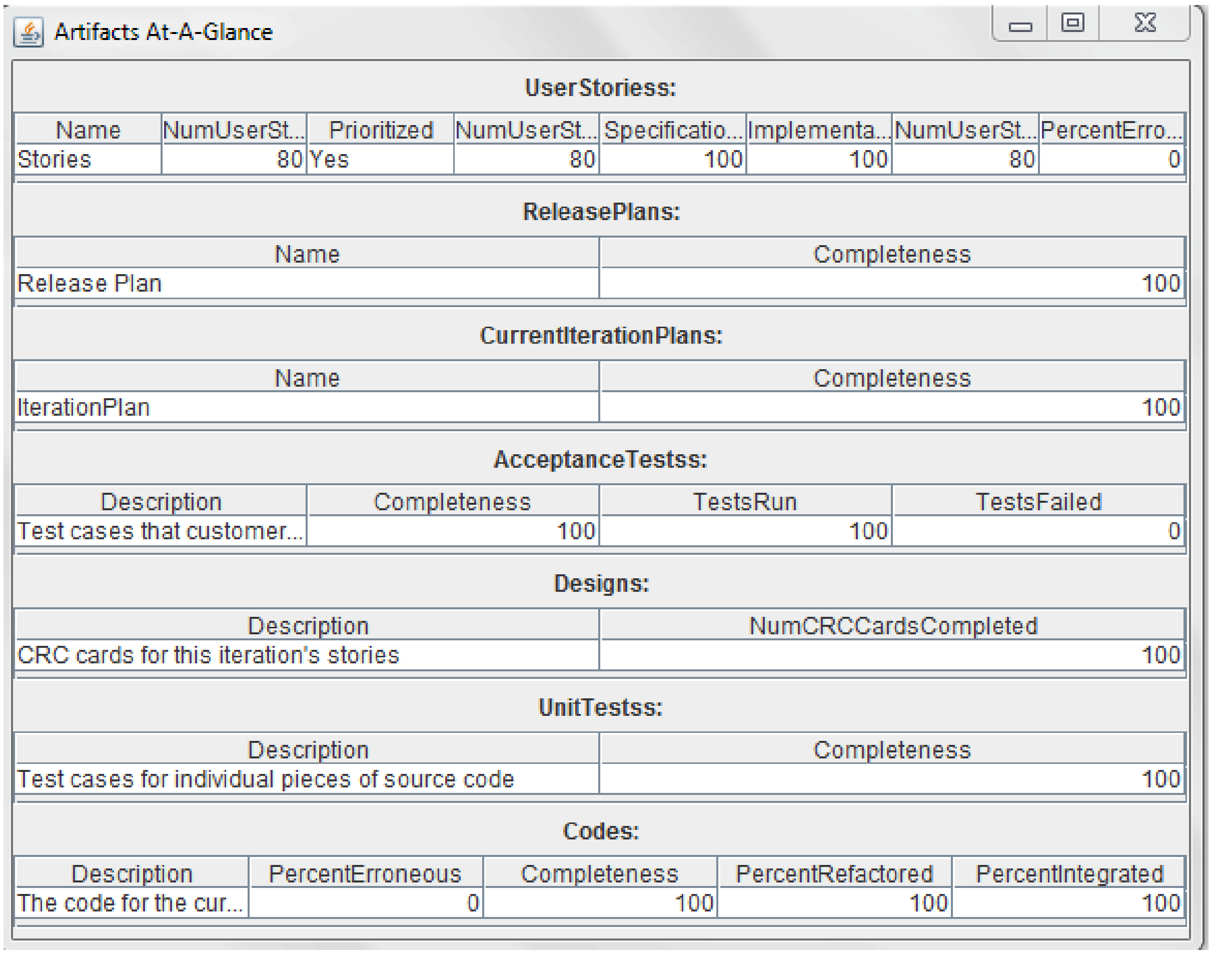
As already discussed in the previous cases, best case is the scenario where the employees are always assigned with work and hence they are never idle. Here no warning message appears. Best case represents an ideal project. The detail of the best case is exactly same as explained earlier. In order to remove the repetition the author is not including the same details here. Please refer to appendix C1.1 for inspection of detailed results.

C5.2: EWS Case

EWS case is the scenario where the user misses to assign work to the employees and hence early warning message pops up. But on encountering the warning message, the user immediately assigns work to the team. The user responds to the first warning message. For the EWS case, release 1 was done at clock tick 549; release 2 at 964 and release 3 at 1379 and release 4 at 1799.

	Release 1	Release 2	Release 3	Release 4
EWS Case	549	964	1379	1799

Table C.19: Clock ticks representing the completion of releases for EWS case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementa...	NumUserSt...	PercentErro...
Stories	80	Yes	80	100	100	80	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customer...	100	100	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the cur...	0	100	100	100

Figure C.28: Final state of the artefacts for EWS case

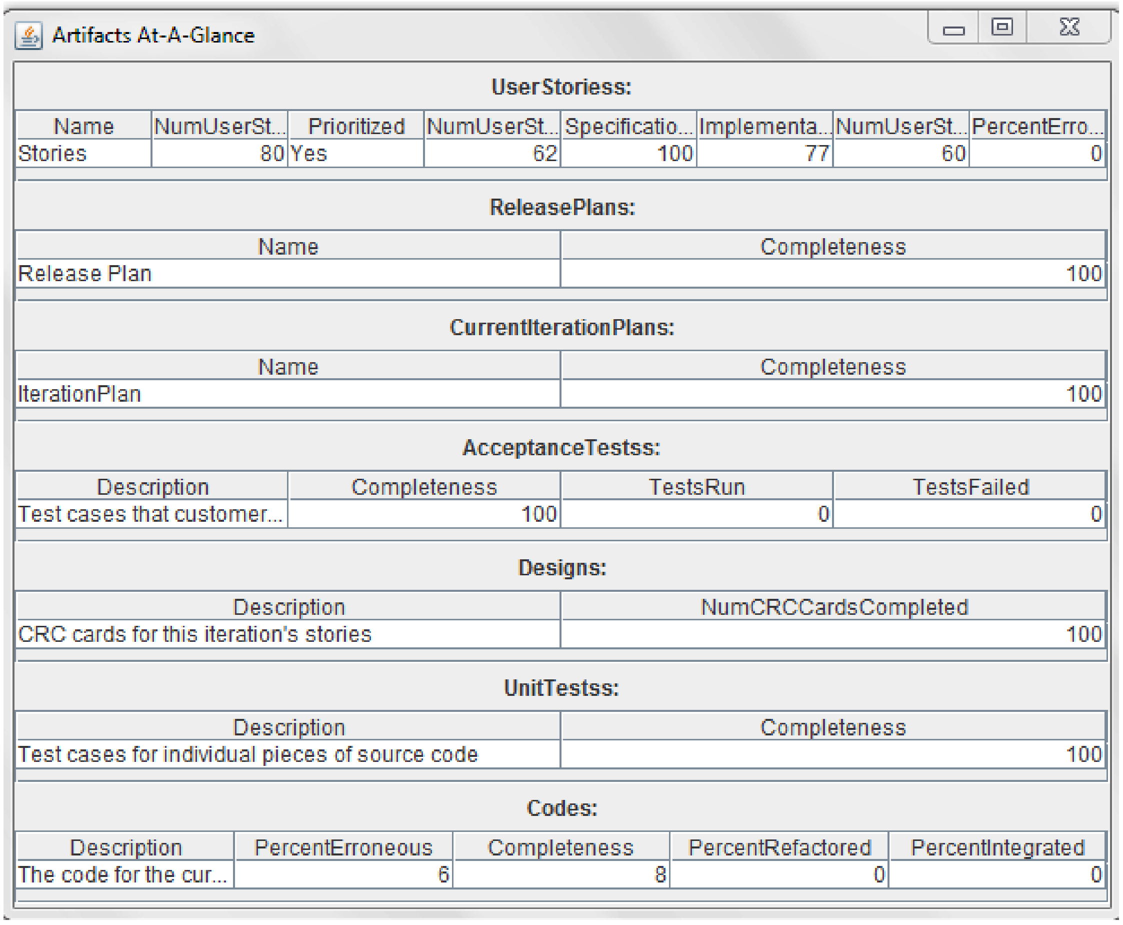
The only difference between the previous case and this case is the difference in the time for releases.

C5.3: Take 4

This is the scenario where the user misses to assign work to the employees and he reacts only when the warning message pops up for the fourth time. The user responds to the fourth warning message. For this case, release 1 was finished at clock tick 607; release 2 at 1084 and release 3 at 1561. Release 4 was not fully completed. Customer complained 3 times throughout the project.

	Release 1	Release 2	Release 3	Release 4
Take 4	607	1084	1561	-

Table C.20: Clock ticks representing the completion of releases for ‘Take 4’ case



UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specificatio...	Implementa...	NumUserSt...	PercentErro...
Stories	80	Yes	62	100	77	60	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customer...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	100

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the cur...	6	8	0	0

Figure C.29: Final state of the artefacts for ‘Take 4’ case

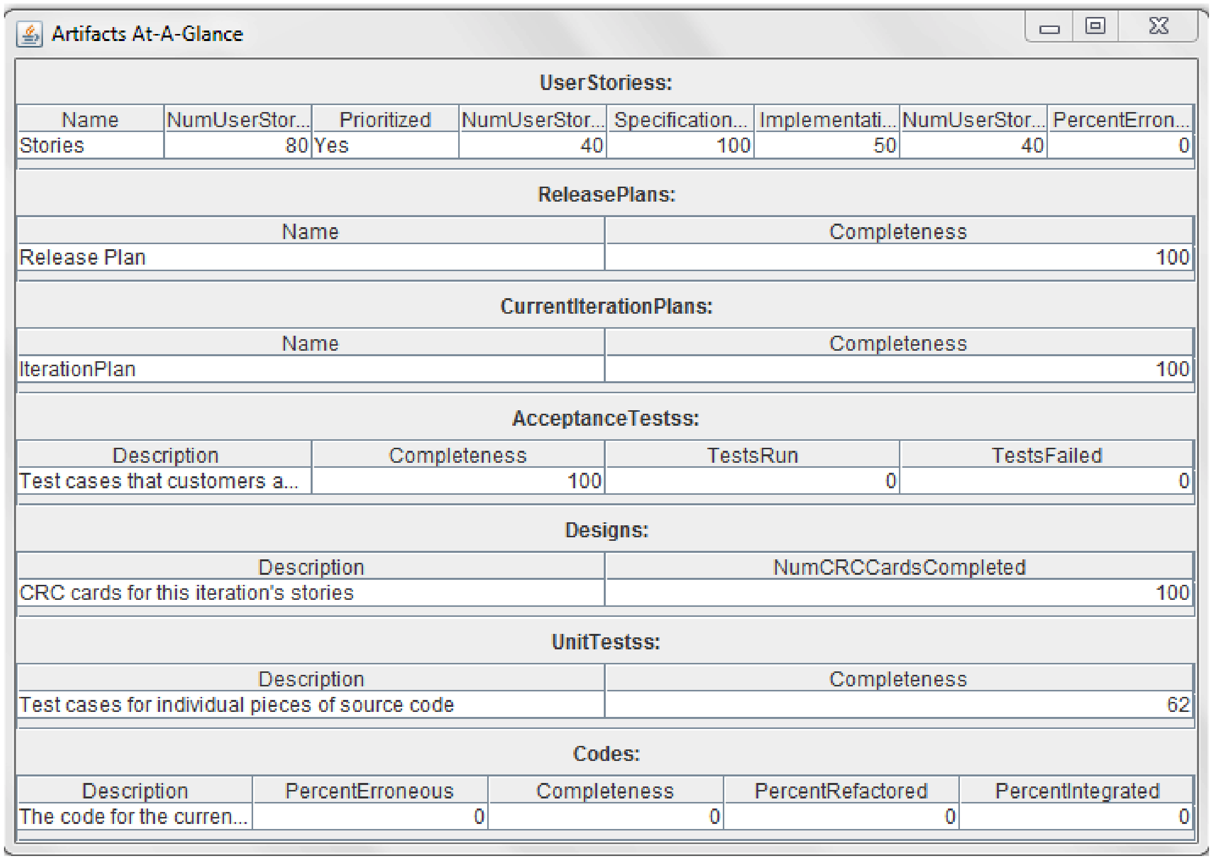
Compared to the previous case, the percent erroneous in the 4th iteration has increased to 6% from 0%. The code completeness is only 8%. The implementation completeness and total number of user stories integrated has dropped to 77 and 60 from 100 and 80 respectively. This is a worse ending situation than the previous example.

C5.4: Take 12

This is the scenario where the user misses to assign work to the employees and he reacts only when the warning message pops up for the 12th time. The user responds to the 12th warning message. For this case, release 1 was finished at clock tick 775; and release 2 at 1420. Release 3 was not fully completed. Customer complained 10 times throughout the project.

	Release 1	**Release 2**	**Release 3**	**Release 4**
Take 12	775	1420	-	-

Table C.21: Clock ticks representing the completion of releases for 'Take 12' case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserStor...	Prioritized	NumUserStor...	Specification...	Implementati...	NumUserStor...	PercentErron...
Stories	80	Yes	40	100	50	40	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers a...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	100

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	62

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curren...	0	0	0	0

Figure C.30: Final state of the artefacts for 'Take 12' case

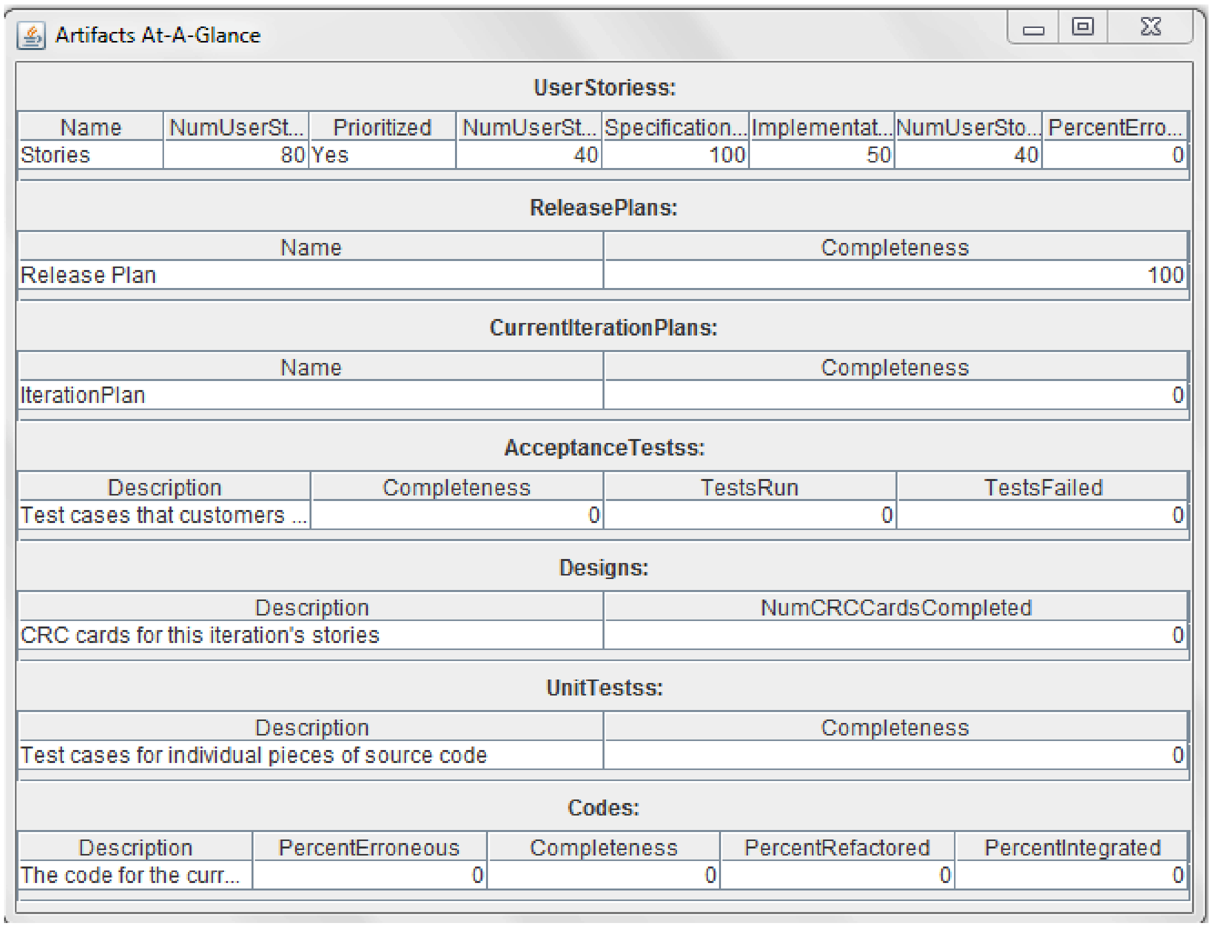
In the previous case, the user managed to finish three iterations and was on the 4th iteration when the game ended but here the user only managed to finish first two iterations and was on the 3rd iteration when the game ended. Hence the above attribute values represent for iteration 3. The implementation completeness and total number of user stories integrated has dropped from 77 and 60 to 50 and 40 respectively. This is a worse ending situation than the previous example.

C5.5: Take 20

This is the scenario where the user misses to assign work to the employees and he reacts only when the warning message pops up for the 20th time. The user responds to the 20th warning message. For this case, release 1 was finished at clock tick 943; and release 2 at 1777. Release 3 was not fully completed. Customer complained 16 times throughout the project.

	Release 1	**Release 2**	**Release 3**	**Release 4**
Take 20	943	1777	-	-

Table C.22: Clock ticks representing the completion of releases for 'Take 20' case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserSt...	Prioritized	NumUserSt...	Specification...	Implementat...	NumUserSto...	PercentErro...
Stories	80	Yes	40	100	50	40	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	0

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers ...	0	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	0

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	0

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the curr...	0	0	0	0

Figure C.31: Final state of the artefacts for 'Take 20' case

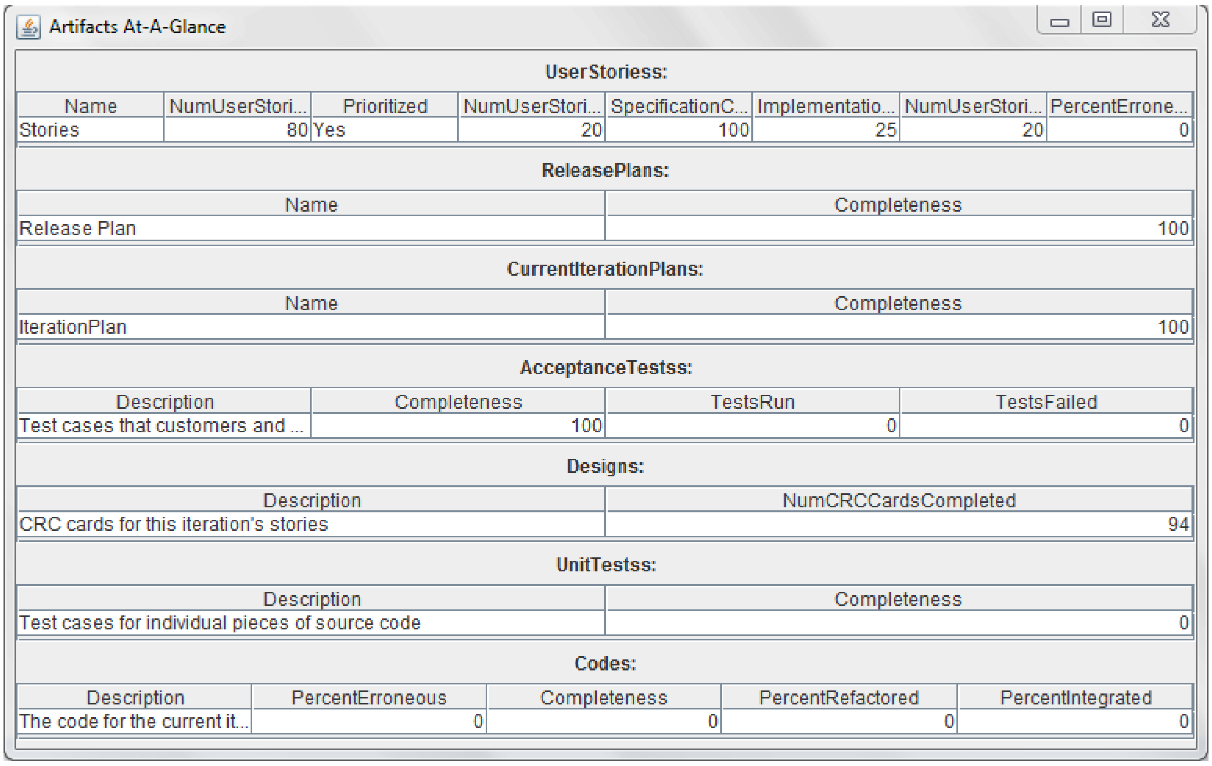
In the previous case, in the 3rd iteration the current iteration plan completeness, acceptance tests completeness and design completeness were 100; and unit tests creation completeness was 62%. Whereas in this case, in the 3rd iteration the current iteration plan completeness, acceptance tests completeness, design completeness and unit tests creation completeness were 0. This is a worse ending situation than the previous example.

C5.6: Take 28

This is the scenario where the user misses to assign work to the employees and he reacts only when the warning message pops up for the 28th time. The user responds to the 28th warning message. For this case, release 1 was finished at clock tick 1112. Release 2 was not fully completed. Customer complained 16 times throughout the project.

	Release 1	**Release 2**	**Release 3**	**Release 4**
Take 28	1112	-	-	-

Table C.23: Clock ticks representing the completion of releases for 'Take 28' case


Artifacts At-A-Glance

UserStoriess:

Name	NumUserStori...	Prioritized	NumUserStori...	SpecificationC...	Implementatio...	NumUserStori...	PercentErrone...
Stories	80	Yes	20	100	25	20	0

ReleasePlans:

Name	Completeness
Release Plan	100

CurrentIterationPlans:

Name	Completeness
IterationPlan	100

AcceptanceTestss:

Description	Completeness	TestsRun	TestsFailed
Test cases that customers and ...	100	0	0

Designs:

Description	NumCRCCardsCompleted
CRC cards for this iteration's stories	94

UnitTestss:

Description	Completeness
Test cases for individual pieces of source code	0

Codes:

Description	PercentErroneous	Completeness	PercentRefactored	PercentIntegrated
The code for the current it...	0	0	0	0

Figure C.32: Final state of the artefacts for 'Take 28' case

In the previous case, the user managed to finish two iterations and was on the 3rd iteration when the game ended but here the user only managed to finish first iteration and was on the 2nd iteration when the game ended. Hence the above attribute values represent for iteration 2. The implementation completeness and total number of user stories integrated has dropped from 50 and 40 to 25 and 20 respectively. This is a worse ending situation than the previous example.