ANUKARNA : A Software Engineering Simulation Game for Teaching Practical Decision Making in Peer Code Review

Ritika Atal

Computer Science

Indraprastha Institute of Information Technology, Delhi (IIIT-D), India

A Thesis Report submitted in partial fulfilment for the degree of

MTech Computer Science

18 July 2015

1. Prof. Ashish Sureka (Thesis Adviser)

2. Prof. Debajyoti Bera (Internal Examiner)

3. Mr. Ravindra Naik (External Examiner)

Day of the defense: 18 July 2015

Signature from Post-Graduate Committee (PGC) Chair:

Abstract

Software Engineering (SE) being a practice-oriented and applied field is taught primarily (at University Level) using instructor-driven classroom lectures as well as team-based projects requiring hands-on skills. In comparison to classroom lectures, team-based hands-on projects require more active participation and experiential learning. SE educators have proposed and shown positive student learning outcomes by teaching certain concepts using simulation games. Some of the advantages of teaching using simulation games are incorporation of real-world dynamics such as critical decision making under multiple and conflicting goals, encountering unexpected and unanticipated events, allowing exploration of alternatives (discovery learning) and allowing incorporation of learning through doing and failure. We develop a web-based interactive educational SE simulation game or environment for teaching benefits and best-practices of peer-code review process. We describe a learning framework and model based on discovery learning, learning from failure, evidence and reasoning for teaching concepts on the practice of peer code review. Finally we evaluate the proposed learning framework and tool by conducting experiments and collecting feedback from users and present the results of our evaluation.

This thesis work is dedicated to my parents and brother for their endless love and encouragement. I have always turned to them for support and strength. I take comfort in knowing that no matter what path I choose, my family stands behind me.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Dr. Ashish Sureka, whose valuable support and consistent guidance has helped me complete my thesis. His work ethics and thought process has motivated and inspired me at each step of this journey and taught me innumerable lessons. If it wasn't for him, it would not have been possible for me to finish this work successfully.

Besides my advisor, I would like to deeply thank my esteemed committee members Prof. Debajyoti Bera and Mr. Ravindra Naik for agreeing to evaluate my thesis.

Last but not the least, I would like to thank all my family members especially my brother Deepak Atal, who encouraged and kept me motivated throughout the thesis.

Declaration

This is to certify that the thesis titled **"Anukarna: A Software Engineering Simulation Game for Teaching Practical Decision Making in Peer Code Review"** submitted by **Ritika Atal** for the partial fulfillment of the requirements for the degree of *Master of Technology* in *Computer Science & Engineering* is a record of the bonafide work carried out by her under my guidance and supervision at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Professor Ashish Sureka
Indraprastha Institute of Information Technology, New Delhi

Contents

List of Figures

List of Tables

1

Research Motivation and Aim

1.1 Peer Code Review: An overview

Peer code review is a staple of the software industry. It is one way that practising professionals continually increase their technical skills. If done thoughtfully and properly, it can also drastically improve the quality of code that is generated [2]. Figure 1.1 presents a software process including code review. During the code phase the developer writes the software code, and it is the phase where defects are injected. Following it, is the code review phase and test phase where the defects are found and fixed. These are all coding defects but there is a good chance that some of the defects are architecture defects, some are requirements defects, some are design defects etc. Thus code reviews require a good planning at each stage of software life cycle to achieve good results.

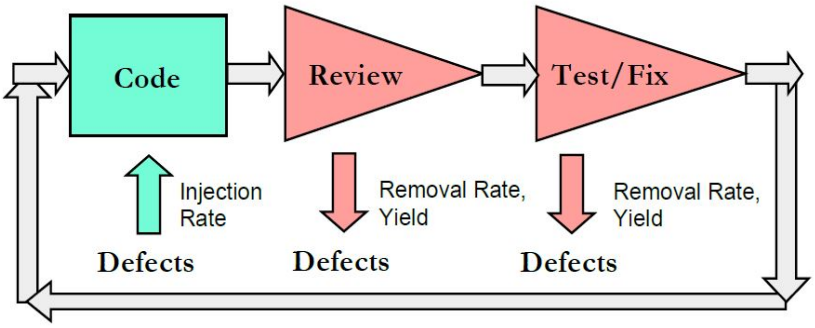


Figure 1.1: Software process including a code review

Code reviews educate novice developers to not make mistakes that veterans would

avoid. It helps keep code base maintainable over a long time. Every defect found in code review is another bug that might have gotten through Quality Assurance and into the hands of a customer. It provides a competitive edge to organisations. The cost of losing market position is unthinkably large, so the cost of every defect is similarly large. Design documentation is traditionally the first place when peer review occurs in software development organizations. The importance of getting the conceptual design right is widely recognized and practiced. Few developers and software project managers would argue that these are benefits of conducting code reviews [3]:

1. Improved code quality
2. Fewer defects in code
3. Education of junior programmers
4. Shorter development/test cycles
5. Reduced impact on technical support
6. More customer satisfaction
7. More maintainable code

Code review is an important practice which is carried out in all the organizations but there is no existing strategic training module or processes which embed these practices among the developers, that is where Software Process Simulation comes into play. Software process simulation modeling can be used to address this issue of preparing students or employers for successful software reviews and code inspection by making them recognize their responsibilities and expected work behavior via a simulation. Simulation game helps in developing a well prepared and trained work force along with the monetary and product quality benefits.

1.1.1 Types of Code Review Processes

Following are the two main categories of code review processes [3] :

1.1.1.1 Heavy Weight Code Review Process

Following are the process which fall in category of heavy weight code review processes.

1. Formal inspection

   Formal reviews are normally called inspections. This method was introduced by Michael Fagan [4]. It requires 4 to 6 participants, holding meetings to discuss the code and find defects and each defect is recorded in great details. It is a time consuming review process as it requires a lot of developer's time in holding meetings time and again. But most organizations cannot afford to tie up that many people for that long.

2. Over-the-shoulder reviews

   This is the type of review which requires reviewer to perform the review in presence of developer at his workstation. This type requires author drive the review by sitting at the keyboard and mouse, opening various files, pointing out the changes and explaining why it was done this way. It is simple in execution and does not require training. But there are no metrics, reports, or tools that measure anything at all about the process.

3. E-mail pass-around reviews

   In e-mail pass around, whole files or changes are packed up by the author and sent to reviewers via e-mail. Reviewers examine the files, ask questions and discuss with the author and other developers, and suggest changes. The hardest part of the e-mail pass-around is in finding and collecting the files under review.

4. Pair-Programming

   Pair-programming is two developers writing code at a single workstation with only one developer typing at a time and continuous free-form discussion and review. The reviewing developer is deeply involved in the code, which might cause trouble later as reviewer too close to the code cannot step back and critique it from a fresh and unbiased.

1.1.1.2 Light Weight Code Review Process

Light weight code review or tool assisted reviews, refers to any process where specialized tools are used in all aspects of the review: collecting files, transmitting and displaying files, commentary and defects among all participants, collecting metrics, and giving product managers and administrators some control over the workflow. Eg- Gerrit (Figure 1.2), Reitveld (Figure 1.3), Review Board (Figure 1.4), etc.

Formal, or heavyweight, inspections have been around for 30 years and they are no longer the most efficient way. It is observed [3] that lightweight code reviews take 1/5th the time (or less) of formal reviews and they find just as many bugs!

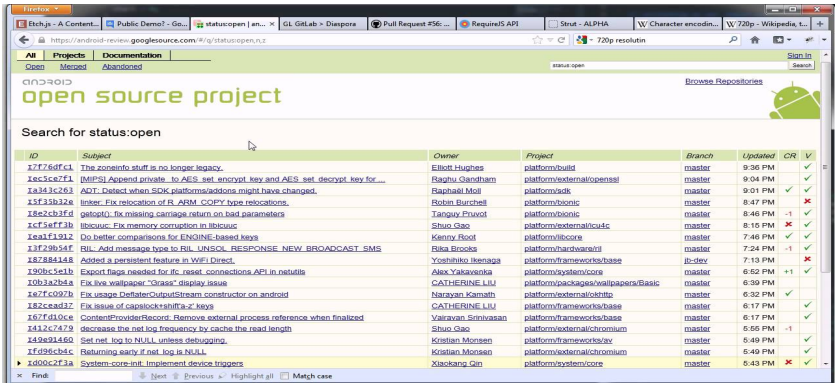


Figure 1.2: Snapshot of Gerrit software

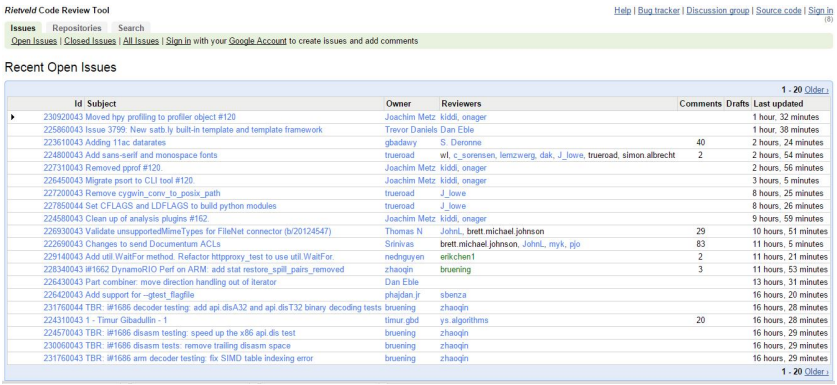


Figure 1.3: Snapshot of Rietveld software

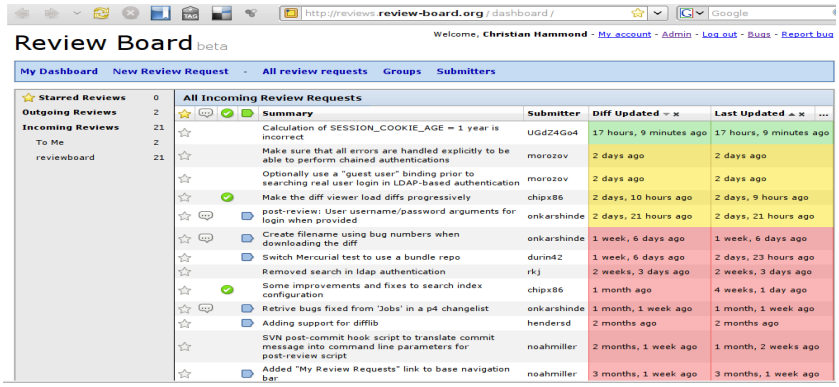


Figure 1.4: Snapshot of Review Board software

1.2 Simulation

Simulation is the process of recreating reality in virtual form. It reproduces the behaviour of a selected system or process. Simulation provides valuable hands-on experience without incurring the high cost of having to perform the actual exercise. It helps exploration of real effects or consequences on selecting a different course of action in a system. One can avoid the risk of dramatic consequences that may occur in case of failure, if it were to happen in reality. Unknown situations can be introduced and practiced while exploring the other alternative solutions. Simulation provides insight into behaviour of any system, which otherwise might be inaccessible, dangerous, too expensive or may not even exist. It's use is not restricted to only educational or training purposes. It's application vary from medical industry, aviation industry, entertainment industry, military etc. and many more.

Aviation industry makes use of simulator to train their pilots as they need to know how to behave in dangerous situations, but nobody would like to expose them to such risk just for the sake of training. Therefore, airlines provide powerful simulators that model the whole aircraft and its environment except for the pilot [5]. Using these simulators all difficulties can be experienced without any risk (though at considerable cost). We apply similar approach for teaching peer code review related practices to player, who acts as the project manager. The project needs to be completed within time and in budget, without compromising on the product quality and without accumulating high technical debt. Player's can thus recognize potentially troublesome situations that

they can confront in organisations, and for identify approaches with which to address such troublesome situations.

1.3 Software Engineering Education Models

Software Engineering (SE) being a practice-oriented and applied field is taught primarily (at University Level) using instructor-driven classroom lectures as well as team-based projects requiring hands-on skills. In comparison to classroom lectures, team-based hands-on projects require more active participation and experiential learning. Academia and industry have long debated the balance between theory and reality in engineering curricula [6]. Industrys perception that universities teach outdated methods and use impractical study examples is countered by the universities need to teach fundamentals before moving on to advanced, industry-relevant material.

While relevant process theory can be and typically is presented in lectures, the opportunities for students to practically and comprehensively experience the presented concepts are limited [7]. The focus strongly remains on creating project deliverable such as requirements documents, design documents, source code, and test cases, and little room is left to illustrate or experience the principles, pitfalls, and dimensions of the software project. SE educators have proposed and shown positive student learning outcomes by teaching certain concepts using simulation games. Simulation games help students analyze past situations and evaluate a different path that the project could have taken if specific decisions were made in particular points. The work presented in this paper is motivated by the need to teach the importance and best practices of peer code review to students using a simulation game.

1.4 Learning Models

Some of the advantages of teaching using simulation games are incorporation of real-world dynamics such as critical decision making under multiple and conflicting goals, encountering unexpected and unanticipated events, allowing exploration of alternatives (discovery learning) and allowing incorporation of learning through doing and failure [7][8].

1.4.1 Discovery Learning

Discovery Learning [9] is based on the idea that an individual learns a piece of knowledge most effectively if they discover it on their own, rather than having it explicitly told to them. This theory encourages educational approaches that are rich in exploring, experimenting, doing research, asking questions, and seeking answers. Educational software engineering simulation approaches [5] [7] are specifically designed to facilitate this type of learning i.e. no knowledge is made explicit in the simulation, as it is rather discovered by students experimenting with different approaches and seeing the effects of their decisions on the outcome of the simulation.

1.4.2 Learning by Failure

Learning Through Failure theory [9] is based on the assumption that the most memorable lessons are those that are learned as a result of failure. The theory argues that:

1. Learning through failure provides more motivation for students to learn, so as to avoid the adverse consequences that they experience first hand when they do not perform as taught, and

2. Failure engages students, as they are motivated to try again in order to succeed.

Proponents of the theory argue that students should be allowed to (and even set up to) fail to encourage maximal learning. In these approaches, the instructor purposely sets the students up for failure by introducing common real-world complications into projects (e.g., deadline preponement, attrition, crashing of hardware etc), the rationale being that students will then be prepared when these situations occur in their future careers.

1.5 Research Aim

The research aim of the work presented in the thesis is the following:

1. To develop a web-based interactive educational SE simulation game or environment for teaching benefits and best-practices of peer-code review process.

2. To investigate a learning framework and model based on discovery learning, learning from failure, evidence and reasoning for teaching concepts on the practice of peer code review.

3. To evaluate the proposed learning framework and tool by conducting experiments and collecting feedback from users.

2

Related Work and Research Contributions

In this chapter we discuss previous work closely related to our research and list the novel research contributions of our work in context to already existing work.

2.1 Related Work

SESAM (Software Engineering Simulation by Animated Models) by Drappa et al[5] is one of the first simulators developed for educational purposes in 2000. It is a software engineering simulation environment in which students manage a team of virtual employees to complete a virtual project on schedule, within budget, and at or above the required level of quality. It uses a highly flexible and expressive language, but its model building process is learning- and labor-intensive and requires writing code in a text editor (Figure 2.1). SESAM represents first example of a software process modeling language that is prescriptive, predictive, and interactive (but not graphical).

In 2004, SimSE an interactive, graphical, customizable simulation game for software engineering education was developed by Navarro et al[7]. SimSE attempts to address the lack of process education present in the typical software engineering course by providing students with a practical, high-level experience of a realistic software engineering process in an engaging manner. It allows students to practice a "virtual" software engineering process (or sub-process) in a fully graphical, interactive, and fun setting in which direct, graphical feedback enables them to learn the complex cause and effect

relationships underlying the processes of software engineering (Figure 2.2).

Same year another game i.e The Incredible Manager was introduced by Dantas et al[10]. It is a simulation game designed specifically to train software project managers. It is concentrated more on project management than on software processes. In essence, it is much like an industrial simulator with an added graphical, game-like user interface. The Incredible Manager (Figure 2.3) also allows for customization of its simulation models through a textual interface.

SimjavaSP is an interactive, web-based, graphical simulation game of the software development process which came in late 2005. The goal of SimjavaSP, is for the student, acting as the project manager, to develop a software project within the required time and budget, and of acceptable quality. Shaw et al[11] developed an interface for SimjavaSP which is a combination of graphical and textual feedback.

Same time Baker et al[12] introduced a card game rather than computer game named Problems and Programmers (Figure 2.4). It is a two player game designed to simulate a waterfall software development process from conception to completion. However, it only simulates one process (waterfall) and, being a simulation that involves physical objects (cards), it is significantly more difficult to customize than a computer-based simulation.

In 2006, SimVBSE a game-based simulation was specifically designed by Jain et al[13], to teach students the theory of value-based software engineering. SimVBSE has user interface which is fully graphical and includes animations and audio (Figure 2.5). The simulation portrays one real-world case study, and does not include facilities for customization.

ProMaSi (Project Management Simulator) by Petalidis et al[14] allows students to actively test their skills in managing the development of a software product exercising various actions through simulation. One of the novel approaches of the solution proposed here is the separation of the various constituent parts of the simulator through a modular architecture allowing third parties to provide their own add-ons.

Wangenheim et al[15] worked on DELIVER in year 2012, a board game to teach Earned Value Management in monitoring and controlling the execution of a software project. On the cognitive level, the learning objective of the game is to reinforce EVM concepts and to teach the competency to apply basic EVM calculations covering the cognitive levels remembering, understanding and application.

In an AMEISE (A Media Education Initiative for Software Engineering) simulation developed by Bollin et al[16] in 2013, students assume the role of a technical project manager. They are challenged to manage a project according to a particular model of the problem structure, selected by the instructor. It is up to the instructor to select the number of trials (simulation runs) to solve given tasks within specified constraints. Students can learn from previous simulation runs, change their strategies and measure their own success using the AMEISE (Figure 2.6) self-assessment feature.

Table 2.1 shows the result of our literature review. Table 2.1 lists the 9 papers mentioned above in reverse chronological order and reveals that teaching Software Engineering concepts using simulation games is an area that has attracted several researchers attention from the year 2000. We infer that teaching software engineering processes and project management are the two most popular target areas for simulation games.

2.2 Novel Research Contributions

In context to existing work and to the best of our knowledge, the study presented in this thesis makes the following novel contributions:

1. While there has been work done in the area of teaching Software Engineering processes and project management skills, our work is the first in the area of building and investigating simulation games for teaching best practices for peer code review.

2. We propose a simulation game to teach peer code review practices based on learning by failure, success and discovery. We demonstrate the effectiveness of our approach, discuss the strengths and limitations of our tool based on conducting user experiments and collecting their feedback.

Aug 10> *ask Patrick about his experience*
Patrick Murphy ponders over the question, before he answers:
"Well, I think my experience is average."
Aug 10> *let specify*
Who shall specify? *Patrick*
Patrick Murphy starts working on the specification.
Aug 10> *proceed 14 days*
Aug 24>*ask Patrick about his work*
Patrick Murphy reports: The specs are now 30 pages long, I think it will soon be finished
Aug 24> *proceed 21 days*
The customer has called, he has received a 63-pages specs on Sep 7. He will check it.
Sep 14>

Figure 2.1: Snapshot of SESAM simulation game

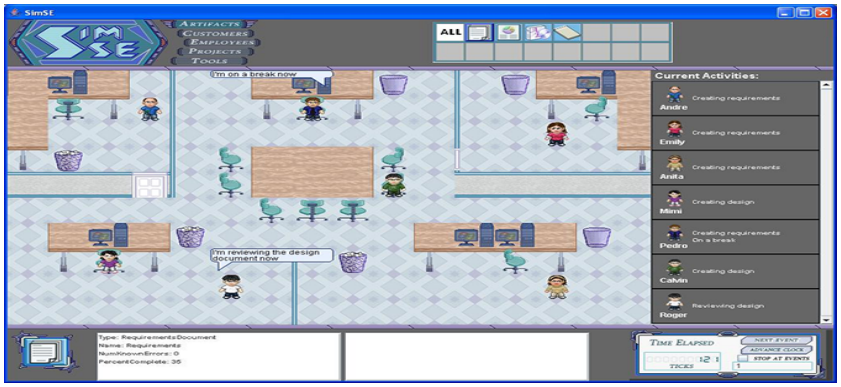


Figure 2.2: Snapshot of SimSE simulation game

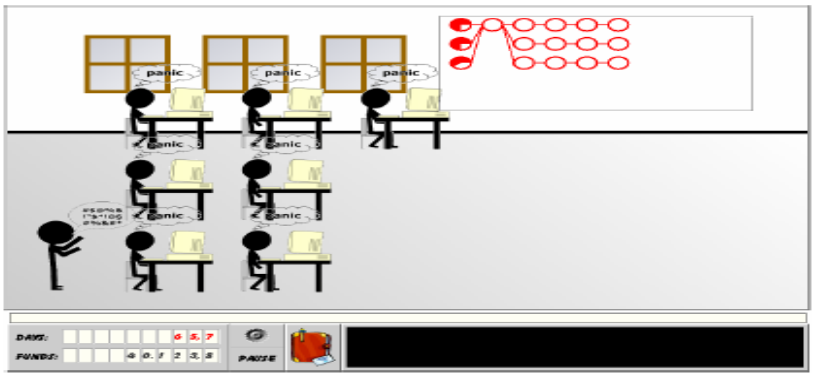


Figure 2.3: Snapshot of Incredible Manager simulation game

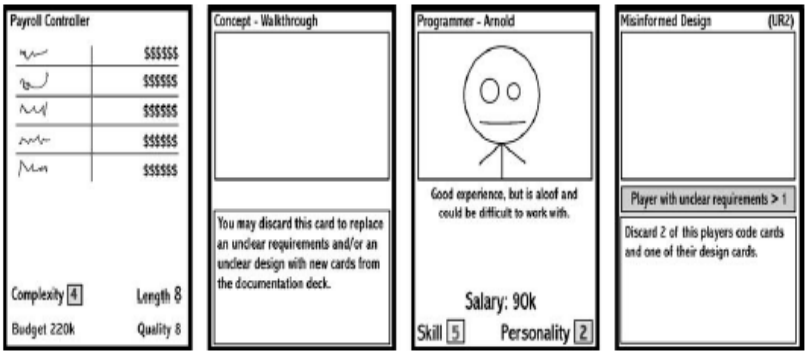


Figure 2.4: Snapshot of Problems and Programmers simulation game

Figure 2.5: Snapshot of SimVBSE simulation game

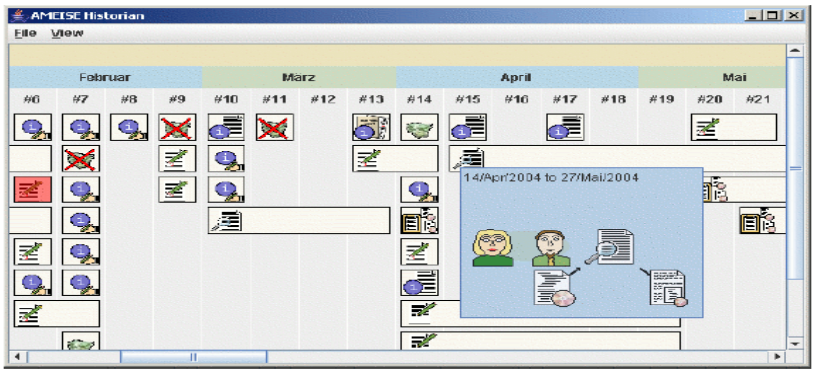


Figure 2.6: Snapshot of AMEISE simulation game

Table 2.1: Related Work (Sorted in Reverse Chronological Order) and Tools on Teaching Software Engineering Concepts using Simulation Games

SNo.	Tool Name	Year	Simulation Topic	University	Interface
1	AMEISE [16]	2013 [Bollin'13]	Managing a SE project (focussing on software quality)	Alps-Adriatic University, Carinthia Tech Inst. and Linz University	Menu based, Command Interface
2	DELIVER [15]	2012 [Wangenheim'12]	Earned Value Management	Federal University of Santa Catarina	Board Game
3	ProMaSi [14]	2011 [Petalidis'11]	Project Management	T.E.I. of Central Macedonia, Serres-Greece	Java based, Desktop environment
4	SimVBSE [13]	2006 [Jain'06]	Value-based Software Engineering	University of Southern California	Animated, 3D interface
5	Problems and Programmers [12]	2005 [Baker'05]	Software Engineering Processes	University of California, Irvine	Card Game
6	SimjavaSP [11]	2005 [Shaw'05]	Software Process Knowledge	University of Tasmania	Menu-based, Graphical Interface
7	Incredible Manager [10]	2004 [Dantas'04]	Project Management Experiential Learning	Brazilian University	Button Driven, Command Interface
8	SimSE [7]	2004 [Navarro'04]	Software Engineering Processes	University of California, Irvine	Web-based, Graphical Interface
9	SESAM [5]	2000 [Drappa'00]	Software project Management	Stuttgart University, Germany	Text Based, Pseudo-Natural Language Command Interface

3

Game Architecture and Design

3.1 Learning Objectives

In this game we define 12 learning objectives covering multiple aspects of peer code review. The reason for limiting the learning objectives to 12 is to restrict the game duration from exceeding 15-20 minutes time duration. The more the duration of game, lesser will be the interest in game play among the students. These learning objectives are captured in our pre game questions[1] and post game questions[2]. Table 3.1 shows all the 12 learning objectives, corresponding decisions and the questions which we ask to the player in game. Table 3.1 shows the structure that we follow to design the game questions. Each question is designed keeping in mind the learning objectives of the game. Based on the learning objectives we come up with a situation or a decision which best questions that objective. An evaluation question is then formed around this decision which challenges the player's knowledge to its best. We therefore present different situations to the player where they have to make decisions like whom to assign the task of review process (Figure 3.8), what inspection rate to choose (Figure 3.10), when to start with review (Figure 3.7), steps required to foster good code review culture in team (Figure 3.13) etc. A decision can map to one or more learning objectives. Similarly two or more decisions may map to one evaluation question.

[1] http://bit.ly/1ddyitO
[2] http://bit.ly/1GEdBBX

Table 3.1: Mapping between the learning objectives, decision taken or situation encountered during the game and the evaluation question during the test

SNo.	Learning Objective	Decision or Situation	Evaluation Question
1	The earlier a defect is found, the better. The longer a defect remains in an artifact, the more embedded it will become and the more it will cost to fix[17]	When to start the code review during project development	Mr. Manager, This is to inform you that 10% of project has been developed. Would you like to forward these code modules for review or send them directly for testing? As a manager, what would your decision be?
2	Authors should annotate source code before the review begins[3]	What are some good code development practices which developers should follow to carry out the code review process smoothly	Mr. Manager, I have been observing that our novice developers lack the knowledge of standard code development practices. I would suggest to get them acquainted with these practices to fasten up the code review process. Which of the following guidelines will you issue for developers?
3	Use of checklists substantially improve results for both authors and reviewers[18]	What are some good code development practices which developers should follow to carry out the code review process smoothly	Mr. Manager, I have been observing that our novice developers lack the knowledge of standard code development practices. I would suggest to get them acquainted with these practices to fasten up the code review process. Which of the following guidelines will you issue for developers?
4	Reviewer should always aim for an inspection rate of less than 300-500 LOC/hour[17]	What is the optimal inspection rate for carrying out code review	The rate at which code is reviewed should minimize the defect density and at the same time increase the productivity of reviewer involved Which amongst the following is the most optimal inspection rate?
5	Review fewer than 200-400 LOC at a time[17]	What is the preferable code size to carry out review	What is the preferable code size to be reviewed at a time, that you would suggest to reviewer for carrying out his task efficiently?
6	Verify that defects are actually fixed after review[17]	How to verify that defects uncovered by review are actually fixed	Mr. Manager, This to inform you that our developers often forget to fix bugs found during code reviews. It is hard for me to keep track of these unresolved defects, which in turn is affecting the code quality too. Please look into this issue and suggest a good way to ensure that defects are fixed before code is given All Clear sign.

Table 3.2: Mapping between the learning objectives, decision taken or situation encountered during the game and the evaluation question during the test

SNo.	Learning Objective	Decision or Situation	Evaluation Question
7	Peer code review is a staple of the software industry[2]	Should the review process be compromised in unforeseen circumstances like -project deadline is pre poned, developer quits in last few days of project release etc.	Hello Mr. Manager, I wanted to talk to you regarding the project deadline. Due to an unforeseen market change we want the deadline to be bumped up by 3 weeks. What do you say? or A developer quits 1 month before the final product release date. What would your decision be?
8	Reviews should be conducted by invested experts and co developers[18]	Who is the best choice to carry out review process based on experience	Among your 8 team members, select a person to carry out the review process.
9	The reviewers expertise must be balanced with their workloads and other factors[18]	How to handle the division of workload to be assigned to reviewer	Mr. Manager, This is to inform you that recently I have been finding myself occupied most of the time with reviewing codes which leaves me almost no time to develop modules assigned to me. I do not want to complain but it is affecting my performance both as reviewer and developer. Please consider my case. What will you do as Manager to handle this situation ?
10	Managers must foster a good code review culture in which finding defects is viewed positively[17]	How to foster a good code review culture in your team	It has come to your notice that reviewer has been threatening the novice developers regarding providing poor feedback in performance evaluation. What steps would you take to foster a good code review culture in your team ?
11	Lightweight-style code reviews are efficient, practical, and effective at finding bugs[17]	Which type of review process (light weight vs heavy weight) to use	Select the type of review process you would recommend to use in this project
12	Reviewers may not be willing to do their best to review the written code by their peer developers, especially when they are assigned poor program writers[19]	How to foster a good code review culture in your team	It has come to your notice that reviewer has been threatening the novice developers regarding providing poor feedback in performance evaluation. What steps would you take to foster a good code review culture in your team?

3.2 Unexpected Events

We introduce unexpected events and unforeseen circumstances (such as internal conflicts between the team members, attrition and change in deadline or demand from the customer) in the game to make it more realistic. Unexpected events are un observable and our goal is to examine the response and the decision making ability of the player to unexpected circumstances. Figure 3.12, 3.16 and3.18 shows screen shot for the simulation game in which the player is presented with an unexpected situation. As shown in Figure 3.18, the project manager is encountered with a situation wherein a developer quits the team or organization just one month before the release date.

3.3 Scoring System

3.3.1 Technical Debt

We apply Technical Debt[1] concept or metaphor as one of the elements of our scoring system. We calibrate our scoring system such that incorrect decisions (quick and dirty solutions) lead to accumulation of technical debt. Figure 3.1 shows technical debt score of various players (data collected during our experimental evaluation of the simulation game) as the game progresses from start to finish. Figure 3.1 reveals various behaviors: we observe cases in which technical debt gets build-up due to lack of knowledge and incorrect decisions and on the other hand we notice cases in which prudent decisions lead to controlled technical debt. Each decision a player makes has certain weight or point associated to it varying from 1 to 5. We take the maximum range value i.e. 5 as standard TD (Technical Debt) point against which all calculations are made. For every decision taken by the player, we calculate the deviation of player's current decision point from the standard TD point and find the average TD point value obtained so far. We then calculate the equivalent percentage value of this TD point relative to the standard TD point which is the overall TD incurred by player so far in the game (refer to Equations 3.1, 3.2, 3.3 and 3.4).

[1] http://martinfowler.com/bliki/TechnicalDebt.html

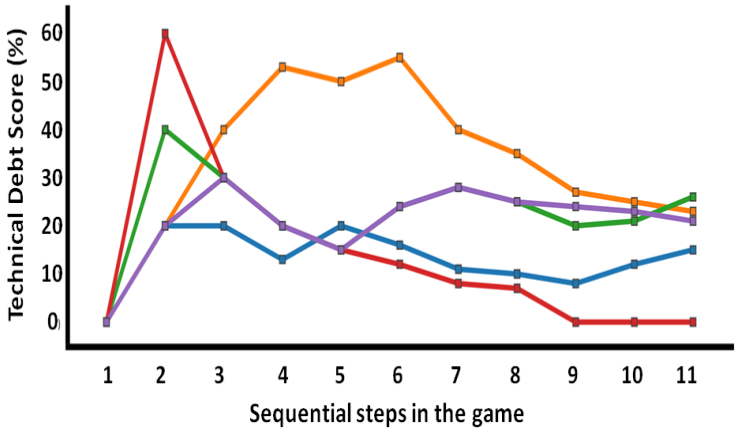


Figure 3.1: Technical debt values (and increasing or decreasing trends) for various players during the execution of the game

$$deviationValue = (standardTD - decisionWeight) \tag{3.1}$$

$$sumTD = sumTD + deviationValue \tag{3.2}$$

$$TD_{avg} = sumTD/decisionCounter \tag{3.3}$$

$$TD = (TD_{avg}/standardTD) * 100 \tag{3.4}$$

3.3.2 Time

Time is the most valuable resource in any project and time management is another key aspect of managing a project. It is therefore required by the player to properly plan the project time and associated decisions to meet project deadline. Every decision that a player takes in game has a positive or negative impact. The magnitude of impact varies depending on the choice or decision made by the player (as shown in Figure 3.2). Series of decisions taken by a player either prevent them from meeting the deadline or they are able to complete the project successfully. All of the decisions have a path time

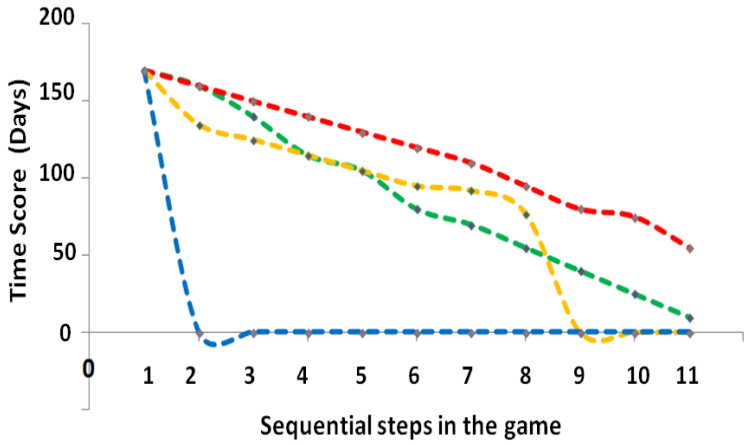


Figure 3.2: Time values (and increasing or decreasing trends) for various players during the execution of the game

associated with them. This path time gets deducted from the remaining time as the player proceeds further in game (Equation 3.5 and 3.6).

$$remainingTime = remainingTime - pathTime \tag{3.5}$$

$$timeScore\ (days) = remainingTime \tag{3.6}$$

3.3.3 Budget

At the beginning of game, player is allotted a budget of Rs 2 million to complete the project. The scoring system that we have built, tracks player's utilisation of this budget. If at any point of time player has consumed the entire budget value, they can go no further in game. Figure 3.3 shows different player behaviours in terms of budget utilisation. Budget remaining at the end of game is the combined effect of the cost consumed to perform review, developer's recruitment, buying tools, team incentives etc. Refer to Equation 3.7 and 3.8 to see how path cost for player's decision is used to obtain the cost score.

$$remainingCost = remainingCost - pathCost \tag{3.7}$$

$$costScore\ (Rs) = remainingCost \tag{3.8}$$

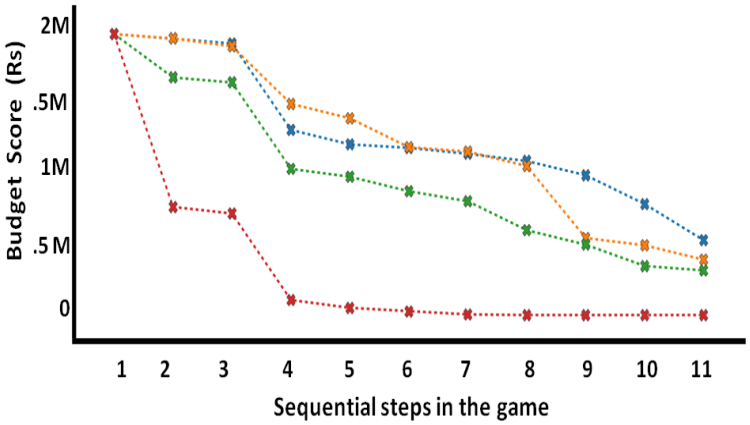


Figure 3.3: Budget values (and increasing or decreasing trends) for various players during the execution of the game

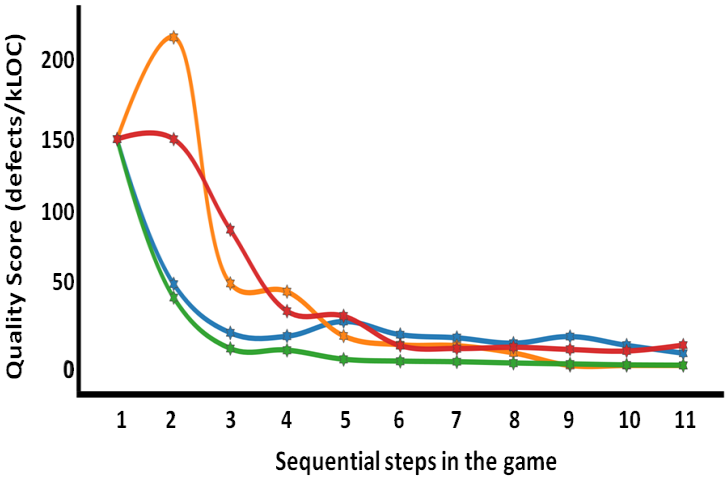


Figure 3.4: Project quality values (and increasing or decreasing trends) for various players during the execution of the game

3.3.4 Quality

We incorporate the concept of software quality in our scoring system to keep a check of code review consistency maintained by the player. The scoring system captures the quality standards from the beginning of the project. We use defect density (defects/kLOC) to measure the software quality maintained during the game. Code bases

with a defect density of 1.0 (or 1 defect for every 1,000 lines of code) are considered good quality software[1]. Figure 3.4 illustrates the varying quality standards for different players as they progress through the game with an initial defect density of 145 defects/kLOC (mentioned at the beginning of game). Each path that player takes has a defect percentage (defect%) associated with it. Depending on player's decision, this defect% either increases or decreases the software quality standards. Equation 3.9, 3.10 and 3.11 captures project quality (projectQlty) calculation.

$$decisionQlty = defect\% * projectQlty \tag{3.9}$$

$$projectQlty = projectQlty \ \pm \ decisionQlty \tag{3.10}$$

$$qualityScore\ (defects/kLOC) = projectQlty \tag{3.11}$$

3.4 Game Tree

Each game consists of a problem space, initial state and single (or a set of) goal states. A problem space is a mathematical abstraction in form of a tree (refer to Figure 3.5) where the root represents starting game state, nodes represent states of the game (decisions in our case), edges represent moves and leaves represent final states (marked as red circles) [20]. Each node or state of tree has four attributes associated with it i.e. cost, time, quality and technical debt incurred. These four values vary with each path i.e. depends on the player's decision of following a certain path. A game tree also demonstrates the different paths a player can follow and the weights associated with each path based on it's correctness. Figure 3.5 represents the game tree for an instance of our game where a player selects best possible path at each step of game.

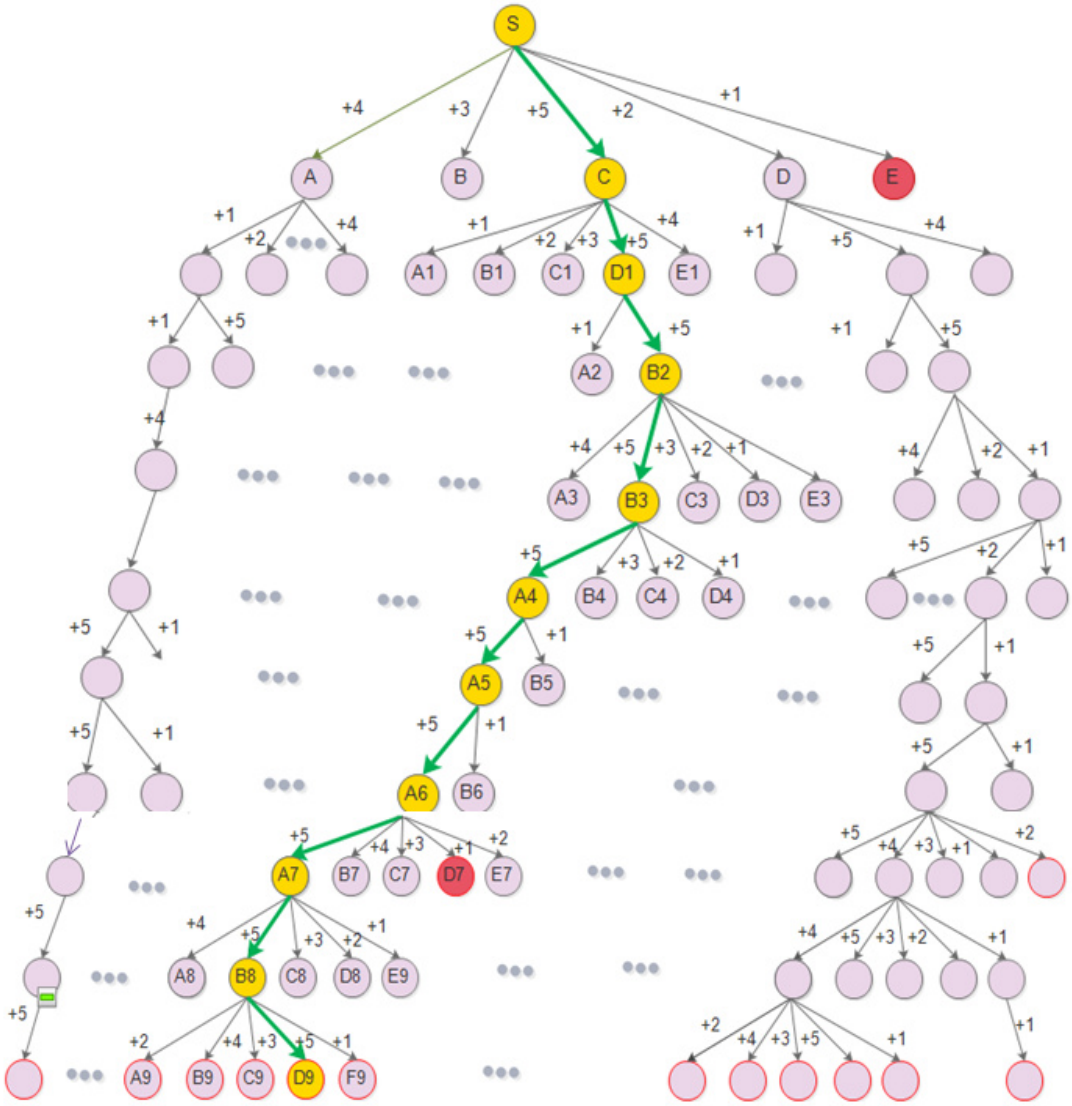


Figure 3.5: Game Tree demonstrating the best path taken by any player

3.5 Final Scoring

At the end of game, each player gets a score reflecting their overall performance. This score produced, takes into account all the factors like path taken, remaining time, budget consumed, project quality index and technical debt accumulated at the end of game. Following are the steps depicting the procedure to compute the performance of each player during the execution of game and based on the decisions made by the player.

1. Each decision a player takes has a weight (W_d) associated with it, which varies from +1 to +5 (represented in Figure 3.5). As the player proceeds in game, the weight associated with each decision keeps on accumulating and is stored in P_{sum} (refer to Equation 3.12). Final value of P_{sum} is then used to determine whether a player followed a poor, optimal, good or an excellent path during the game.

$$P_{sum} = \sum W_d \ (out\ of\ 50) \tag{3.12}$$

2. Next we scale and obtain the equivalent values for cost (C_f, Table 3.3), time (T_f, Table 3.4) and quality (Q_f, Table 3.5) remaining at the end of game. Project attributes sum (P_{attr_sum}) holds the average of equivalent values of these three project attributes (Equation 3.13) [21].

$$P_{attr\_sum} = (C_f + T_f + Q_f)/3 \ (out\ of\ 50) \tag{3.13}$$

3. Score sum (S_f') is obtained by adding the values calculated in step 1 and 2 (Equation 3.14).

$$S'_f(out\ of\ 100) = P_{sum} + P_{attr\_sum} \tag{3.14}$$

4. We then make use of TD_{avg} (Equation 3.3) to calculate the final score (S_f) in equation 3.15. The value of S_f is used to evaluate player's performance (refer to Table 3.6).

$$S_f = S'_f * (1 - TD_{avg}/standardTD) \tag{3.15}$$

Table 3.3: Table representing the equivalent value of cost i.e. C_f for the budget remaining at the end of game

Remaining Cost	**Equivalent C_f**
More than Rs 2,000,000 used	10
Exact Rs 2,000,000 were used i.e. Rs 0 left	40
20% or less of Rs 2,000,000 left	45
20% or more of Rs 2,000,000 left[6]	50

Table 3.4: Table representing the equivalent value of time i.e. T_f for the time remaining at the end of game

Remaining Time	**Equivalent T_f**
More than 180 days	10
Exact 180 days used	40
5% or less of 180 days i.e 1-10 days left	45
5% or more of 180 days i.e. 10 days +	50

Table 3.5: Table representing the equivalent value of quality i.e. Q_f for the defects density remaining at the end of game [1]

Remaining Defect Density	**Equivalent Q_f**
More than 1 defects/kLOC	10
0.69 - 1 defect/kLOC	40
0.2 - 0.69 defects/kLOC	45
0 - 0.2 defects/kLOC	50

Table 3.6: Table representing the player's performance remarks based on final score obtained i.e S_f

Players Performance	**Player's Score (S_f)**
Poor	1 - 40
Average	40 - 70
Good	70 - 95
Excellent	95 - 100

3.6 Feedback and Analysis

To meet the game's objective, it is necessary to ensure that player is learning throughout the course of game. The player is given a proper feedback at each step of the game so they can observe the effect of decisions taken by them on the project. Every decision of player has a direct impact on the four parameters of scoring system i.e. time, cost, quality and technical debt, which is made visible by the presence of four dynamic meters in game screens (demonstrated from Figure 3.22 to 3.31). The value of score meters change after each decision and makes player aware of the consequences of his decision on the available resources. In the end, final values in score meters obtained for the player is compared with the ideal values that should be there (refer to Figure 3.21). It helps student compare the consumption of resources done during their project course and the quality standards that they managed to maintain. Along with step wise feedback, we also provide a detailed analysis of player's performance at the end of the game. We reflect to player's the decisions taken by them during the game and provide feedback in form of remarks. The remark that a player gets, help them discover about the correctness of their decisions. For example, if a player chooses a developer with 4-6 years of experience(Figure 3.23), they are reminded that there is a more experienced and expert co developer present in the team who can carry out this task. In case they select a very fast inspection rate like 900 or above LOC/hour (Figure 3.25) they are told that with this high inspection rate they can conclude that the reviewer is not looking at the code at all. This in depth analysis helps player trace the events and reasoning (learning through success and failure as well as discovery) behind the points awarded to them.

3.7 Game Architecture

The game structure consists of different components. These components are independent of each other and have different purposes to serve in game. It is the interaction of these components that leads to successful functioning of game. Figure 3.6 illustrates the different components of game structure and their interaction.

1. **Rule and Scenario Model:** This is the component where we define the rule and learning objectives of the game. Different scenarios based on the game rules

are defined here.

2. **Simulation Machine:** Rule and scenario model forms the basis of simulation machine. It is the part where all these are integrated together to develop the simulation engine which is the core part of simulation.

3. **Game Machine:** It consists of an interface for the simulation model to interact with the player and get them acquainted with the real world scenarios that they can encounter in organizations.

4. **Analysis Machine:** It interacts with the simulations machine once the player is done with game. This analyses the decisions taken by player and provides them with detailed analysis of their decisions and performance in game.

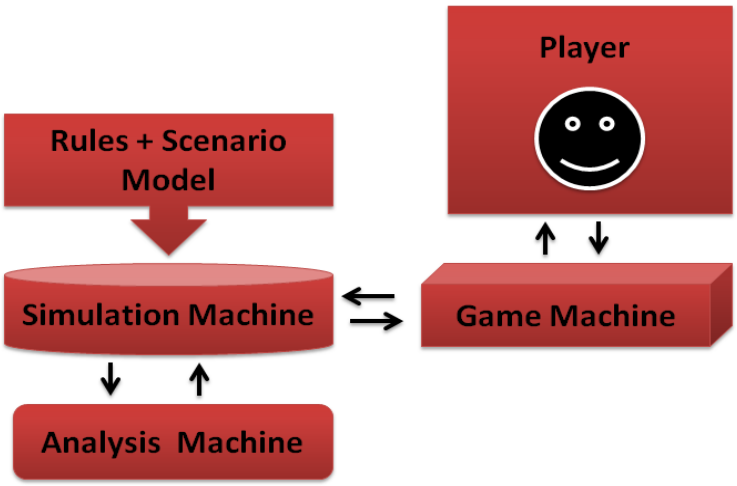


Figure 3.6: Diagram representing game components and their interaction

3.8 Snapshots of Game Play

This section contains the snapshots of screens that appear before a player. These screens represents the decisions (specified in Table 3.1, 3.2) and associated options that player is presented with. Each screen exposes player to different practices of peer code review and reveals several possible moves that a player can make. Based on these decisions and the meter values, next screen is presented to the players. The decisions involves

deciding on review time (Figure 3.7), choosing the reviewer (Figure 3.8), selecting review inspection rate (Figure 3.10), steps to foster good code review culture (Figure 3.13), handling deadline changes (Figure 3.16), dealing with team attrition (Figure 3.18) etc. Snapshots attached here illustrates the screens showing a game instance for a player. Figure 3.7 to Figure 3.18 shows different decisions that a player takes in the course of game. Figure 3.20 presents the performance score of the player calculated using the method mentioned in Final Scoring section. Figure 3.21 compares the player's performance with ideal performance values for scoring system meters. Then player is exposed to an in depth analysis of the decision they took. Figure 3.22 to Figure 3.31 demonstrates the analysis presenting feedback to player on his decision, in form of remarks. These remarks reveals the correctness of their decisions to player's, but do not provide the answer directly (supporting learning by discovery and failure).

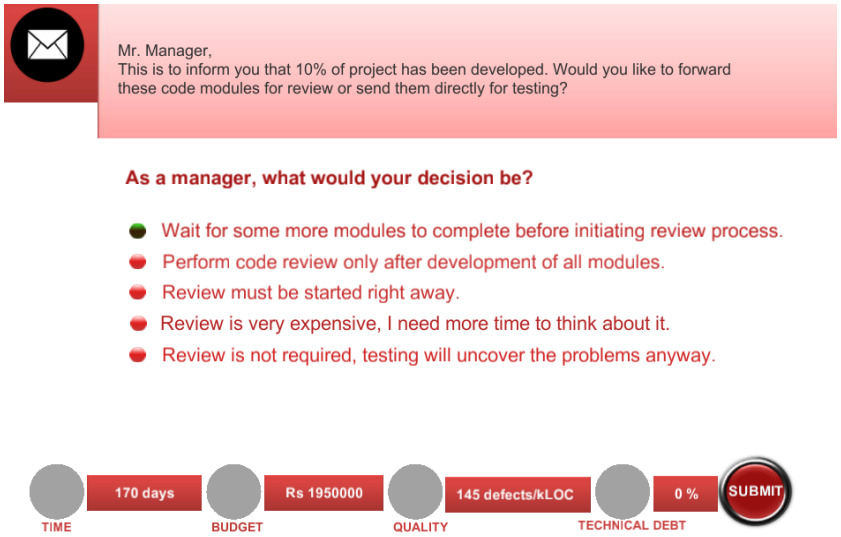


Figure 3.7: Snapshot of screen where player is asked to decide on when to start the review process

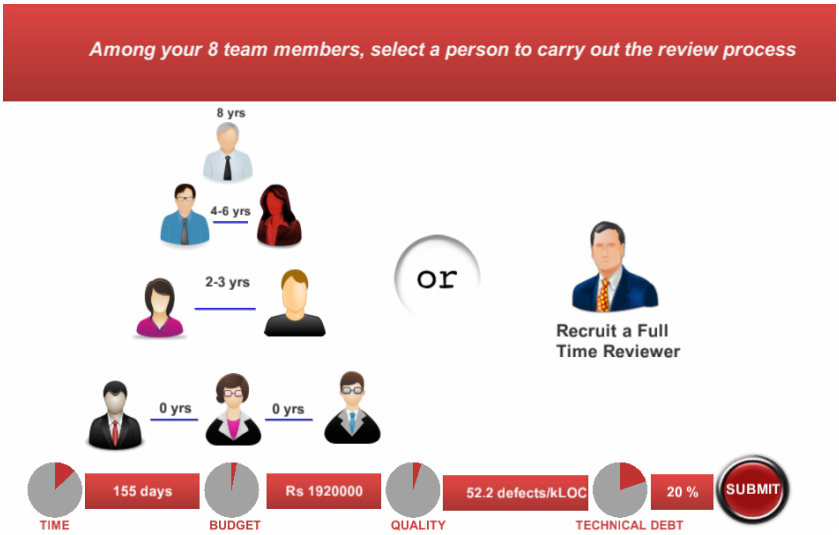


Figure 3.8: Snapshot of screen where player is to select the person to perform the task of peer code review

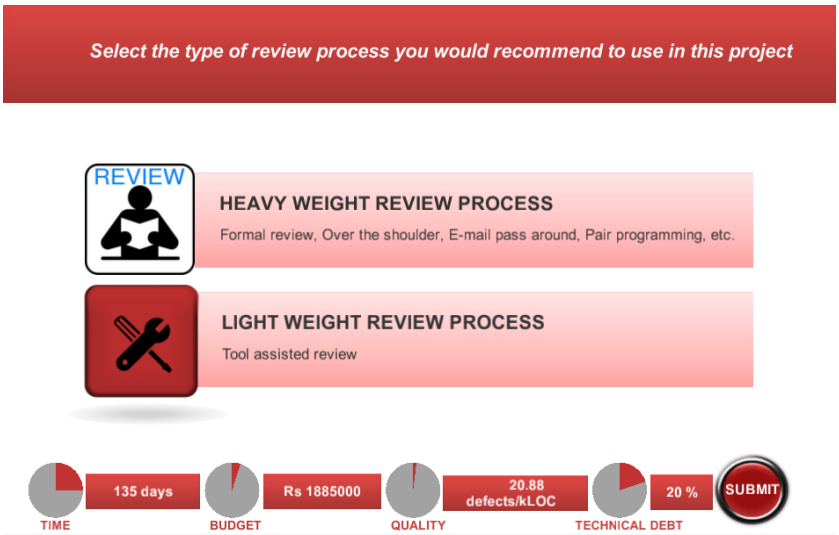


Figure 3.9: Snapshot of screen where user recommends the review process to be used

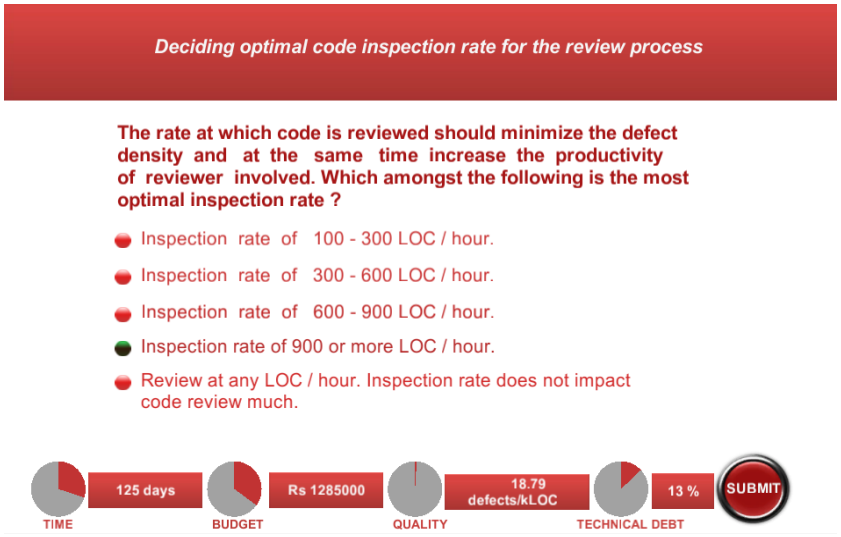


Figure 3.10: Snapshot of screen requiring player to decide on the inspection rate

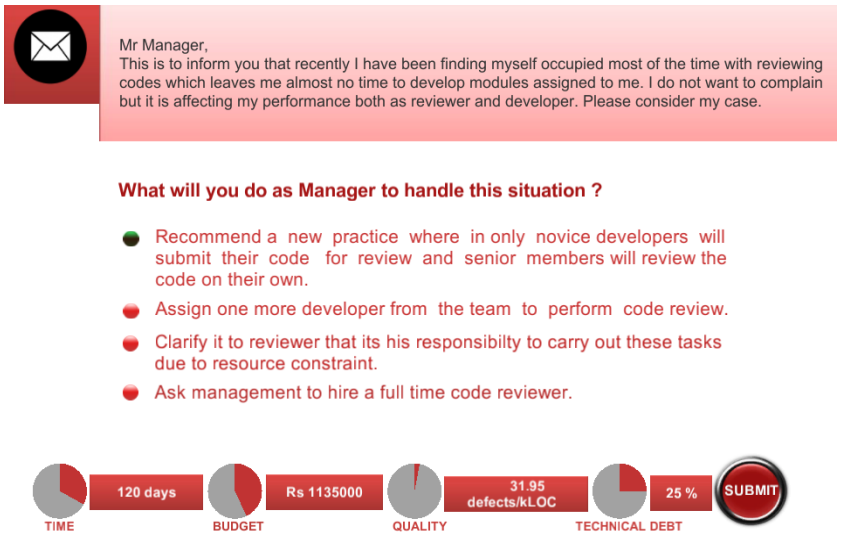


Figure 3.11: Snapshot of screen where the manager gets to handle the division of workload to be assigned to reviwer

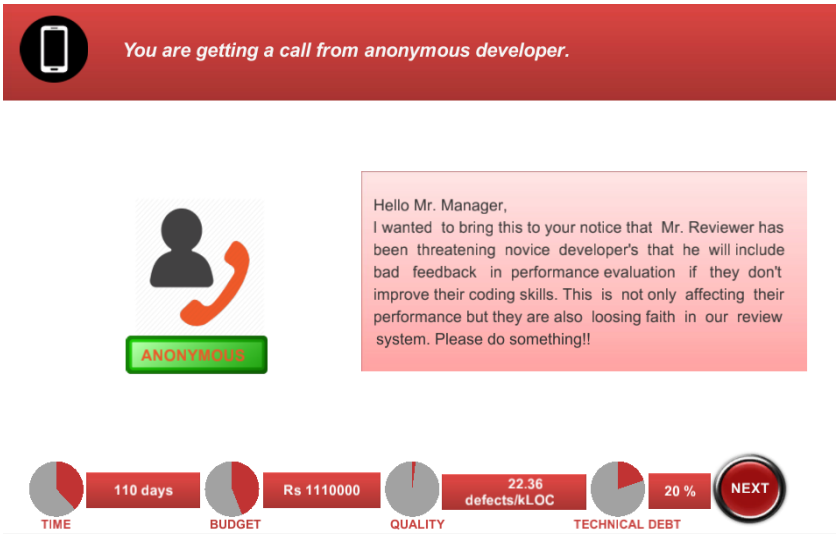


Figure 3.12: Snapshot of screen where manager gets a call from a developer complaining about reviewer's behavior

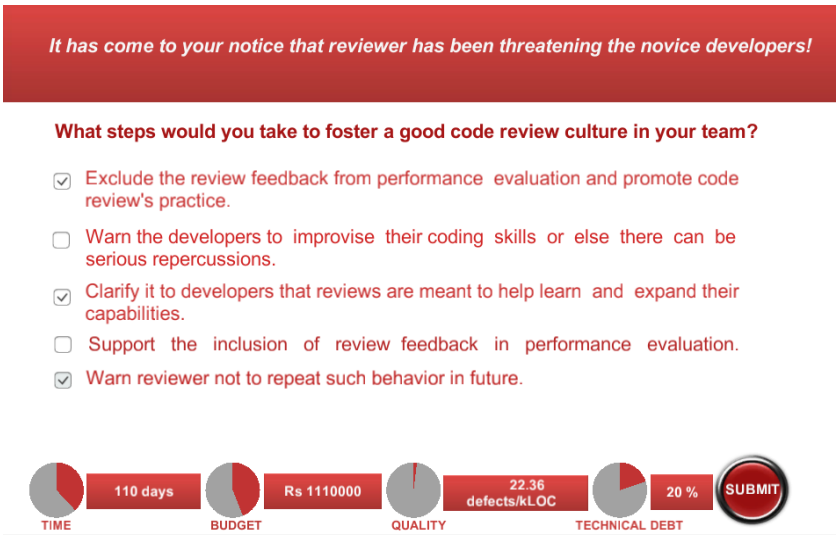


Figure 3.13: Snapshot of screen where player has to take steps to foster a good code review culture in team

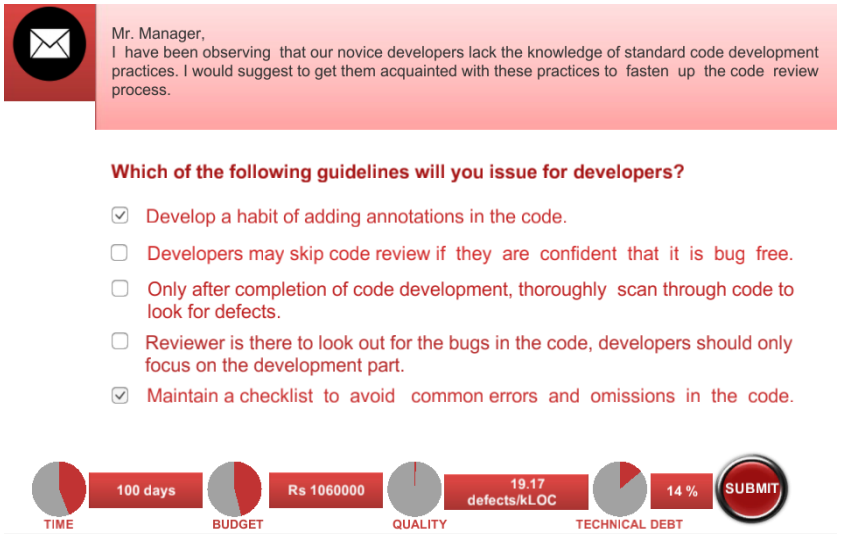


Figure 3.14: Snapshot of screen where manager gives instructions on good industrial code development practices to novice developers

Figure 3.15: Snapshot of screen where project sponsor calls manager regarding deadline preponement

Figure 3.16: Snapshot of screen where manager has to decide on how to deal with unexpected deadline change

Figure 3.17: Snapshot where manager has to take steps to ensure that defects uncovered in review are fixed

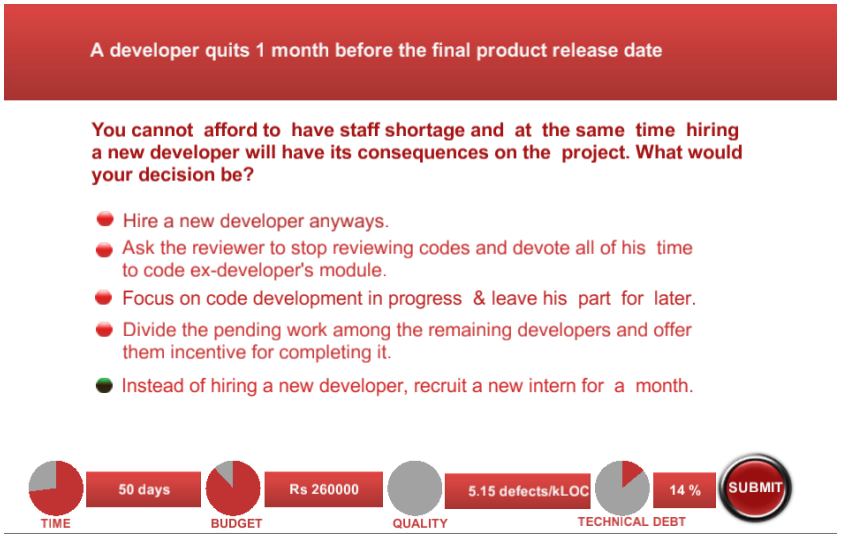


Figure 3.18: Snapshot of screen where a developer quits team, one month before the project deadline

Figure 3.19: Snapshot of screen marking the end of game

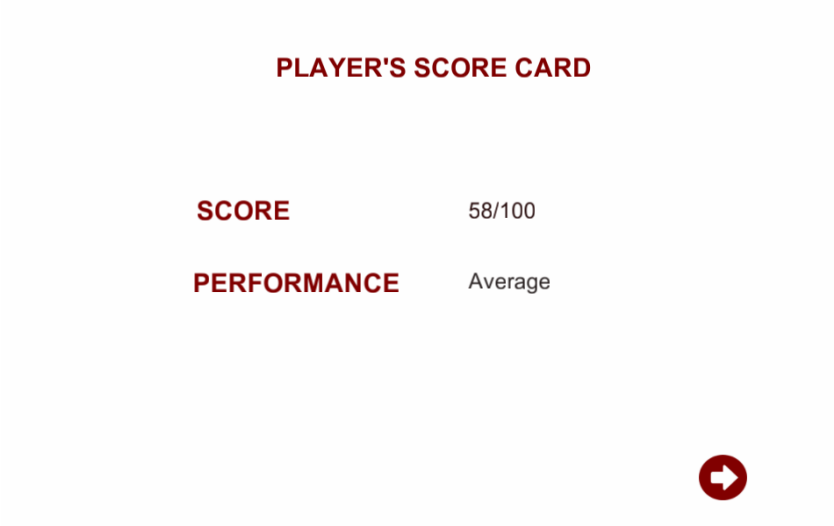


Figure 3.20: Snapshot of screen representing the score card of player's performance

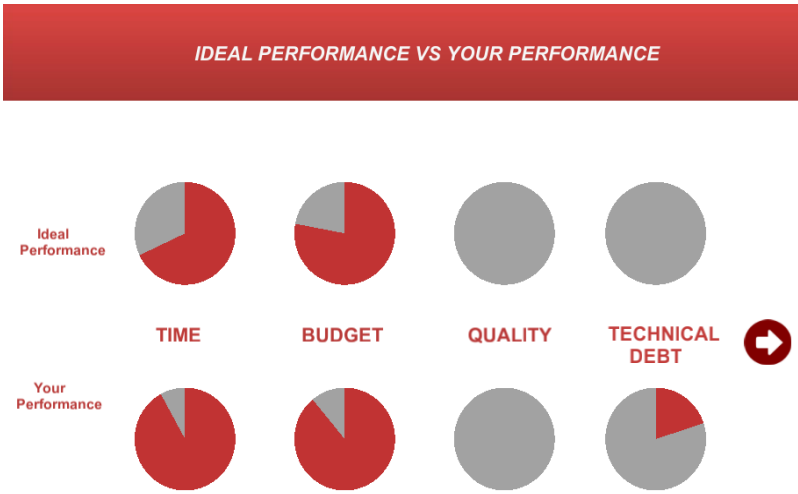


Figure 3.21: Snapshot of screen comparing the different scoring meters of player w.r.t ideal meter values

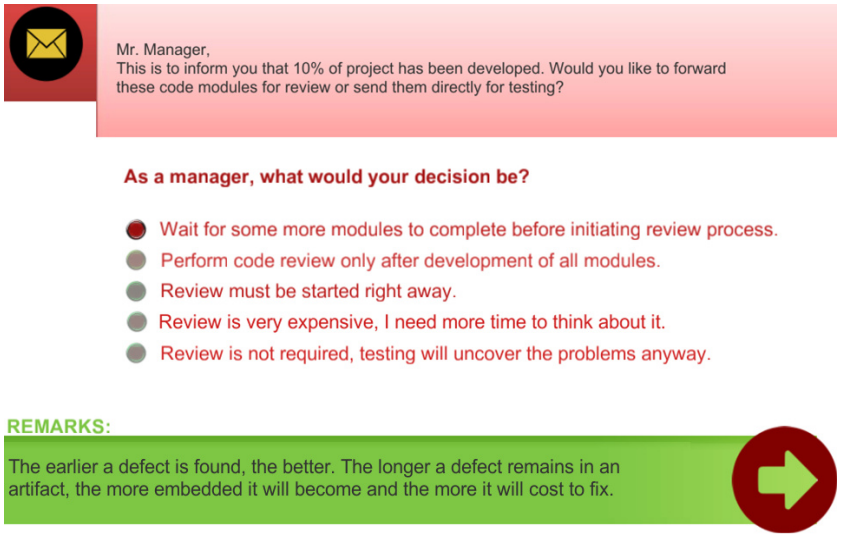


Figure 3.22: Snapshot of screen analyzing the player's decision on review start time

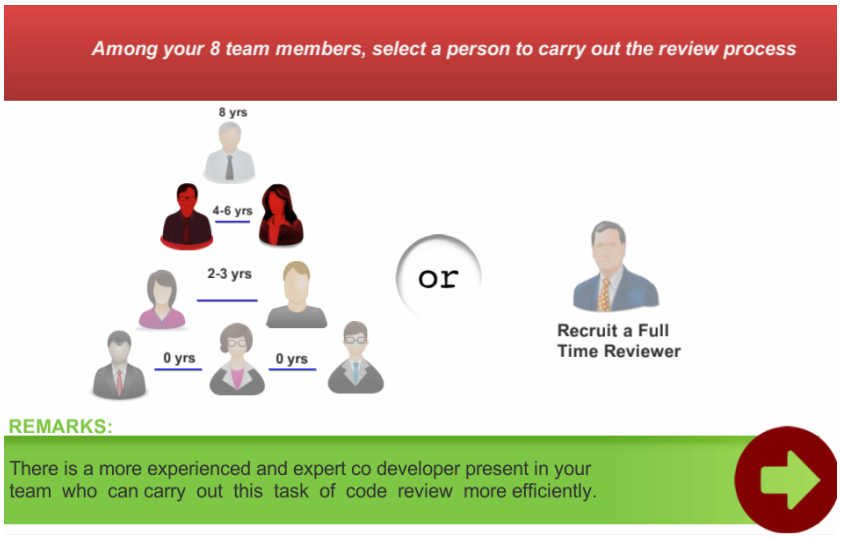


Figure 3.23: Snapshot of screen analyzing player's decision on selecting reviewer

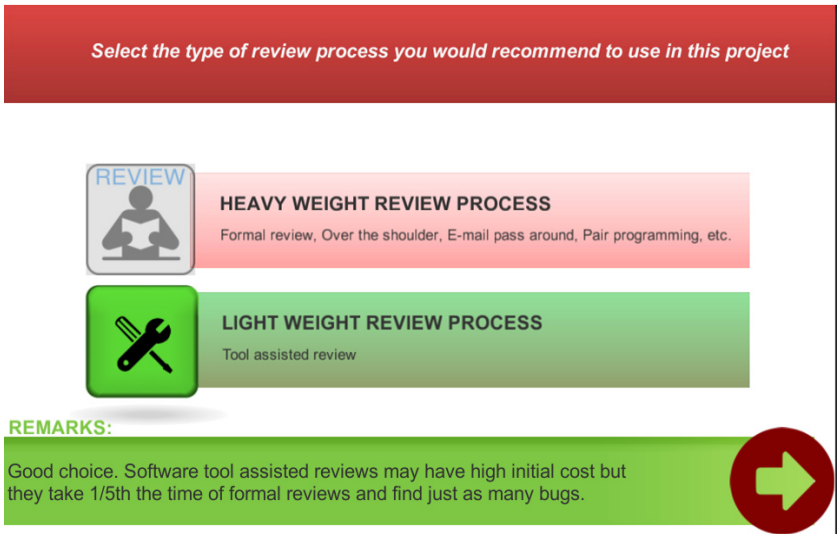


Figure 3.24: Snapshot of screen analyzing the decision of player of opting for light weight review process

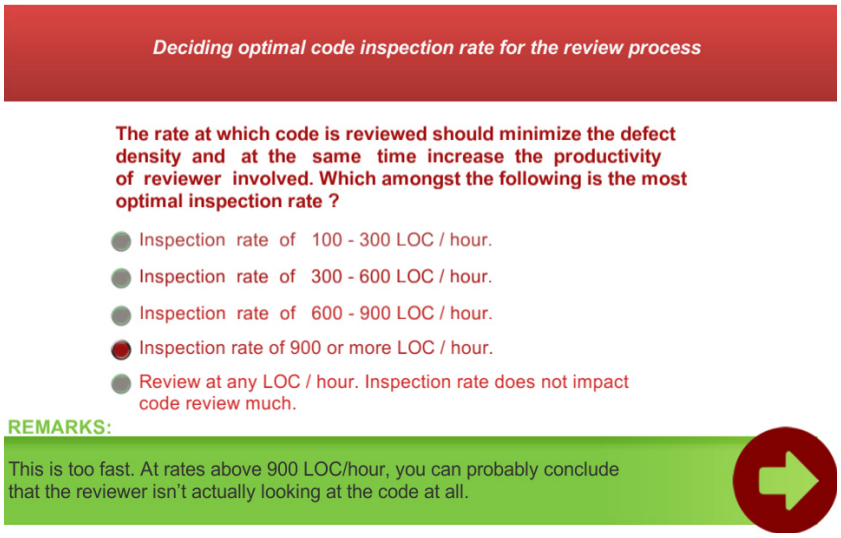


Figure 3.25: Snapshot of screen analyzing optimal inspection rate opted by player

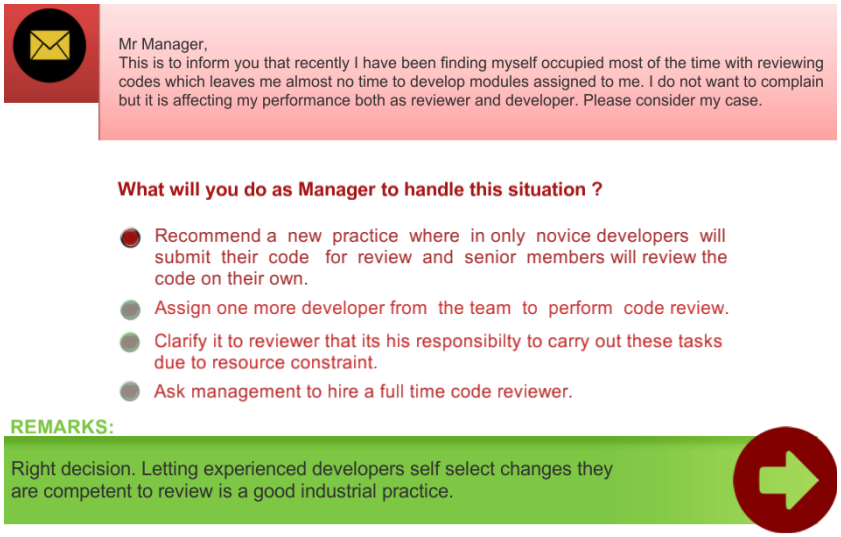


Figure 3.26: Snapshot of screen analyzing player's decision of handling excess workload of reviewer

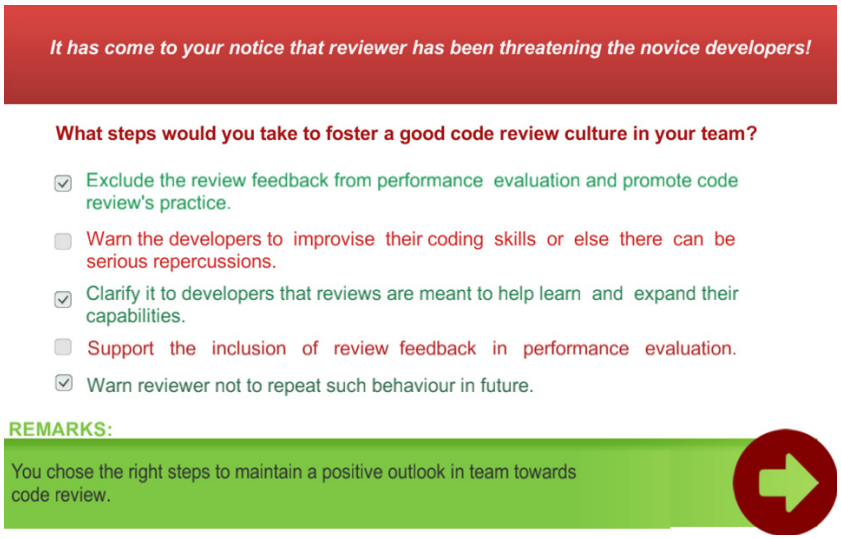


Figure 3.27: Snapshot of screen analyzing steps player took to foster good code review culture in team

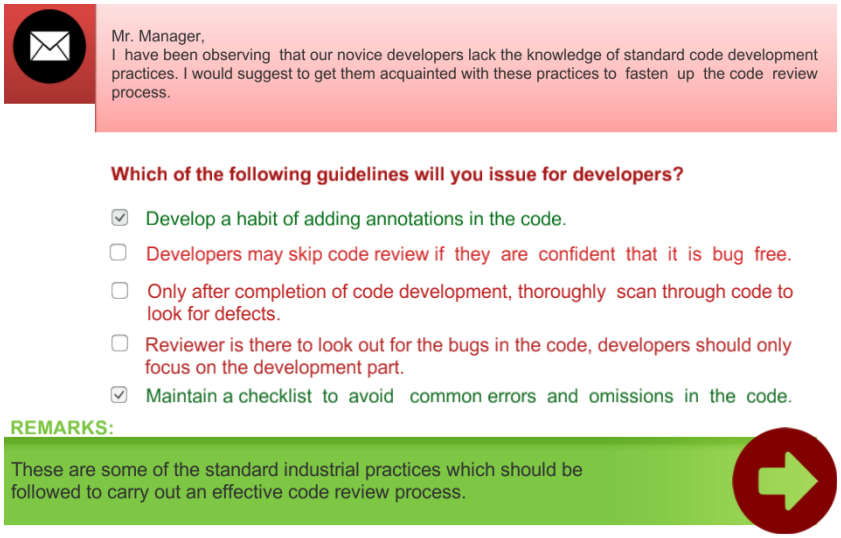


Figure 3.28: Snapshot of screen analyzing guidelines issued by player to novice developers

Figure 3.29: Snapshot of screen analyzing player's take on deadline preponement

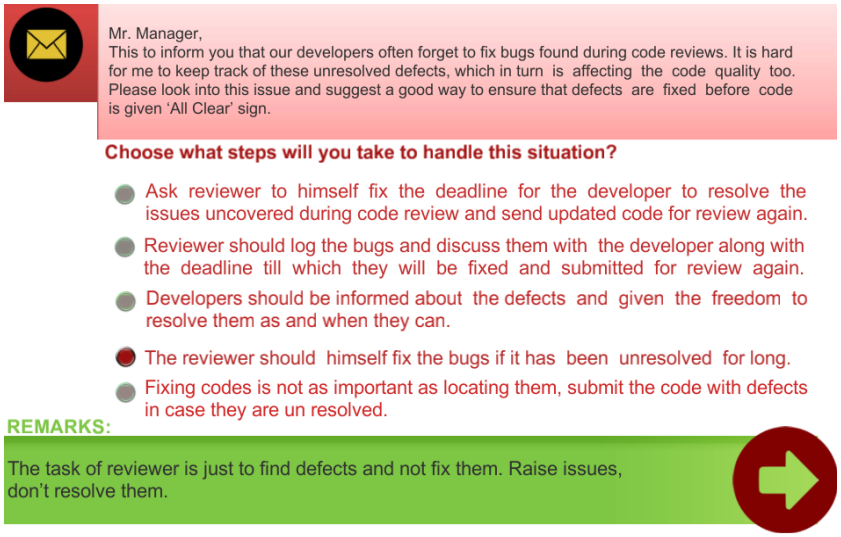


Figure 3.30: Snapshot of screen analyzing player's decision on steps to verify that defects found in review are fixed

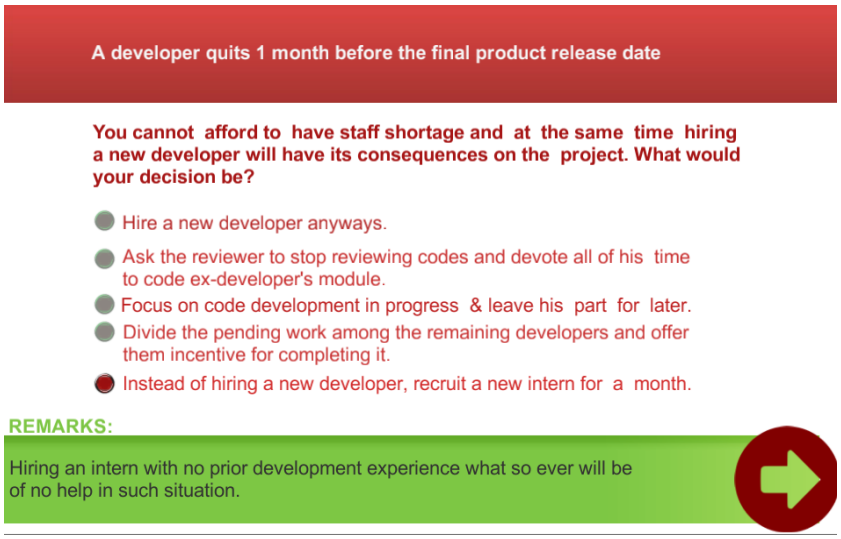


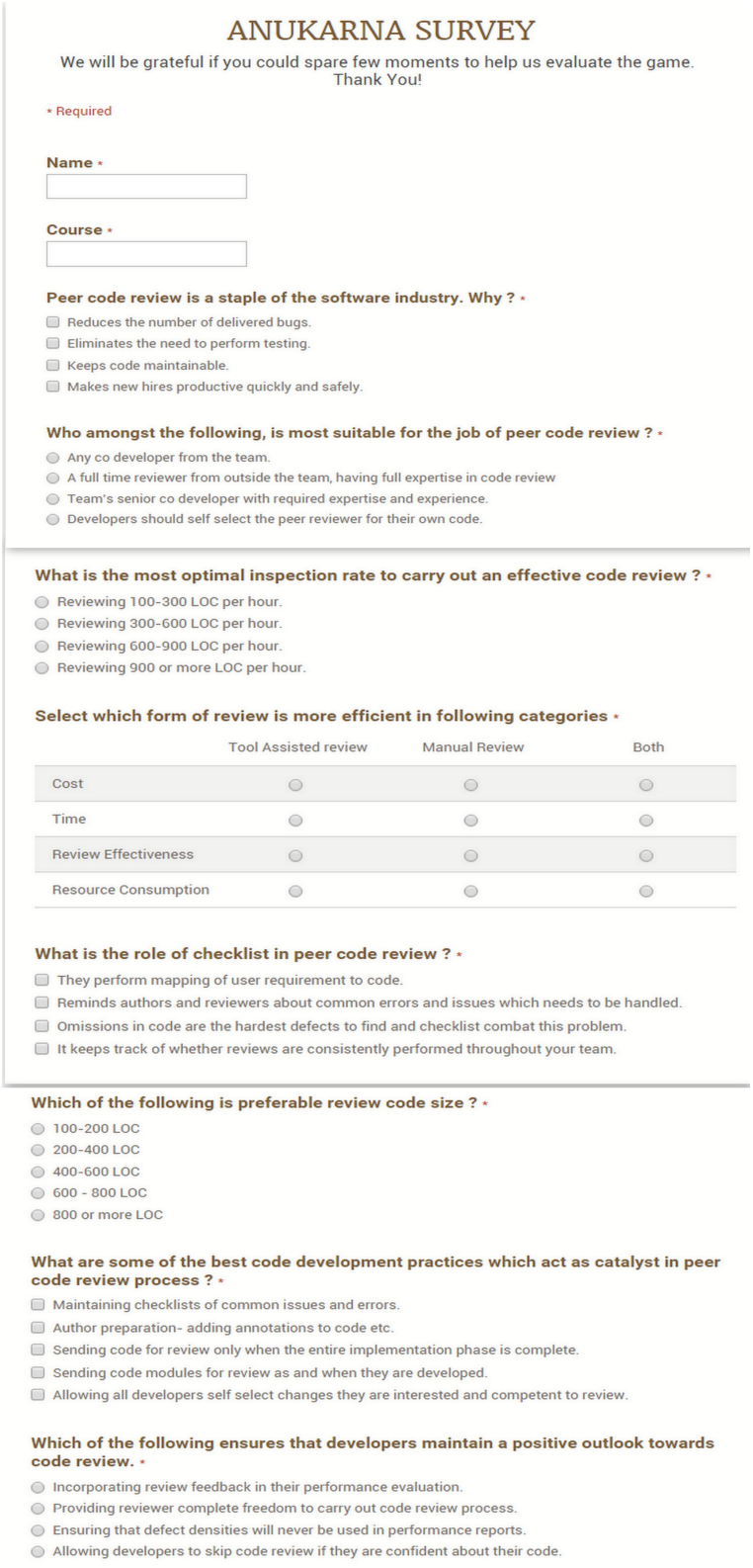
Figure 3.31: Snapshot of screen analyzing how player handles the resignation of a developer from the team 1 month before deadline

4

Experimental Analysis and Results

4.1 Experimental Dataset

We conduct experiments to evaluate our proposed approach by asking 17 students (10 Undergraduate or Bachelors in Computer Science and 7 Graduate or Masters in Computer Science students) to play the game. The students are asked to fill a questionnaire testing their knowledge of peer code review related concepts before and after playing the game. The data obtained can thus be broadly categorised into two datasets i.e. pre game and post game survey dataset (refer to Figure 4.1 and 4.2).



ANUKARNA SURVEY

We will be grateful if you could spare few moments to help us evaluate the game.
Thank You!

* Required

Name *

Course *

Peer code review is a staple of the software industry. Why ? *

- [ ] Reduces the number of delivered bugs.
- [ ] Eliminates the need to perform testing.
- [ ] Keeps code maintainable.
- [ ] Makes new hires productive quickly and safely.

Who amongst the following, is most suitable for the job of peer code review ? *

- ( ) Any co developer from the team.
- ( ) A full time reviewer from outside the team, having full expertise in code review
- ( ) Team's senior co developer with required expertise and experience.
- ( ) Developers should self select the peer reviewer for their own code.

What is the most optimal inspection rate to carry out an effective code review ? *

- ( ) Reviewing 100-300 LOC per hour.
- ( ) Reviewing 300-600 LOC per hour.
- ( ) Reviewing 600-900 LOC per hour.
- ( ) Reviewing 900 or more LOC per hour.

Select which form of review is more efficient in following categories *

	Tool Assisted review	Manual Review	Both
Cost	( )	( )	( )
Time	( )	( )	( )
Review Effectiveness	( )	( )	( )
Resource Consumption	( )	( )	( )

What is the role of checklist in peer code review ? *

- [ ] They perform mapping of user requirement to code.
- [ ] Reminds authors and reviewers about common errors and issues which needs to be handled.
- [ ] Omissions in code are the hardest defects to find and checklist combat this problem.
- [ ] It keeps track of whether reviews are consistently performed throughout your team.

Which of the following is preferable review code size ? *

- ( ) 100-200 LOC
- ( ) 200-400 LOC
- ( ) 400-600 LOC
- ( ) 600 - 800 LOC
- ( ) 800 or more LOC

What are some of the best code development practices which act as catalyst in peer code review process ? *

- [ ] Maintaining checklists of common issues and errors.
- [ ] Author preparation- adding annotations to code etc.
- [ ] Sending code for review only when the entire implementation phase is complete.
- [ ] Sending code modules for review as and when they are developed.
- [ ] Allowing all developers self select changes they are interested and competent to review.

Which of the following ensures that developers maintain a positive outlook towards code review. *

- ( ) Incorporating review feedback in their performance evaluation.
- ( ) Providing reviewer complete freedom to carry out code review process.
- ( ) Ensuring that defect densities will never be used in performance reports.
- ( ) Allowing developers to skip code review if they are confident about their code.

Figure 4.1: Represents the section common in both pre game and post game survey

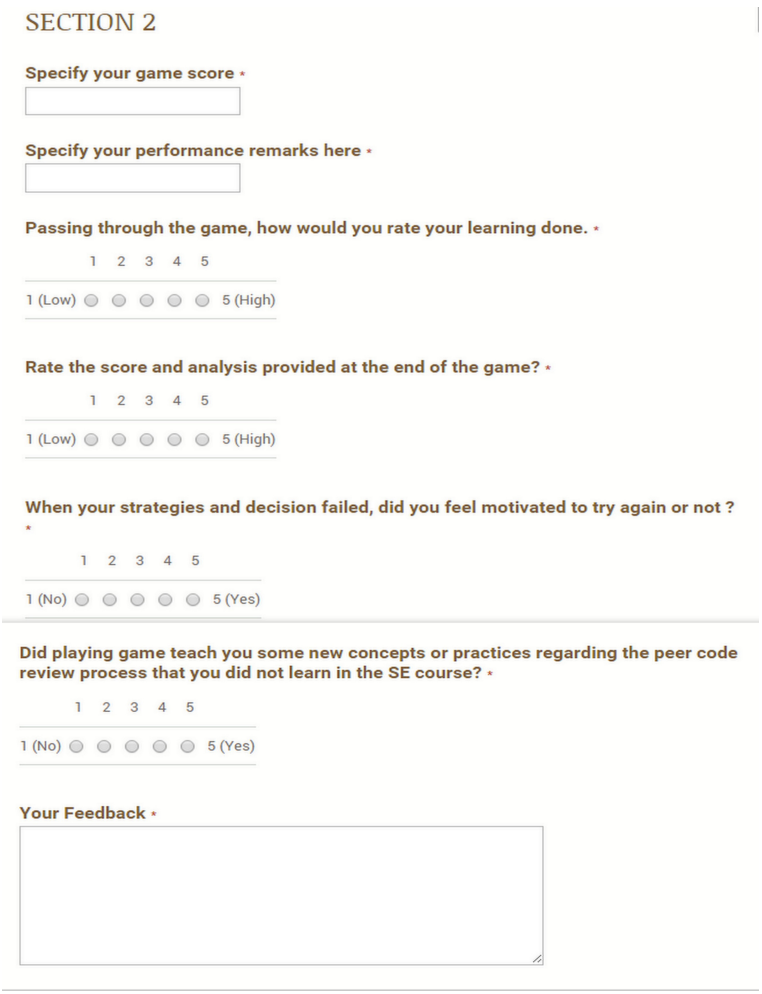


Figure 4.2: Represents the Section 2 of post game survey, where players are asked to rate the game on different aspects

4.1.1 Pre Game Survey Dataset

Pre game survey dataset[1], holds the responses to the questionnaire we ask students before playing the game. Table 4.1, 4.2 presents the list of questions and their responses in the pre-game questionnaire. The pre-game questionnaire consists of questions having one correct answer (2, 3, 5, 7, 8, 9, 10 and 11) and also questions having more than one correct answer (1, 4 and 6). Table 4.1 displays the questions, answer choices, correct answer(s) and the responses of 17 students. Table 4.1 reveals that 30% of the respondents selected the correct choice for the most optimal inspection rate for peer

[1]http://bit.ly/1ddyitO

code review and only 18% selected the correct answer for optimal code review size. It is observed that 47.05% respondents preferred to recruit a full time reviewer from

Table 4.1: Questionnaire [Q1-Q7] Results before and after the Game Play

[1] Peer code review is a staple of the software industry. Why?	Pregame	Postgame
Reduces the number of delivered bugs [✓]	70.58%	76.47%
Eliminates the need to perform testing	17.64%	05.88%
Keeps code maintainable [✓]	47.05%	64.70%
Makes new hires productive quickly and safely[✓]	47.05%	76.47%
[2] Who amongst the following is most suitable for the job of peer code review?		
Any co-developer from the team	23.53%	00.00%
A full time reviewer from outside the team, having full expertise in code review	47.05%	00.00%
Team's senior co developer with required expertise and experience [✓]	29.42%	100.00%
Developers should self select the peer reviewer for their own code	00.00%	00.00%
[3] What is the most optimal inspection rate to carry out an effective code review ?		
Reviewing 100-300 LOC per hour	17.65%	00.00%
Reviewing 300-600 LOC per hour [✓]	29.41%	64.71%
Reviewing 600-900 LOC per hour	41.18%	35.29%
Reviewing 900 or more LOC per hour.	11.76%	00.00%
[4] What is the role of checklist in peer code review?		
It keeps track of whether reviews are consistently performed throughout your team	23.52%	41.17%
Omissions in code are the hardest defects to find and checklist combat this problem [✓]	29.41%	82.35%
They perform mapping of user requirement to code	35.29%	00.00%
Reminds authors and reviewers about common errors and issues which needs to be handled [✓]	76.47%	82.35%
[5] Which of the following is preferable review code size?		
100-200 LOC	05.88%	00.00%
200-400 LOC [✓]	17.65%	58.82%
400-600 LOC	35.29%	41.18%
600-800 LOC	41.18%	00.00%
800 or more LOC	00.00%	00.00%
[6] What are some of the best code development practices which act as catalyst in peer code review process?		
Maintaining checklists of common issues and errors [✓]	76.47%	94.11%
Author preparation- adding annotations to code etc [✓]	64.70%	82.35%
Sending code for review only when the entire implementation phase is complete	23.52%	00.00%
Sending code modules for review as and when they are developed [✓]	35.29%	64.70%
Allowing all developers self select changes they are interested and competent to review	17.64%	00.00%
[7] Which of the following ensures that developers maintain a positive outlook towards code review?		
Incorporating review feedback in their performance evaluation	52.94%	23.53%
Providing reviewer complete freedom to carry out code review process	17.64%	00.00%
Ensuring that defect densities will never be used in performance reports [✓]	29.42%	64.70%
Allowing developers to skip code review if they are confident about their code	00.00%	11.77%

outside the team. Only 29.42% respondents opted to select team's senior co-developer with required expertise and experience to perform the task of code review. It is seen that 52.94% believe that incorporating review feedback in performance evaluation will foster a positive outlook among developers towards code review whereas only 29.42%

understand that defect densities should never be used in performance reports. These questions were asked with the intent to check the knowledge of respondents regarding the standard practices of peer code review.

Table 4.2: Questionnaire [Q8-Q11] Results before the Game Play

Select which form of review is more efficient in following categories			
	Tool assisted review	**Heavy weight review**	**Both**
[8] Cost	47.05 %	29.42 % [✓]	23.53%
[9] Time	70.59 % [✓]	05.88 %	23.53 %
[10] Review Effectiveness	23.53 %	41.18 %	35.29 % [✓]
[11] Resource Consumption	35.29 % [✓]	17.66 %	47.05 %

Table 4.3: Questionnaire [Q8-Q11] Results after the Game Play

Select which form of review is more efficient in following categories			
	Tool assisted review	**Heavy weight review**	**Both**
[8] Cost	35.30 %	64.70 % [✓]	0.00%
[9] Time	82.36 % [✓]	05.88 %	11.76 %
[10] Review Effectiveness	23.53 %	17.65 %	58.82 % [✓]
[11] Resource Consumption	47.05 % [✓]	11.77 %	41.18 %

Table 4.2 displays a set of inter related questions, answer choices (common for all questions) and the responses of 17 students for the same. We asked students these questions to test their knowledge of existing review processes. The questions test the efficiency of review processes when compared in different categories like cost of using that process, time taken to carry out review process, review effectiveness and resource consumption. It can be observed that 47.05% of students think that tool assisted review process are more efficient in terms of cost and only 29.42% are aware that manual review processes actually have low investment cost. The general belief that tool assisted reviews are better that manual review processes in every aspect is evident from Table 4.2, despite the fact that both of the review process find just as many bugs.

We asked such questions to test the knowledge of the students and investigate improvement after playing our game.

4.1.2 Post Game Survey Dataset

Post game survey dataset[1], holds the responses to the questionnaire we ask students after playing the game. Table 4.1, 4.3 presents the list of questions and their responses in the post-game questionnaire. The questions asked in post game survey are similar to that asked in pre game survey. The reason for keeping the questionnaire same is to investigate the amount of learning happening in respondent's knowledge, before and after playing the game.

Table 4.1 displays the questions, answer choices, correct answer(s) and the responses of 17 students after playing the game. Table 4.1 reveals that now 64.71% of the respondents selected the correct choice for the most optimal inspection rate for peer

code review and 58.82% selected the correct answer for optimal code review size. A remarkable improvement is seen among respondents i.e 100% agreed that team's senior co-developer with required expertise and experience should perform the task of code review. The respondents who preferred incorporating review feedback in performance evaluation now reduced to only 23.53%. Whereas 64.70% selected correct practice of not using defect densities in code review. A gradual improvement can be observed among the responses to questionnaire in the two datasets.

Table 4.3 displays a set of inter related questions on review processes and the responses of 17 students for the same. It can be observed that post game play only 35.30% of respondents think that tool assisted review process are more efficient in terms of cost, whereas 64.70% are now aware that manual review processes actually have low investment cost. Both of the processes may be leading in different categories but when it comes to review effectiveness, both of them finds almost similar bugs. 58.82% respondents selected the correct option of choosing both processes in review effectiveness category, whereas 23.53% still opted for tool assisted review and 17.75% selected manual review process.

4.1.3 Scoring Scheme for Datasets

We use these two dataset to obtain the results of improvement in learning, before and after playing the game. We come up with a scoring scheme to perform the evaluation.

[1]http://bit.ly/1GEdBBX

There are total 11 questions in each questionnaire. Every question weighs 1 mark. These questions have one, two or three correct answers. For each wrong answer, we give +0 marks and for each right option selected, +1/n marks are given (n is the number of correct options for that question). This way we score the performance of student's in each question and calculate their final performance score. As mentioned above, questions in survey have both single and multiple correct answers, therefore value of n varies from 1 to 3 (n=1 for Q2, 3, 5, 7, 8, 9, 10, 11; n=2 for Q4; n=3 for 1, 6).

4.2 Experimental Results

4.2.1 Pre Game Survey Dataset Results

Figure 4.3 represents the score values of students in pre game survey i.e. before playing the game. This score is obtained by following the scoring criteria described above. It can be observed that a mean score value of 4.44 is present for pre game survey. The score value of students vary from 1.49 (student ID 12) to 7.32 (student ID 3). The values obtained are scored out of a maximum score of 11. Values in Figure 4.3 will help in comparing the learning of student's that take place after the game play.

Figure 4.4 displays the score values for corresponding questions that we ask the students in pre game survey. The score is obtained using the responses of 17 students to the specified questions in pre game survey. Scoring the questions, makes it easy to identify the concepts or questions which respondents are aware of and could answer easily (like Q3, 5, 7, and 8) before playing the game. As seen from Figure 4.4, an average score of 6.86 (out of 17) is observed for the pre game questionnaire.

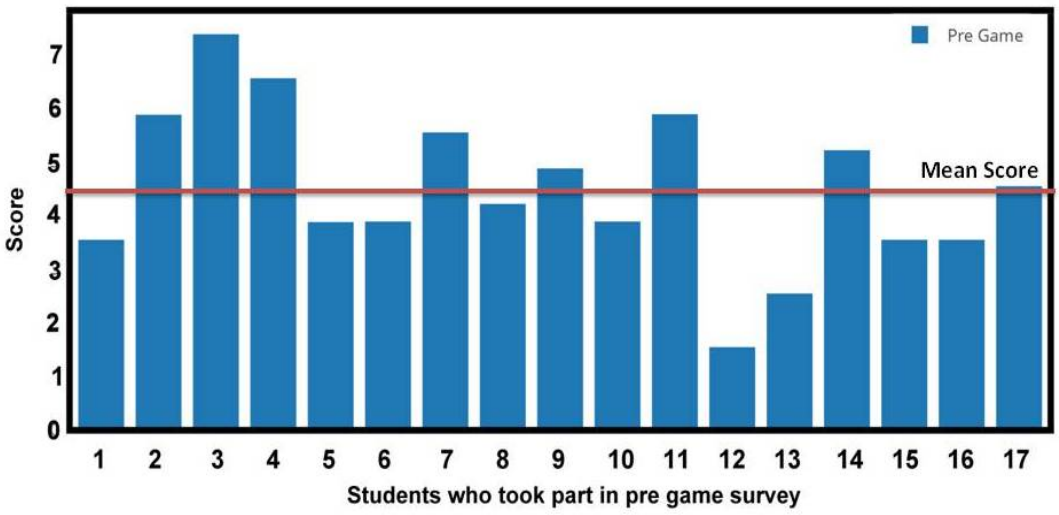


Figure 4.3: Survey score values (in pre game survey) of students who took part in the game evaluation

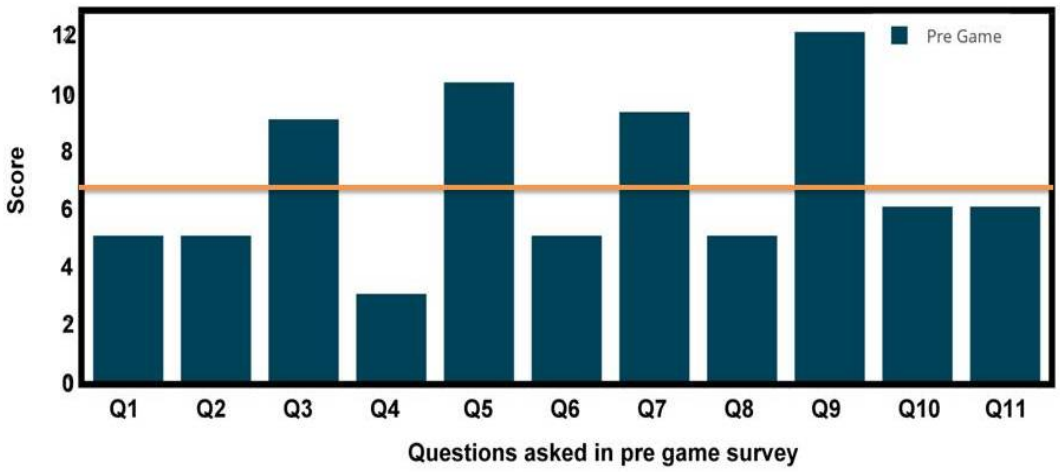


Figure 4.4: Survey score values (in pre game survey) for various questions asked to students while performing game evaluation

4.2.2 Game Play Results

Table 4.4 represents the student ID, game score and feedback that players gave to the game after playing it. It can be seen that most of the player's performance was average. Only 18% players scored poorly and were unable to complete the project assigned to them in game, on time and in budget. 47% players scored well and managed to get good scores, though none of the player's could manage to get an excellent performance remarks. An average mean score of 62 is observed in game play. We ask students to give a feedback on the simulation game. The feedback that we got shows that students find game more interactive and innovative method of teaching Software Engineering concepts.

Table 4.4: Student's game score and their feedback after playing the game

Student ID	Game Score	Game Performance	Feedback
1	53	Average	It was an interesting take on SE practices
2	72	Good	I experienced the difficulty of job of manager and this kind of survey helps to face such practical and real cases
3	82	Good	I learnt How code review should be performed
4	75	Good	This game was a very innovative and interesting technique to learn about a new thing peer code review beforehand. It gave a very good insight to how things actually happen in a company
5	22.5	Poor	It was good. Completely changed my outlook for the real time scenario development and peer review. Not only the quality , but other constraints like cost and time also have to be kept in mind and also that the developers are kept motivated
6	63	Average	It was an unusual way of learning SE concepts.Love d it, though I did not score well but it was a good experience
7	68	Average	This is a better way to teach such practices than rote learning them for an exam as it gives a real feel of handling such situations
8	76	Good	Could be made more effective.
9	16.5	Poor	It was a nice learning about peer code review process
10	89	Good	It was an excellent game giving nice learning experience
11	68	Average	Good game, nice GUI
12	63	Average	Very insightful game.
13	75	Good	It made me aware of a lot of new standard SE practices which I had no clue about
14	15.5	Poor	It was a nice concept to present peer code review in this way
15	58	Average	I liked the idea, though it should be refined more
16	80	Good	I enjoyed playing game
17	78	Good	It tested my knowledge about peer code review and helped me learn a lot of new things

4.2.3 Post Game Survey Dataset Results

Following the scoring criteria mentioned above, we calculate the improvement in survey (pre game and post game) score values for each student who took part in the evaluation. Table 4.5 presents the list of student's ID, their pre game survey score (P_r) (out of 11), post game survey score (P_o) (out of 11), improvement in score (P_o-P_r) and the improvement percentage equivalent ((P_o-P_r)/P_r)*100). It can be seen that the

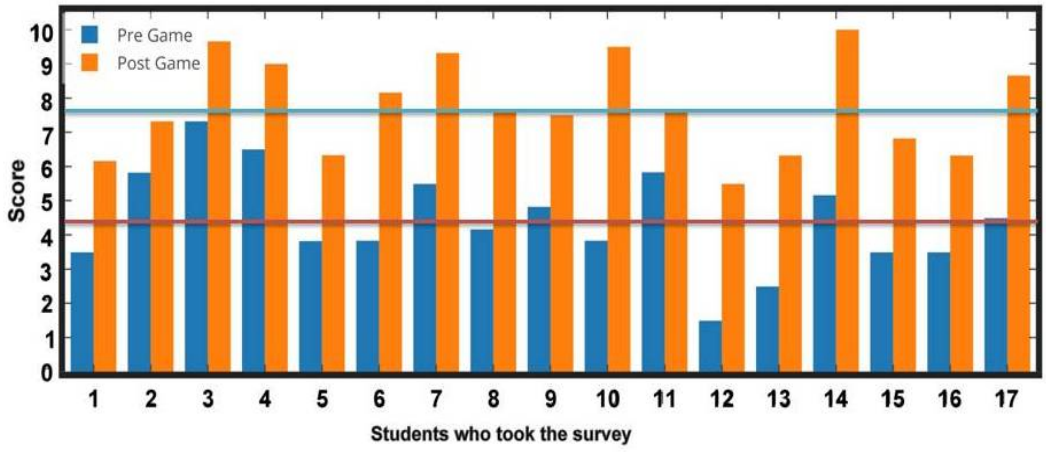


Figure 4.5: Survey score values (in pre game and post game survey) of students who took part in the game evaluation

Table 4.5: Improvement in survey score values for students

Student ID	Pre Game Survey Score (P_r)	Post Game Survey Score (P_o)	Improvement in score (P_o-P_r)	Improvement in score (%) ((P_o-P_r)/P_r)*100
1	3.49	6.16	2.67	76.50%
2	5.82	7.32	1.50	25.77%
3	7.32	9.66	2.34	31.96%
4	6.50	9.00	2.50	38.46%
5	3.82	6.33	2.51	65.70%
6	3.83	8.16	4.33	113.05%
7	5.49	9.32	3.83	69.76%
8	4.16	7.66	3.50	84.13%
9	4.82	7.50	2.68	55.60%
10	3.83	9.50	5.67	148.04%
11	5.83	7.66	1.83	31.38%
12	1.49	5.49	4.00	268.45%
13	2.49	6.32	3.83	153.81%
14	5.16	10.00	4.84	93.79%
15	3.49	6.82	3.33	95.41%
16	3.49	6.32	2.83	81.08%
17	4.49	8.66	4.17	92.87%
Average	**4.44**	**7.75**	**3.31**	**74.54%**

improvement in absolute term ranges from 1.5 (Student ID 2) to 5.67 (Student ID

10). An average mean score of 4.44 is obtained in pre game survey which increase to 7.75 in post game survey. Thus, there is an average improvement of 74.54% i.e. 3.31 in score post game play. Figure 4.5 displays a bar graph which represents the information of Table 4.5. Figure 4.5 shows the survey score values for every student, along with the lines displaying the mean score values for pre and post game survey.

Figure 4.6 is a bar graph representing the survey scores varying for each question, before and after game play. The values in Figure 4.6 are the values for each question that we obtained from the responses of 17 students who took part in evaluation. We follow the same scoring criteria that we use to draw Figure 4.5. It explores the improvement trend that exists for each and every question, before and after the game play. As visible from the graph, there are some questions which could be categorized easy as more than 50% of students could answer them correctly in pre game survey (Q1, 3, 5 and 9). Similarly there is another set of questions which students found hard and only 30% or less could answer correctly like Q2, 4, 6, 7 and 8. A mean pre game score of 6.86 is observed for the survey questions, which then raises to 11.98 in post game survey. Thus an average improvement of 74.63% (5.12 in absolute) is observed, which is similar to that observed in Figure 4.5. In absolute terms, a maximum improvement of 12 (Q7) and minimum improvement of 2 (Q9 and Q11) is observed. Table 4.6 displays all the statistics behind the formation of bar graph in Figure 4.6.

Table 4.6: Improvement in survey score values for questions

Question	**Pre Game Survey Score (P_r)**	**Post Game Survey Score (P_o)**	**Improvement in score (P_o-P_r)**	**Improvement in score (%) ((P_o-P_r)/P_r)*100**
1	9.25	12.27	3.02	32.64%
2	5.00	11.00	6.00	120.00%
3	9.00	14.00	5.00	55.55%
4	3.00	10.00	7.00	233.33%
5	10.27	13.61	3.34	32.52%
6	5.00	11.00	6.00	120.00%
7	5.00	17.00	12.00	240.00%
8	5.00	11.00	6.00	120.00%
9	12.00	14.00	2.00	16.66%
10	6.00	10.00	4.00	66.66%
11	6.00	8.00	4.00	33.33%
Average	**6.86**	**11.98**	**5.12**	**74.63%**

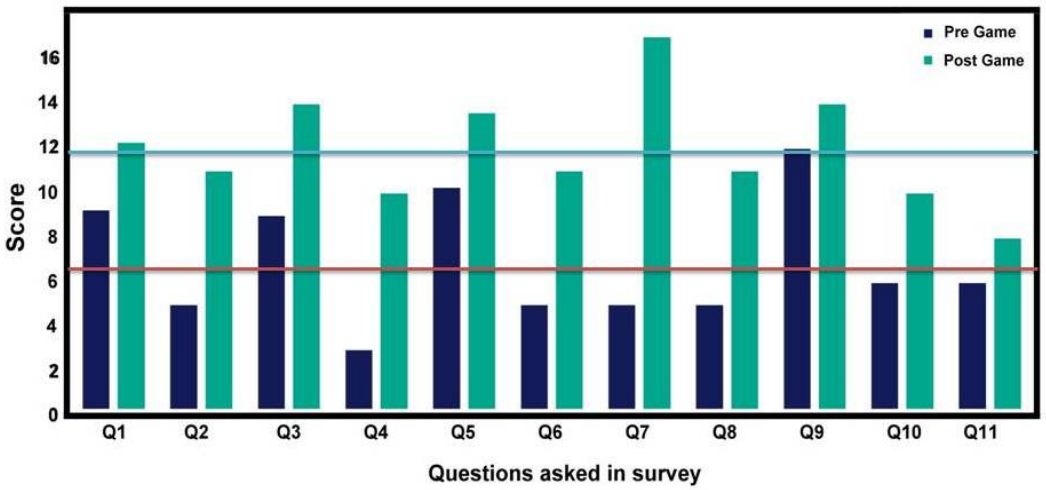


Figure 4.6: Survey score values (in pre game and post game survey) for various questions asked to students while performing game evaluation

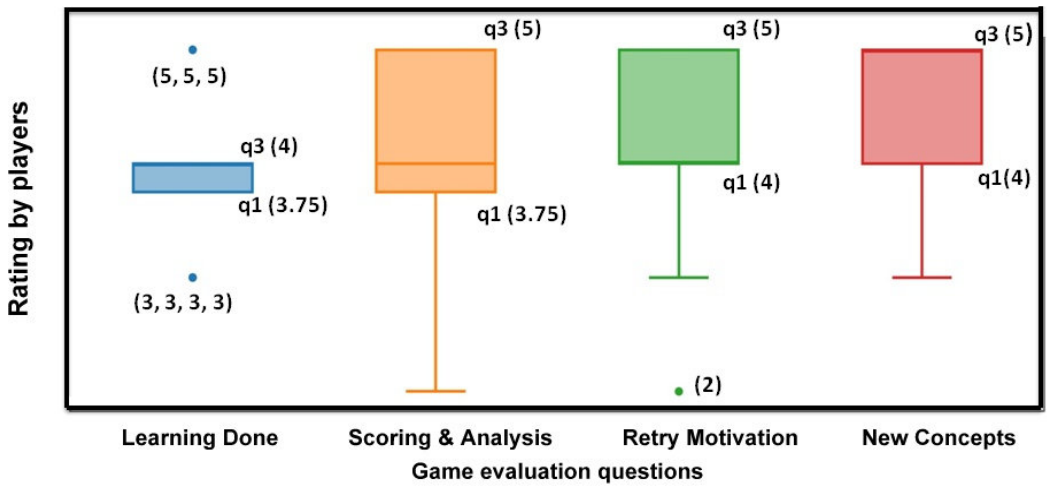


Figure 4.7: Rating distribution in post game survey by players of the game evaluating learning done through the game, scoring and analysis provided at the end of game, motivation to try again on failure and introduction of new concepts

We perform post game survey once the students are done playing the game. It is divided into two sections. First section (Figure 4.1) contains all the questions specified

in Table 4.1 and 4.3, to test students learning after the game play. There is another section (refer to Figure 4.2) which requires the player to rate the game on different aspects like learning done throughout the game, score and analysis provided at the end of game, motivation to try again on strategy or decision failure and introduction of new concepts or practices which student was not aware of. Figure 4.7 represents the box plot demonstrating the distribution of ratings provided by the player to game in these categories. It can be seen that learning done throughout the game has a median of 4 with most of the rating distribution spread from 3.75 to 4 with 7 outliers ({3, 3, 3, 3}, {5, 5, 5}) as visible from Figure 4.7. Similar information can be obtained about other aspects too from Figure 4.7.

5

Limitations and Future Work

The experiments that we have performed are conducted out-of class on a very small number of students i.e. 17. Future work involves performing in and out-of class experiments with more number of students.

Peer code review is a vast field and we have incorporated only 12 standard practices as the learning objectives in our game. Future work involves exploring other aspects of peer code review as well.

There is no feedback or involvement of industry people in the game. Future work involves exposing our work to industrial environment.

The UI of game has limited inter activeness and thus more work need to be done on improving UI. Adding animations, sound effects etc. will be used in future extension.

It is a single player, role playing game. Future scope includes adding more designations other than Project Manager, introducing multi user mode, handling multiple projects etc. in game.

6

Conclusion

We describe an educational and interactive software engineering simulation game to teach important peer code review concepts and best practises. We define 12 learning objectives covering various aspects of peer code review and design our game as well as scoring system to teach the pre-defined learning goals. We evaluate the effectiveness of our approach by conducting experiments involving 17 undergraduate and graduate students. We observe a variety of responses and behaviour across various players. We found that students lack basic knowledge of peer code review and it's standard industrial practices. The tool exposes students to different aspects of peer code review along with the complexities involved in project management. Our experiments reveal that there is a significant improvement in the knowledge of the participants after playing the game.

The proposed learning framework and model based on discovery learning and learning from failure, has relevant impact on student's knowledge in short time. Evaluation of learning framework and tool reveals the potential use of such simulation games to teach SE concepts and practices. From a practical point of view, simulation games can also be used as an alternative to test concepts taught in classroom lectures. Providing evidence and reasoning to students on their decisions motivates them to think and retry again, in case of failure. Scoring the decisions, providing feedback, presenting unexpected scenarios to students engage them and encourage them to perform better. Through this tool assisted teaching methodology we try to bridge the gap between student's conceptual and practical knowledge.

References

[1] **Coverity Scan: 2012 Open Source Integrity Report**. *2012-Coverity-Scan-Report.pdf*. xii, 22, 25

[2] Deborah A Trytten. **A design for team peer code review**. In *ACM SIGCSE Bulletin*, **37**, pages 455–459. ACM, 2005. 1, 17

[3] Jason Cohen, Eric Brown, Brandon DuRette, and Steven Teleki. *Best kept secrets of peer code review*. Smart Bear, 2006. 2, 4, 16

[4] Michael Fagan. **Design and code inspections to reduce errors in program development**. In *Software pioneers*, pages 575–607. Springer, 2002. 3

[5] Anke Drappa and Jochen Ludewigé. **Simulation in Software Engineering Training**. In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE '00, pages 199–208, 2000. 5, 7, 9, 14

[6] Dale Callahan and Bob Pedigo. **Educating experienced IT professionals by addressing industry's needs**. *IEEE software*, (5):57–62, 2002. 6

[7] Emily Oh Navarro and André van der Hoek. **SimSE: An Educational Simulation Game for Teaching the Software Engineering Process**. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '04, pages 233–233, 2004. 6, 7, 9, 14

[8] Nauman bin Ali and Michael Unterkalmsteiner. **Use of simulation for software process education: a case study**. 6

[9] Emily Oh Navarro and André Van Der Hoek. **On the role of learning theories in furthering software engineering education**. *Software Engineering: Effective Teaching and Learning Approaches and Practices, IGI Global*, 2008. 7

[10] Marcio Barros Alexandre Dantas and Claudia Werneré. **A Simulation-Based Game for Project Management Experiential Learning**. In *Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering*, SEKE'04, pages 19–24, 2004. 10, 14

[11] Katherine Shaw and Julian Dermoudy é. **Engendering an Empathy for Software Engineering**. In *Proceedings of the 7th Australasian Conference on Computing Education*, ACE'05, pages 135–144, 2005. 10, 14

[12] Emily Oh Navarro Alex Baker and André van der Hoek. **An experimental card game for teaching software engineering processes**. In *The Journal of Systems and Software*, pages 3–16, 2005. 10, 14

[13] Apurva Jain and Barry Boehmé. **SimVBSE: Developing a Game for Value-Based Software Engineering**. In *Proceedings of the 19th Conference on Software Engineering Education and Training*, pages 103–114, 2006. 10, 14

[14] Alexandros Theodoridis Nicholaos Petalidis, Gregory Gregoriadis and Antonios Chronakisé. **PROMASI A PROject Management SImulator**. In *Proceedings of the 2011 15th Panhellenic Conference on Informatics*, PCI '11, pages 33–37, 2011. 10, 14

[15] Rafael Savi Christiane Gresse von Wangenheim and Adriano Ferreti Borgattoé. **DELIVER! An educational game for teaching Earned Value Management in computing courses**. **54**, pages 286–298. 10, 14

[16] Elke Hochmller Susanne Jger Roland Mittermeir1, Andreas Bollin1 and Daniel Wakounig1é. **AMEISE: An Interactive Environment to Acquire Project-Management Experience**. 2013. 11, 14

[17] Peter C Rigby, Daniel M German, and Margaret-Anne Storey. **Open source software peer review practices: a case study of the apache server**.

In *Proceedings of the 30th international conference on Software engineering*, pages 541–550. ACM, 2008. 16, 17

[18] Peter C Rigby, Brendan Cleary, Frederic Painchaud, Margaret-Anne Storey, and Daniel M German. **Contemporary peer review in action: Lessons from open source development**. *Software, IEEE*, **29**(6):56–61, 2012. 16, 17

[19] Yanqing Wang, LI Yijun, Michael Collins, and Peijie Liu. **Process improvement of peer code review and behavior analysis of its participants**. In *ACM SIGCSE Bulletin*, **40**, pages 107–111. ACM, 2008. 17

[20] Roger B Myerson. *Game theory*. Harvard university press, 2013. 22

[21] KN Jha and KC Iyer. **Commitment, coordination, competence and the iron triangle**. *International Journal of Project Management*, **25**(5):527–540, 2007. 24