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

Ritika Atal* and Ashish Sureka†
*Indraprastha Institute of Information Technology, Delhi (IIIT-D), India
Email: ritika13103@iiitd.ac.in
†ABB (India)
Email: ashish.sureka@in.abb.com

Abstract—**Application of educational and interactive simulation games to teach important concepts is an area that has attracted several Software Engineering researchers and educators attention. Previous research and studies on usage of simulation games in classroom to train students have demonstrated positive learning outcomes. Peer code review is a recommended best practice during software development which consists of systematically examining the source code of peers before releasing the software to Quality Assurance (QA). Practitioners and Researchers have proposed several best practices on various aspects of peer code review such as the size of code to be reviewed (in terms of lines of code), inspection rate and usage of checklists. We describe a single player educational simulation game to train students on best practices of peer code review. We define learning objectives, create a scenario in which the player plays the role of a project manager and a scoring system (as a factor of time, budget, quality and technical debt). We design a result screen showing the trace of events and reasoning (learning through success and failure as well as discovery) behind the points awarded to the player. We conduct a survey of the players by conducting a quiz before and after the game play and demonstrate the effectiveness of our approach.**

Keywords*—*Peer Code Review, Simulation Game, Software Engineering Education and Training, Teaching Critical Decision Making

I. RESEARCH MOTIVATION AND AIM

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 [1][2]. Peer code review consists of reviewing and critiquing team members source code in order to detect defects and improve the quality of the software [3][4][5][6][7]. Software code review is practiced in several open-source and closed-source software project settings. There are several best practices on various aspects of peer code review such as the code review size, coverage and rate. Code reviewer expertise, reviewer checklist and usage of tools (such as mailing lists, Gerrit or Rietveld) also play an important role in influencing the impact of code review on software quality [5][6]. 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. The research aim of the work presented in this paper 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.

II. RELATED WORK & RESEARCH CONTRIBUTIONS

In this Section, we discuss closely related work and state our novel research contributions in context to the related work. We conduct a literature survey of papers published on the topic of teaching Software Engineering concepts using simulation games. Table I shows the result of our literature review. Table I lists 9 papers 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 until 2013. We characterize 9 papers based on the tool name, year, simulation topic, University and interface. We infer that teaching software engineering processes and project management are the two most popular target areas for simulation games. The game interfaces various from simple command line and menu driven model to animated and 3D interfaces. Researchers have also experimented with board games in addition to computer-based games. In context to closely related work, the study presented in this paper 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

TABLE I. 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 [8]	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 [9]	2012 [Wangenheim'12]	Earned Value Management	Federal University of Santa Catarina	Board Game
3	ProMaSi [10]	2011 [Petalidis'11]	Project Management	T.E.I. of Central Macedonia, Serres-Greece	Java based, Desktop environment
4	SimVBSE [11]	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 [13]	2005 [Shaw'05]	Software Process Knowledge	University of Tasmania	Menu-based, Graphical Interface
7	Incredible Manager [14]	2004 [Dantas'04]	Project Management Experiential Learning	Brazilian University	Button Driven, Command Interface
8	SimSE [1]	2004 [Navarro'04]	Software Engineering Processes	University of California, Irvine	Web-based, Graphical Interface
9	SESAM [15]	2000 [Drappa'00]	Software project Management	Stuttgart University, Germany	Text Based, Pseudo-Natural Language Command Interface

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.

III. GAME ARCHITECTURE AND DESIGN

A. Learning Objectives

In this game we define 12 learning objectives covering multiple aspects of peer code review. These learning objectives are captured in our pre game questions[1] and post game questions[2]. Table II shows 6 of the 12 learning objectives (due to space constraint), corresponding decisions and the questions which we asked to the player in game. Table II 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 1), what inspection rate to choose (Figure 2), when to start with review, steps required to foster good code review culture in team etc. A decision can map to one or more learning objectives. Similarly two or more evaluation questions may map to one decision.

B. 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 unobservable and our goal is to examine the response and the decision making ability of the player to unexpected circumstances. Figure 3 shows a screenshot for the simulation game in which the player is presented with an unexpected situation. As shown in Figure 3, the project manager is encountered with a situation wherein a developer quits the team or organization just one month before the release date.

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

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

C. Scoring System

1) Technical Debt: We apply Technical Debt[3] 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 4 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 4 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 1, 2, 3 and 4).

$$deviationValue = (standardTD - decisionWeight) \quad (1)$$

$$sumTD = sumTD + deviationValue \quad (2)$$

$$TD_{avg} = sumTD/decisionCounter \quad (3)$$

$$TD = (TD_{avg}/standardTD) * 100 \quad (4)$$

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 5). Series of decisions taken by a player either

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

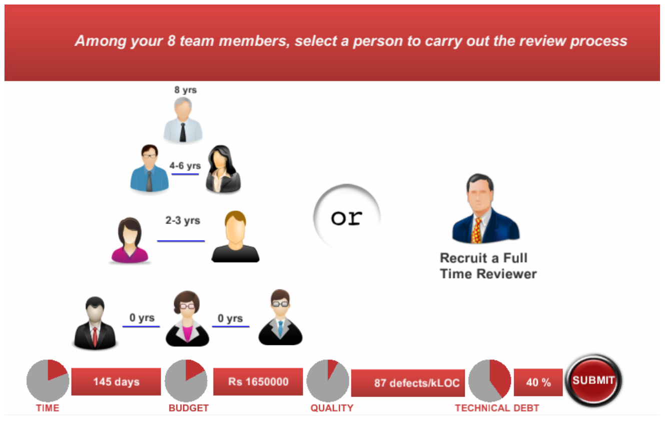
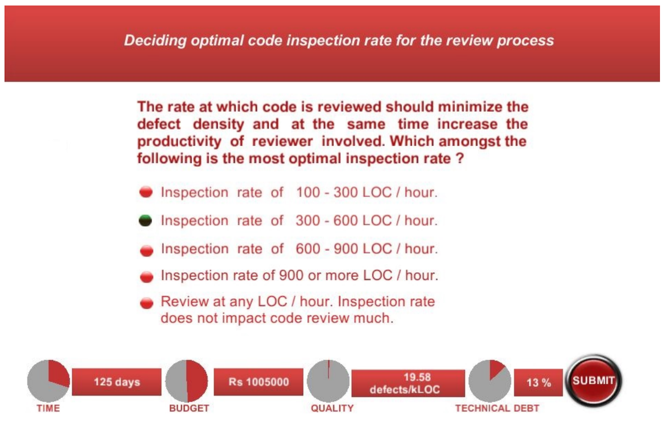


Fig. 1. Screenshot of the game in which the player needs to make a decision on developer expertise for a code review assignment


Deciding optimal code inspection rate for the review process

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 ?

- Inspection rate of 100 - 300 LOC / hour.
- Inspection rate of 300 - 600 LOC / hour.
- Inspection rate of 600 - 900 LOC / hour.
- Inspection rate of 900 or more LOC / hour.
- Review at any LOC / hour. Inspection rate does not impact code review much.

Fig. 2. Screenshot of the game in which the player needs to make a decision on the best practice of code review rate

TABLE II. 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	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	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	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	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	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	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. Choose what steps will you take to handle this situation?

prevent them from meeting the deadline or they are able to complete the project successfully. All of the decisions have a path time associated with them. This path time gets deducted from the remaining time as the player proceeds further in game (Equation 5 and 6).

$$remainingTime = remainingTime - pathTime \tag{5}$$

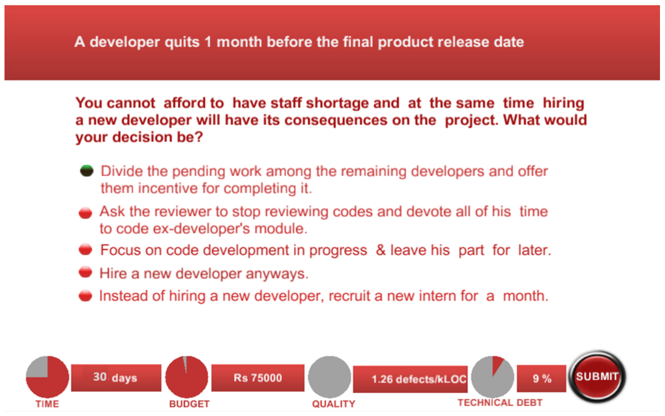
$$timeScore\ (days) = remainingTime \tag{6}$$

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 6 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 7 and 8 to see how path cost for player's decision is used to obtain the cost score.

$$remainingCost = remainingCost - pathCost \tag{7}$$

$$costScore\ (Rs) = remainingCost \tag{8}$$

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[16]. Figure 7 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.



A developer quits 1 month before the final product release date

You cannot afford to have staff shortage and at the same time hiring a new developer will have its consequences on the project. What would your decision be?

- Divide the pending work among the remaining developers and offer them incentive for completing it.
- Ask the reviewer to stop reviewing codes and devote all of his time to code ex-developer's module.
- Focus on code development in progress & leave his part for later.
- Hire a new developer anyways.
- Instead of hiring a new developer, recruit a new intern for a month.

30 days | Rs 75000 | 1.26 defects/kLOC | 9 %

TIME | BUDGET | QUALITY | TECHNICAL DEBT

SUBMIT

Fig. 3. Screenshot of the game in which the player is presented with an unexpected situation consisting of developer attrition

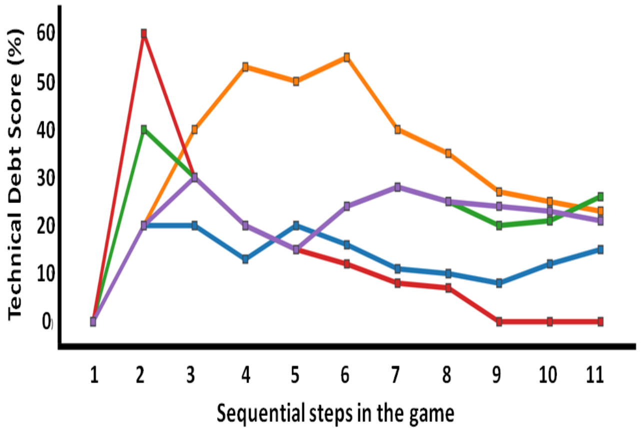


Fig. 4. Technical debt values (and increasing or decreasing trends) for various players during the execution of the game

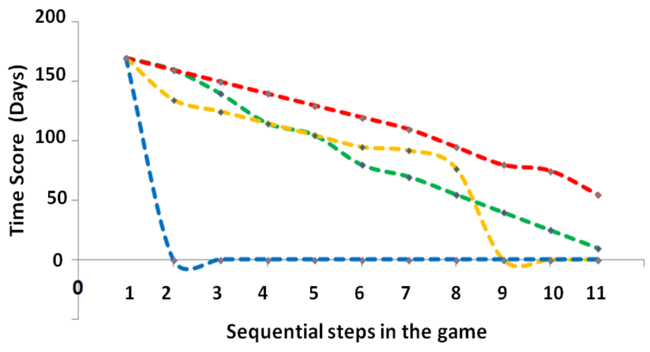


Fig. 5. Time values (and increasing or decreasing trends) for various players during the execution of the game

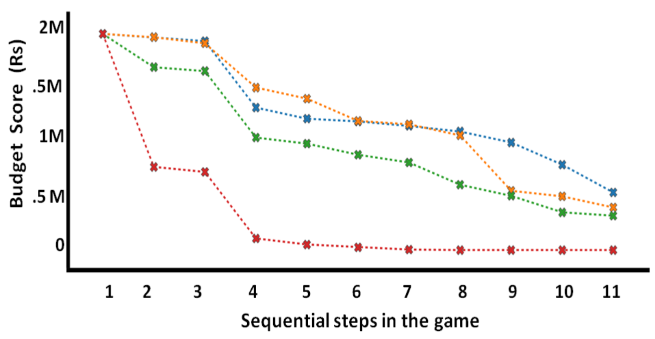


Fig. 6. Budget values (and increasing or decreasing trends) for various players during the execution of the game

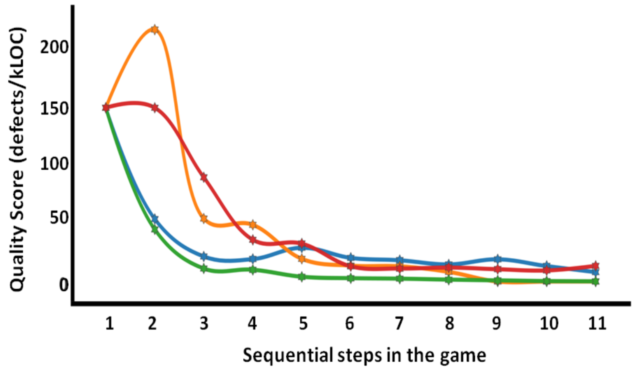


Fig. 7. Project quality values (and increasing or decreasing trends) for various players during the execution of the game

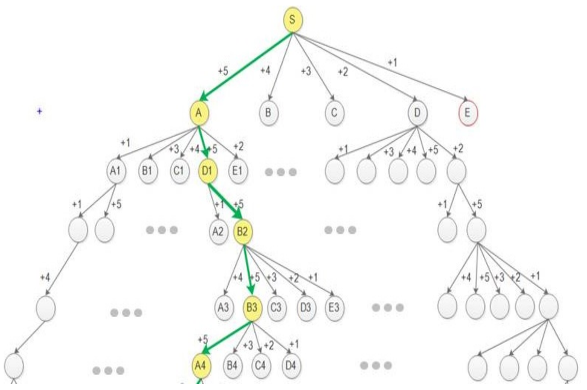


Fig. 8. Screenshot of the game tree (upto 5 levels) demonstrating the different game paths and weights associated with each path of the game

Depending on player's decision, this defect% either increases or decreases the software quality standards. Equation 9, 10 and 11 captures project quality (projectQlty) calculation.

$$decisionQlty = defect\% * projectQlty \tag{9}$$

$$projectQlty = projectQlty \pm decisionQlty \tag{10}$$

$$qualityScore\ (defects/kLOC) = projectQlty \tag{11}$$

D. 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 8) 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) [17]. Figure

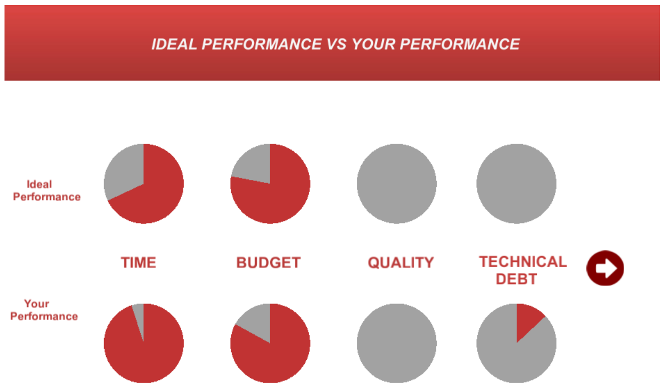
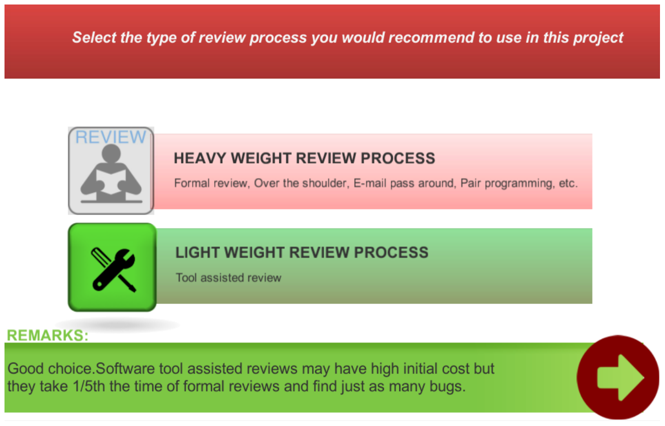


Fig. 9. Screenshot of the score meters comparing the player's performance with that of an ideal performance at the end of game


Select the type of review process you would recommend to use in this project

REVIEW

HEAVY WEIGHT REVIEW PROCESS

Formal review, Over the shoulder, E-mail pass around, Pair programming, etc.

LIGHT WEIGHT REVIEW PROCESS

Tool assisted review

REMARKS:

Good choice.Software tool assisted reviews may have high initial cost but they take 1/5th the time of formal reviews and find just as many bugs.

Fig. 10. Screenshot of the game in which the player is presented with analysis of his decision to recommend the use of tool assisted review

8 represents the game tree for our game. It has a branching factor of 4 with a solution depth of 9. The time complexity for its traversal is $O(4^9)$.

E. 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 8). As the player proceeds in game, the weight associated with each decision keeps on accumulating and is stored in P_{sum} (refer to Equation 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.

$$P_{sum} = \sum W_d \; (out\ of\ 50) \tag{12}$$

2) Next we scale and obtain the equivalent values for cost (C_f), time (T_f) and quality (Q_f) remaining at the end of game. Project attributes sum (P_{attr_sum}) holds the average of equivalent values of these three project attributes (Equation 13) [18].

$$P_{attr\_sum} = (C_f + T_f + Q_f)/3 \; (out\ of\ 50) \tag{13}$$

3) Score sum (S_f') is obtained by adding the values calculated in step 1 and 2 (Equation 14).

$$S'_f(out\ of\ 100) = P_{sum} + P_{attr\_sum} \tag{14}$$

4) We then make use of TD_{avg} (refer to Equation 3) to calculate the final score (S_f) in equation 15.

$$S_f = S'_f * (1 - TD_{avg}/standardTD) \tag{15}$$

F. 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 in Figure 1, 2 and 3). 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 9). 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 to recruit a full time reviewer (refer to screenshot in Figure 1) 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 2) 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 (Figure 10 shows remarks provided to player).

IV. Experimental Analysis and Performance Evaluation

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. Table III, IV presents the list of questions and their responses in the pre-game questionnaire[4].

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

TABLE III. QUESTIONNAIRE RESULTS BEFORE THE GAME PLAY

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

TABLE IV. QUESTIONNAIRE 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 %

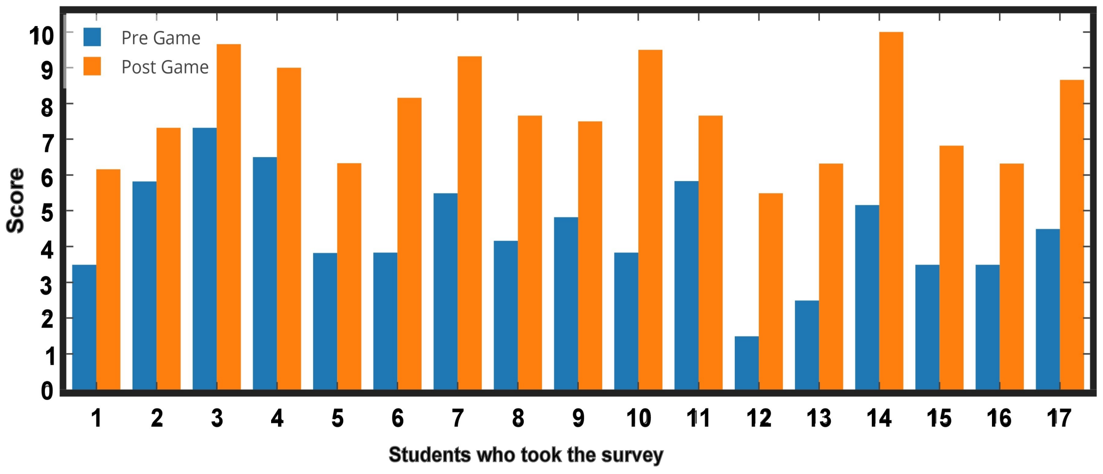


Fig. 11. Survey score values (in pre game and post game survey) of students who took part in the game evaluation

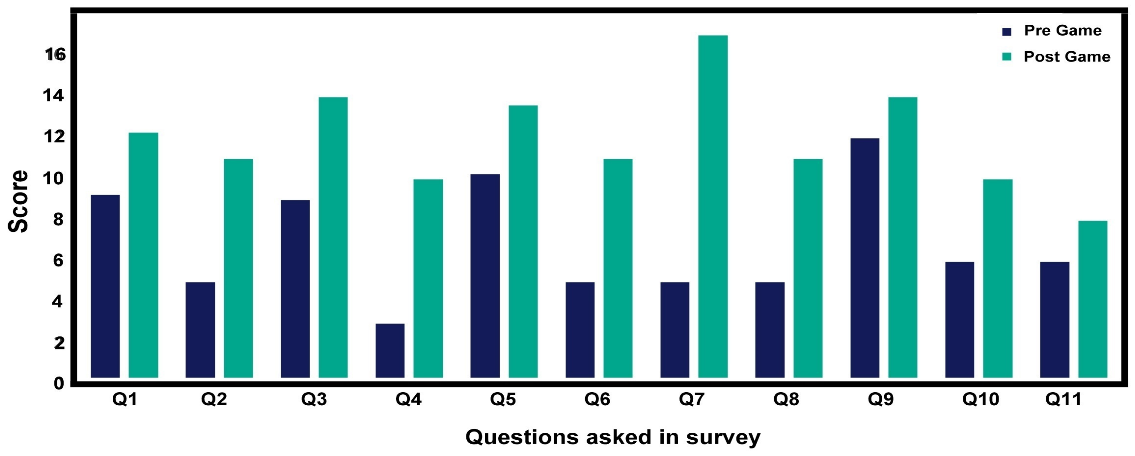


Fig. 12. Survey score values (in pre game and post game survey) for various questions asked to students while performing game evaluation

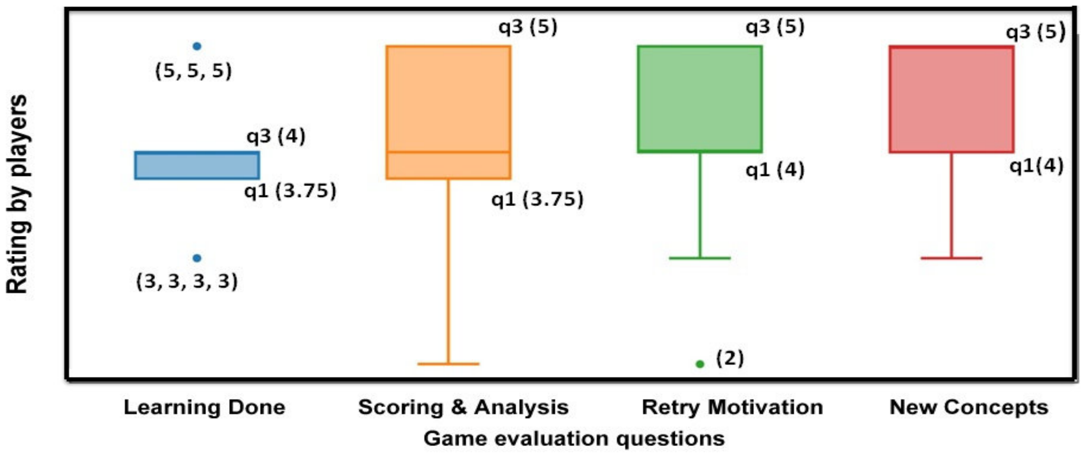


Fig. 13. Rating distribution in post game survey by players of the game evaluating learning done through the game, scoring and analysis provided at the end of game, motivation to try again on failure and introduction of new concepts

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 III displays the questions, answer choices, correct answer(s) and the responses of 17 students. Table III reveals that 30% of the respondents selected the correct choice for the most optimal inspection rate for peer code review and only 18% selected the correct answer for optimal code review size. We asked such questions to test the knowledge of the students and investigate improvement after playing our game. As shown in Figure III, our pre-game questionnaire consists of questions on various aspects of peer code review.

Table IV 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 is 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 IV, despite the fact that both of the review process find just as many bugs.

As observed from Table III and IV, there are total 11 questions in the survey questionnaire, which we made mandatory for each student to answer. To obtain the score of pre and post game survey questions we assign a weight age of 1 mark to each question. For every wrong response +0 marks were given and for each right response selected +1/n marks were given (n is the number of correct answers for that question). 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). Using this scoring criteria, we made a bar graph to see the improvement in score before and after playing the game (refer to Figure 11). It can be seen that mean score of students in pre game survey is 4.44 which increases to 7.75 in post game survey. Thus, there is an average improvement of 74.54% i.e. 3.31 in score. In terms of absolute score a maximum improvement of 5.67 (Student ID 10) and minimum improvement of 1.5 (Student ID 2) can be seen.

Figure 12 is a bar graph representing how survey scores vary for each question, before and after game play. The values in Figure 12 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 11. 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, Q4, Q6, Q7 and Q8. A mean pre game score of 6.86 is observed for these 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 11. In absolute terms, a maximum improvement of 12 (Q7) and minimum improvement of 2 (Q9 and Q11) is observed.

We perform post game survey once the students are done playing the game. It is divided into two sections. First section contains all the questions specified in Table III and IV, to test student's learning after the game play. There is another section 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 13 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, 5) as visible from Figure 13. Similar information can be obtained about other aspects too from Figure 13.

V. CONCLUSION

We describe an education 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 its standard industrial practices. Our experiments reveal that there is a significant improvement in the knowledge of the participants after playing the game. We observe that students find simulation game more engaging and interesting platform for learning.

REFERENCES

[1] E. O. Navarro and A. 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, pp. 233–233, 2004.

[2] N. bin Ali and M. Unterkalmsteineré, "Use of simulation for software process education:a case study,"

[3] M. Mittal and A. Sureka, "Process mining software repositories from student projects in an undergraduate software engineering course," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, (New York, NY, USA), pp. 344–353, ACM, 2014.

[4] R. Mishra and A. Sureka, "Mining peer code review system for computing effort and contribution metrics for patch reviewers," in *Mining Unstructured Data (MUD), 2014 IEEE 4th Workshop on*, pp. 11–15, Sept 2014.

[5] P. C. Rigby, D. M. German, L. Cowen, and M.-A. Storey, "Peer review on open-source software projects: Parameters, statistical models, and theory," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, pp. 35:1–35:33, Sept. 2014.

[6] S. McIntosh, Y. Kamei, B. Adams, and A. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, pp. 1–44, 2015.

[7] S. Sripada, Y. Reddy, and A. Sureka, "In support of peer code review and inspection in an undergraduate software engineering course," in *Software Engineering Education and Training (CSEET), 2015 IEEE 28th Conference on*, pp. 3–6, May 2015.

[8] E. H. S. J. Roland Mittermeir1, Andreas Bollin1 and D. Wakounig1é, "Ameise: An interactive environment to acquire project-management experience," 2013.

[9] R. S. Christiane Gresse von Wangenheim and A. F. Borgattoé, "Deliver! an educational game for teaching earned value management in computing courses," vol. 54, pp. 286–298.

[10] A. T. Nicholaos Petalidis, Gregory Gregoriadis and A. Chronakisé, "Promasi a project management simulator," in *Proceedings of the 2011 15th Panhellenic Conference on Informatics*, PCI '11, pp. 33–37, 2011.

[11] A. Jain and B. Boehmé, "Simvbse: Developing a game for value-based software engineering," in *Proceedings of the 19th Conference on Software Engineering Education and Training*, pp. 103–114, 2006.

[12] E. O. N. Alex Baker and A. van der Hoek, "An experimental card game for teaching software engineering processes," in *The Journal of Systems and Software*, pp. 3–16, 2005.

[13] K. Shaw and J. D. é, "Engendering an empathy for software engineering," in *Proceedings of the 7th Australasian Conference on Computing Education*, ACE'05, pp. 135–144, 2005.

[14] M. B. Alexandre Dantas and C. 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, pp. 19–24, 2004.

[15] A. Drappa and J. Ludewigé, "Simulation in software engineering training," in *Proceedings of the 22nd International Conference on Software Engineering*, ICSE '00, pp. 199–208, 2000.

[16] "Coverity scan: 2012 open source integrity report," *2012-Coverity-Scan-Report.pdf*.

[17] R. B. Myerson, *Game theory*. Harvard university press, 2013.

[18] K. Jha and K. Iyer, "Commitment, coordination, competence and the iron triangle," *International Journal of Project Management*, vol. 25, no. 5, pp. 527–540, 2007.