Reinforcing Software Engineering Learning Through Provenance

Troy C. Kohwalter Esteban W. G. Clua Leonardo G. P. Murta
Instituto de Computação
Universidade Federal Fluminense
Niteroi – RJ, Brazil
{tkohwalter, esteban, leomurta}@ic.uff.br

***Abstract*—Software engineering is focused on practical and theoretical aspects of the software production. Teaching software engineering is traditionally done through theoretical classes with some practical exercises. Recently, games and simulators were introduced as a ludic alternative for software engineering learning, where decisions and interactions become key factors to transmit and acquire knowledge. However, mistakes made by wrong decisions may jeopardize the learning process, especially when reproducing its effects is not a viable option due to the non-deterministic nature of games. With this in mind, in a previous work we proposed a novel approach based on provenance concepts in order to present the decisions and effects of such decisions when learning through games. In this work, we present an experimental evaluation of that approach with undergraduate students. The obtained results show that the use of provenance leads to faster and more accurate answers from students, including learning aspects that could not be achieved by a traditional educational game.**

Keywords—software engineering; serious games; provenance; education; game flux.

I. INTRODUCTION

Traditional Software Engineering teaching process consists of lectures and usually a course project, which has the intent of applying the theory in a practical situation. Moreover, these projects are restricted to the length of the course, which limits the opportunities for the students to practice and comprehend all the concepts taught in classroom. Also, due to time constraints, most course projects occurs in a straightforward fashion that leaves little room for experiencing the many facets of the software lifecycle. Lastly, these course projects typically focus on project deliverables, which usually do not stimulate the student's interest. In order to solve this problem, software engineering games [1]–[5] have been used for helping students to learn concepts taught in classrooms by stimulating curiosity and providing motivation for learning.

However, the outcomes of a digital game session derive from a series of decisions and actions made throughout the game. In many situations, analyzing and understanding the events, mistakes, and fluxes of a concrete game session may be useful for understanding the achieved results. Game session analysis is also fundamental for detecting symptoms of problems that occurred due to wrong decision-making and to better understand if the student learned the concepts presented by the game. Without a game flux analysis, the student would be required to play the game multiple times to intuitively guess which actions were incorrect. Similarly, the tutor would be required to watch the game being played in order to identify the mistakes made by the student. Depending on the game dynamics and its complexity, reproducing the same state can be unviable, making it difficult to adopt a trial and error approach.

In our previous work [6], we introduced the usage of digital provenance[1] in games. The main goal of the previous work was to propose a conceptual framework that collects information during a game session and maps it to provenance terms, providing the means for a post-game analysis. We applied this conceptual framework over a serious game named SDM [8], which focus on teaching Software Engineering processes. The provenance support in SDM allowed for a broader range of analysis by using collected game session provenance information to generate a provenance graph [9].

Students and tutors can use this provenance information to identify cause-and-effect relations amongst actions made during a game session in order to understand the outcomes. The provenance analysis process collects data and generates a provenance graph, relating actions, decisions, and events that occurred throughout the game in a high level model. This high level model allows a broader range of analysis and data mining. For instance, the provenance graph allows browsing the data, identifying actions that influenced specific outcomes. It also helps to understand how events were generated and which decisions contributed to them. This process also aids in the identification of mistakes, allowing students to reflect upon them for future interactions or allowing tutors to know which concepts students are having difficulties. Thus, this knowledge can help on (1) confirming the hypotheses formulated by students, (2) supporting tutors for a better guidance, (3) motivating practical exercises around some case studies, and (4) extracting behavior patterns from individual sessions or groups of sessions.

The goal of this paper is to evaluate if the use of provenance during a game analysis aids students to understand the underlying reasons for the game outcome. This is accomplished by using the provenance visualization tool *Prov Viewer* [9], which was customized to work with SDM to

[1]Provenance refers to the documented history of an object's life cycle and is generally used in the context of art, digital data, and science [7].

978-1-4799-4223-7/14 $31.00 © 2014 IEEE
DOI 10.1109/SBES.2014.16

visualize the provenance graph. We evaluated our approach with different undergraduate classes to assess the viability of analyzing a game session by using provenance. The goals of these experiments were to know if the provenance analysis is more efficient and effective than analyzing the game by re-watching the session, trying to observe if it has a higher identification rate of the cause-and-effect relations between the actions and their outcomes. To do so, we answered the following research questions:

1. Does provenance analysis help to understand events that emerged during the game?
2. Is provenance analysis faster than only watching a replay of the game session?
3. Is provenance analysis more accurate than only watching a replay of the game session?

The rest of the paper is organized as follows: Section 2 presents some related work. Section 3 presents SDM with provenance support and *Prov Viewer*. Section 4 explains the experiment with two undergraduate classes, while Section 5 details our findings about using provenance analysis to understand the game session. Section 6 concludes and presents future works.

II. Related Work

Exposing students to choices and decisions increase the motivation and understanding process. For this reason, games are being used as a powerful teaching tool, including in Software Engineering. Navarro and van der Hoek [3] conceived a Software Engineering simulation digital game called SimSE. The purpose of this game is to address a gap in the traditional techniques of Software Engineering teaching, where students are exposed to various concepts and theories, but have few opportunities to apply these ideas into practice. In SimSE, the player assumes the position of a project manager who has a team of developers and manages the software development process by hiring and firing developers, monitoring development progress, assigning tasks, and buying new tools. The fundamental goal of the SimSE project is to allow the customization of the simulated process model and therefore to be used by professors during the presentation of content related to the software life cycle.

SimSE provides an explanatory tool that includes plotting graphics showing game values and action details, such as when an action started and ended (e.g., creating requirement document), the participants involved with the action (employees, tools, and artifacts), and game rules associated with it. It also shows all triggers and destroyers for each action, displaying what could have caused the action to start and end. The explanatory tool is a powerful analysis mechanism that shows details for each clock-tick in the game.

However it does not show details about how each participant of each action contributed to it, nor their respective contribution values. It is not clear how well SimSE can show cause-and-effect relationships. However it was stated by the author that the graphs in their explanatory tool is not helpful due to the difficulty of formulating a meaningful object and action graph combination that produces an insightful composite graph. The true usefulness of the tool lies in rule descriptions.

Dantas et al. [1] present a simulation based digital game for teaching Software Engineering, named *The Incredible Manager*. The focus of this game is project management, where the player main tasks are planning and managing software development projects. As a project manager, the player establishes a development plan for the project, estimate the duration of each task, assign tasks to developers, negotiate with stakeholders, control how long the team will work per day, and determine the effort spent on quality assurance. One important limitation reported by players was the inability to trace and explain each action and their consequences during the game in order to evaluate their own performance after playing the game.

Drappa and Ludewig [2] present SESAM, a simulation game where students assume the role of a project manager by hiring, firing, or designating tasks to employees. The game adopts a text-based interface where the student uses natural language to interact with employees, receiving replies in the form of statements. These statements are the only feedback available for the player to gauge his decisions during the game. At the end of the game session, SESAM displays the player's score, detailing the development statistics, such as the number of days to finish the project, human effort, cost, and requirements coverage. Previously hidden attributes of the customer requirements are also displayed to the player.

However, according to their evaluation over eighteen undergraduate students, students were making the same mistakes when replaying a session, thus the game had no apparent learning effect. The authors assumed that the cause was related to the score output, since students were not making a detailed analysis of the results. This was assumed by the authors because, during their evaluation, students were failing to reflect on the details of the game session and was doing a trial-and-error approach. When their final score was fairly good, they kept the same approach in the next simulation. Otherwise they tried a different approach.

Other digital games were proposed for teaching Software Engineering, such as MO-SEProcess and Groupthink [5] which are add-ons for the multiplayer online game Second Life. The first game, MO-SEProcess, is based on SimSE but focus on the waterfall approach where each player is a member of the development team. The second game, Groupthink, is also an add-on for Second Life and is based on a software specification exercise developed at MIT [10]. In this game, players form teams and answer questions related to software development. In both games, a final score is displayed at the end of the session to the teams, with no further feedback.

Finally, Pex4Fun [4], another digital game for teaching Software Engineering, focuses on code duels, where the player goal is to implement a puzzle method that follows a defined specifications and is equivalent to the hidden puzzle method. The only decision that needs to be made is related to the code that will be written since there is no interaction with other entities. Thus Pex4Fun is not compatible with our approach.

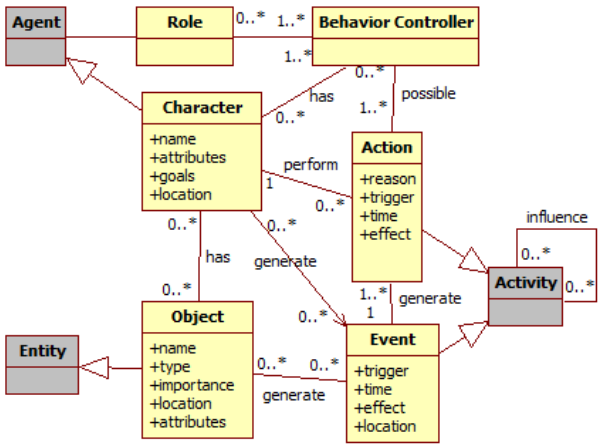

Figure 1: Mapping of provenance and game domains. Gray classes belong to the provenance domain. Yellow classes belong to the game domain.

III. USING PROVENANCE IN GAMES FOR ENHANCING SOFTWARE ENGINEERING LEARNING

A typical digital game architecture is mainly composed of game objects and the game loop. All objects present in a game, from environment objects to characters, are inherently defined as game objects. Game objects by themselves do not add characteristics to the game. Instead, they are containers that hold components that implement actual functionality, such as scripts (i.e., artificial intelligence, player controller, etc.), meshes (the object structure or "body"), physics, textures, animations, and audio. Meanwhile, the game loop is responsible for the sequence of events that occur in a game, allowing the game to keep running regardless of the user's input. The game loop keeps the game alive, updating game object states and executing their actions and behaviors. Each script in a game object has a function *update*, which is called by the game loop in order to execute the specific game object functionalities. Every time the game loop is ticked, it executes the *update* function of the scripts that belongs to the game objects present in the scene.

In a simulation or in a serious game some facts might not be clear or transparent enough for the player to understand why something went wrong. While in a traditional game this can be solved with a new game session, in a serious games or simulation it is important to give the opportunity to the player to find what caused this situation in an analytical way. Thus, in a previous work [6], we proposed a novel usage for provenance in the game field. In order to adopt provenance for the context of games, we mapped each type of vertices of a provenance graph into elements typically found in games.

The PROV provenance model assume that provenance of objects is represented by an annotated causality graph, which is a directed acyclic graph enriched with annotations. These annotations capture further information belonging to the system execution. According to Luc Moreau [11], a provenance graph is the record of a past or current execution, and not a description of something that could happen in the future. Using the PROV [12] notations, an *entity* was mapped to static game objects present in a game, such as weapons, equipment, and furniture. *Agents* were mapped to dynamic game objects, such as characters, event controllers, and plot triggers. Lastly, *activities* were mapped to actions or events executed throughout the game, such as interactions with other *agents* and *entities*. The causal relations, which are the edges of a provenance graph, were mapped to influences occurred during the game. Figure 1 illustrates this mapping of provenance concepts into the game context, outlining important information of each element type to be collected during game execution for provenance analysis.

The provenance analysis infrastructure, which uses the framework presented in [6], was instantiated in a software engineering educational game named SDM (Software Development Manager) [8]. The goal of SDM is to allow undergraduate students to understand the existing cause-effect relationships in the software development process. Thus, the adoption of provenance has the potential to better support knowledge acquisition, allowing tracking mistakes made during a game session or identifying concepts that are not well understood by the students.

A. SDM

In SDM the player manages a team of employees that develop software according to contracts made with customers. The gameplay and game mechanics are modeled presenting possibilities to the player to decide strategies for development and defines the roles and tasks for each staff member. As in any contract, the software has requirements that must be followed during development. From a gameplay point of view, these requirements help to balance the mechanics and rules. When the software is completed and delivered to the customer, there is a quality assessment of the software and a project completion payment accordingly to the product assessment. Since SDM focuses in people management, the main elements of the game are the employees, which represent the player's labor force. Employees can perform different roles (analyst, architect, manager, marketing, programmer, and tester), which use the employees' human attributes to calculate their performance depending on the respective roles. Another attribute present in the game is specialization, which is used to define the employee working competence. With the specialization system, it is possible for employees to undergo training to learn new sets of skills. Also the concepts of working hours, morale, and stamina are used to modify the employee's productivity.

These characteristics are illustrated in Figure 2, which shows a simplified version of SDM's class diagram focusing on the employee. Each employee is defined by his human attributes (adaptability, auto didacticism, human relations, logical reasoning, meticulousness, negotiation, objectivity, organization, and patience), can have specializations categorized in three different types (with a total of 14 different specializations), and can be allocated for training in order to acquire new specializations.

Each employee can have up to two different roles at the same time, among six possible roles available. Each role has a different set of tasks, which are administered by decisions

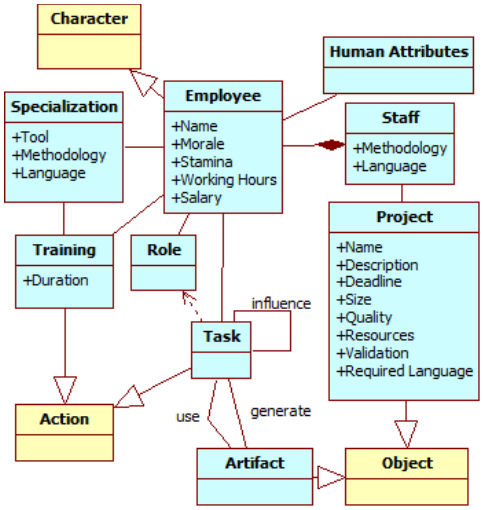

Figure 2: Mapping of game and Software Engineering domains. Yellow classes belong to the game domain, showed in Figure 1. Blue classes belong to SE domain (SDM game).

trees [13] that considers internal (attributes, morale, and stamina) and external (player or staff) influences to determine how these tasks are executed. Tasks can influence and be influenced by other tasks from another employee and can also generate artifacts, which can represent prototypes, used to validate software requirements, or test cases (unit, integration, system, and user acceptance). Lastly, employees belong to the player's staff and develop the software for a customer, respecting the customer's requirements and deadlines.

B. *Provenance Gathering in SDM*

The data structure used in SDM to collect provenance information was adapted and mapped to be suitable for the proposed provenance structure presented in [6], which is as follows: each project contains a list of employees involved in its development. In turn, each employee has a list of his actions executed throughout the development. If any action had an external influence during its execution, then the action also has a pointer to the action that influenced it. Throughout the game, the information about actions that are executed or triggered is collected at runtime and stored for later usage. Executed actions go to their respective employee lists. When new employees are added to the project, they receive their own list of actions and are added to the project's employee list. Each day of the game universe stores the state of the software development at the end of that day.

Since the information collected is used for the generation of the provenance graph, its content is mapped to one of the three possible types of provenance vertex: *activities*, *agents*, or *entities*. This mapping is made according to the data model explained in [6] and previously mentioned at the beginning of this section: *activities* map to actions or events, *entities* map to static game objects (prototypes, test cases, software development state), and *agents* map to dynamic game objects (employees and clients).

The majority of the provenance gathering, which is related to *activities*, is administrated by decisions trees and occurs at leaf nodes of the tree, where actions are executed. The

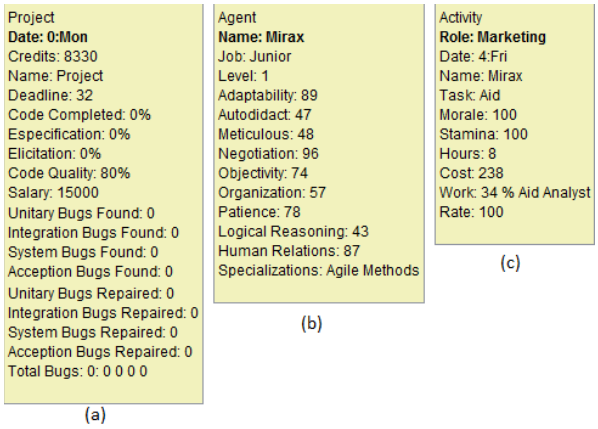

Figure 3: Provenance information regarding the project as a whole (a), an employee (b), and an action (c).

information gathered varies according to the element type, as can be seen in Figure 3. *Activities*' provenance information (c) is taken directly from the decision tree, getting the execution information and retracing the tree path from the leaf to the root. *Agents*' information (b) is gathered when they first interact in the game. *Entities*' information gathering varies according to the entity type. For example, the project as a whole (a) has its information gathered in a daily basis, recording the current state of development. On the other hand, prototypes and test cases *entities* have their provenance collected when they are created.

Moreover, the causal relationship between elements is also gathered. This occurs, for instance, when an *activity* is influenced by another *activity* or generates an influence to an *entity*. Examples of influences include an employee aiding another employee or when a task changes the state of the software under development.

C. *Provenance Visualization*

With the adaptations for provenance gathering made in the original SDM [6], it became possible to use the collected provenance data to generate a provenance graph for analysis. The collected game data, known as *game flux log*, is exported to *Prov Viewer* [9], which is a provenance graph visualization tool adapted for usage with SDM. In *Prov Viewer*, the game flux data is processed and automatically used to generate an interactive provenance graph of the game session to aid the analysis process.

Figure 4 illustrates the graphical user interface (GUI) of *Prov Viewer* and the displayed provenance graph from a gameplay session generated by SDM. Using the visual notations defined in [11], square vertex represents an *activity*, while circle represents an *entity* and an octagon represents an *agent*. The provenance graph is displayed at the center of the screen but only part of it is visible due to the graph size. However it is possible to zoom in or out and navigate through the graph. The graph layout is set to be similar to a spread sheet, where each "line" represents the *activities* of each *agent* and each "column" represents a day in the game. The filters, specifically defined for SDM, are located at the lower region of the interface. The "Collapse Agent" button collapses all the

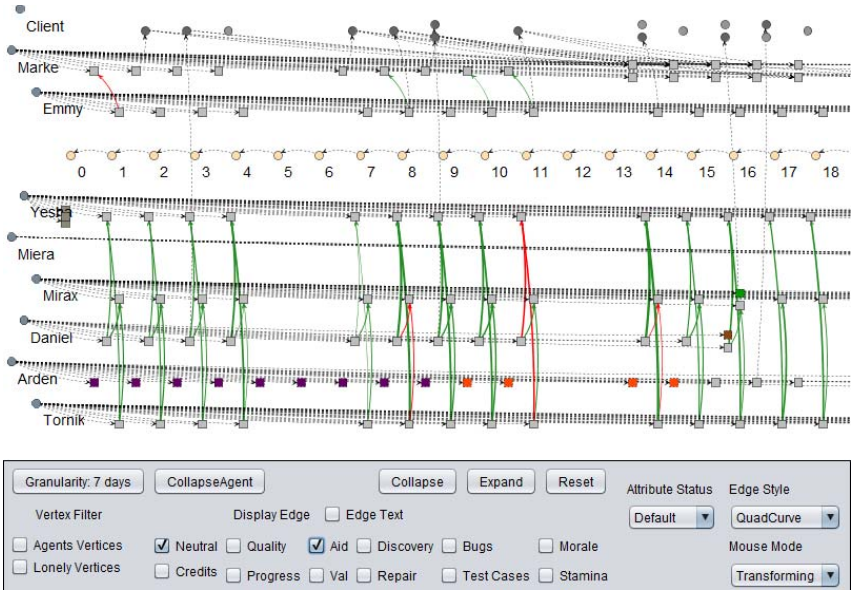

Figure 4: Prov Viewer's GUI instantiated for SDM.

agent's vertices into the *agent* itself. It is useful to detect if an *agent* had any influence throughout the game, instead of looking vertex by vertex. The "Collapse" button allows the user to collapse selected vertices, creating a meta-vertex that summarizes edges (influences) by type. The "Extend" button removes the last collapse made to generate the selected meta-vertex.

The "Display Edge" is an important aspect during analysis, allowing for the identification of types of influences in the graph, filtering the graph edges that are not relevant for the desired analysis. The displayed graph only shows the selected edges types, omitting unselected types. For example, in Figure 4 the edge types "Neutral" and "Aid" are selected, thus showing all positive (green) and negative (red) influences of the "Aid" type and all "Neutral" (dotted-black) type edges, which in this case are association edges.

Another usage of the display edge is to detect the reasons for drastic changes during the game. For example, detecting a major variation in the analyst's performance as shown in Figure 5, which dropped from *342* to *34 requirement validation*. The left picture has the "Val" edge display on, while the right picture has the "Aid" edge display on. The employees' roles in the figure are manager (upper tasks), marketing (Middle), and analyst (bottom). The change in

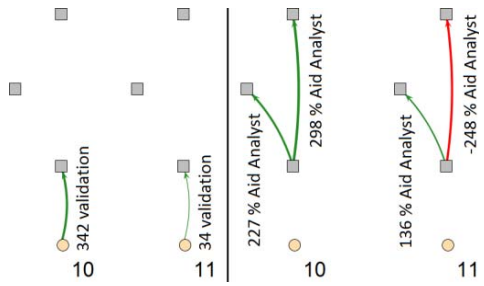

Figure 5: Analyzing the analyst's productivity.

performance was detected by activating the "Val" edge filter and comparing the values (342 versus 34). The reason for this sudden drop can be traced to the manager and marketing employees by changing the filter to "Aid", which is a possible type of influence. By analyzing the displayed edges, the manager employee provided an aid of 298% in day 10 and a penalty of 248% at day 11 to the analyst due to wrong decision making. Moreover, the marketing employee provided a bonus of 227% and 136%, respectively for days 10 and 11. By combining these factors, at day 10 the analyst received a bonus of 525% in his task, while at day 11 he had, in total, a penalty of 112% for the execution of his task. The analyst productivity without any bonus was 65 at day 10 and 53 at day 11, which is within his productivity margin.

The "Attribute Status" changes the vertex color according to their values from the selected attribute. In SDM they can be: Morale, Stamina, Hours (short for Working Hours), Weekend (highlighting "Saturday" and "Sunday" vertices), Credits, and Role. The vertex color does not change if it does not have the selected attribute. The default mode shows common activities with a shade of gray and uncommon activities with different colors. Common activities in SDM are normal tasks executed by employees during their roles, while uncommon activities are activities that do not happen frequently. The color difference amongst vertices is useful to quickly identity non-ordinary events. For example, by looking at the graph shown in Figure 4 it is possible to quickly identify that an employee was trained (purple vertices) during some days and was idle (red vertices) for a couple of days after the training was complete. This type of visualization, based on the evaluation of attributes, is useful to quickly identify particular sections in the graph.

With the available features, users can manipulate the graph by deciding which type of edges and color schemes will be displayed on the provenance graph. Furthermore, he can navigate in details through the graph, exploring different

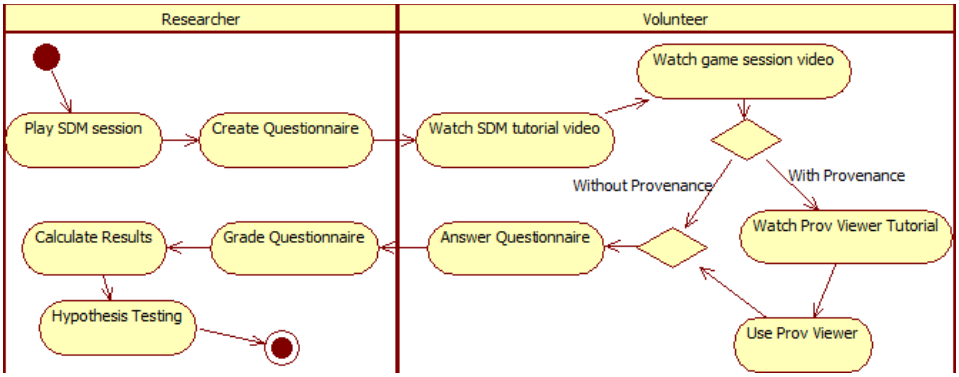

Figure 6: Experiment process.

sections of the game session or zoom out for a broader view. It is also possible to collapse sections of the graph in order to reduce its size and thus omitting vertices that might not be relevant for the current analysis. All graph manipulations can be reverted and no information is lost during this process.

IV. Evaluation

As discussed before, the goal of this paper is to answer the following research questions:

1. Does provenance analysis help to understand events that emerged during the game?
2. Is provenance analysis faster than only watching a replay of the game session?
3. Is provenance analysis more accurate than only watching a replay of the game session?

To assess the possibility of using provenance analysis for improving understanding, we generated a replay of a game session and compared it with provenance analysis using a provenance graph. This comparison was conducted through a questionnaire containing specific questions about events that occurred during the game session. Volunteers were divided into two groups: with and without provenance. Both groups watched the replay of the game session. The group with provenance also had access to the provenance graph. At the end, both groups answered the same questionnaire. Note, however, that we do not want to measure if watching a replay video is better or not than analyzing with provenance. Our goal is to verify if by giving access to provenance data about a game session, which in this case is the video, the usage of provenance is beneficial and if it aids in the understanding of the events.

Lastly, we used two metrics to compare the results obtained by both groups: precision and time. The precision metric shows the correctness of the answers provided by both groups, which is related to understanding the factors that influenced the results. The time metric shows how long each volunteer took to answer all questions, thus allowing us to know if using or not provenance is faster.

A. *Experiment Execution*

We originally planned to run the experiment with students playing the game, which in turn would result in different game scenarios. However, due to the nature of game dynamics and its randomness, the questionnaire would need to have general questions for all types of outcomes. So we opted for a more controlled environment in order to reduce the number of independent variables, which would be beyond our control. This way, instead of playing the game, volunteers were required to watch a recorded game session previously played by a third person. Thus, the questionnaire could be customized to this game session, allowing asking specific questions about events that occurred in that particular session. Also, the questionnaire was designed to measure the precision of the answers provided by both groups (with and without provenance) and the time volunteers took to finish it. Precision [14] is a traditional metric for information retrieval and can be seen as a measure of correctness, which is the percentage of results that are relevant. Time was measured in minutes taken to complete the questionnaire.

The execution was divided in two stages: a pilot experiment to detect any issues that needed to be addressed, and the experiment itself. During the pilot, volunteers were required to read and watch tutorials due to the unfamiliarity with the game and the *Prov Viewer* tool before filling the questionnaire. The pilot was applied to an undergraduate class composed of 28 volunteers and was structured as follows: volunteers were randomly divided into two groups and the pilot experiment began with volunteers watching the SDM tutorial, then the *Prov Viewer* tutorial (only for the group with provenance) and the replay of the game session video. Lastly, they received the questionnaire.

By analyzing the pilot execution and the obtained results, we decided to change the order of the videos due to the fact that volunteers were reviewing the *Prov Viewer* tutorial while answering the questionnaire. This happened because they were forgetting how to operate the tool after watching the replay of the game session video, which takes around seven minutes. Another change made for the experiment was related to the questions in the questionnaire. Some questions were allowing different interpretations, which caused too many mistakes on both groups. Thus, we decided to create a new scenario (and video) with a different set of questions. The collected data from the pilot was discarded during statistical analysis.

After changing the original experiment structure used during the pilot, the resulting experiment plan was divided in

three stages, as illustrated in Figure 6: Generating the questionnaire, running the experiment with volunteers (students), and analyzing the results. According to the plan shown in Figure 6, we executed the first stage (Create Questionnaire) before running the experiment. In this stage we recorded a game session of SDM that narrates the player's decisions throughout the game.

The next stage was the experiment execution with volunteers. We applied the experiment in two different undergraduate classes [15], composed of 18 and 19 volunteers each. From the total of 37 volunteers, only 32 finished the experiment, thus 5 partially answered questionnaires were discarded because volunteers decided to abandon the experiment. Figure 7 illustrates the volunteer's characteristics, where the majority has never heard about software engineering. The volunteers watched the SDM tutorial, which explains details about the game interface, and read a written document summarizing key features. Subsequently, they watched the game session video and were randomly selected for two groups: those that would use provenance and those that would not. After watching the game session video, the questionnaire was handled to the volunteers. However, the group with provenance also watched another tutorial video about the tool before receiving the questionnaire. This stage also has a time limit to avoid fatigue. The game session and its provenance graph are available at http://gems.ic.uff.br/ping. Lastly, we performed a statistical analysis over the results by means of hypothesis test in order to compare the obtained results with and without provenance.

An important factor for the design of the experiment concerns the definition of the significance level used during statistical analysis. We used a confidence interval of 95%, which translates to $\alpha = 0.05$, where α is the probability of rejecting the null hypothesis given that it is true (Type I error) [16]. The following subsections describe the game session used for the experiment and the questionnaire.

1) Game Session Scenario

We adopted the following scenario for recording the game session. The player had at his disposal four employees: Yesha, Tornik, Mirax, and Emmy. He first assigned roles for each employee. Yesha was assigned as the staff's manager and had the task of aiding analysts. Tornik was assigned as an analyst. Mirax was assigned as marketing, which aids analysts and provides cash income to the player by making deals. Lastly, Emmy was assigned as programmer. Then, the player asked Yesha to hire tree new employees: Arden, which was placed in training, Marke, an architect, and Daniel, an analyst that would work for 14 hours a day. After a two-week training, Arden was allocated to work as programmer.

Starting the third week in the game, the player had

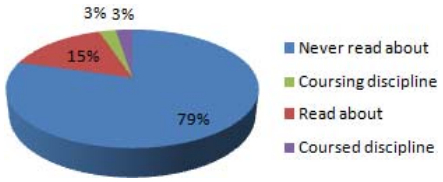

Figure 7: Volunteer's characterization results for SE knowledge.

financial problems and runs out of cash. Daniel, due to the extra hours, was tired and quitted. The game continued with some rearrangements in task: Tornik was assigned to do both elicitation and specification tasks and Arden started to work as programmer. Mirax was later promoted at the third week. During the fourth week, Marke's role was changed to programmer, focusing on repairing reported bugs, accumulating with the role of tester. Near the end of the week, Arden and Marke resigned due to lack of payments since the player was having financial problems. At the start of the next month, and after receiving cash for achieving a milestone, the player hired another employee (Miera) as a programmer to replace Arden. At the same week, the player set Mirax to negotiate with the client, asking to extend the project's deadline by one extra week, since the deadline was ending. Because of the deadline extension, the staff managed to finish the project, delivering the software to the client.

The software delivered still had one known unfixed bug, plus other 25 unknown bugs that were not identified during development but eventually showed up in production. Aside from the bugs, the code quality was with a rate of 75.84. This rate can vary from 10 to 120, where 10 represent the maximum negative modifier, 100 is neutral and does not generate a negative modifier, and above 100 is superb, providing a positive modifier. Thus, the value 75.84 is near the average (65.0). Concerning the player's financial status, the player started the game with 40,000 credits and at the end he had 5,969 credits and gained another 8,335 credits (out of 34,335) for delivering the software. The difference in payment is due to the number of bugs left in the software (26 bugs). Also, the player's reputation did not increase because of the poor quality of the delivered software (number of bugs). Concerning the staff, the player kept all starting employees, but lost three, out of four, hired employees. Three of the remaining employees lost morale during the development and one is fatigued. At the end of the session, the *game flux log* was generated by using the collected information from the game (employees, actions, and the project daily progression).

2) Questionnaire

The questionnaire was designed based on the video, consisting of ten questions. The first and the last questions are related to time measurements: the time when the volunteer started and finished the questionnaire. The second question was designed to identify the group of the volunteer: with provenance, which used *Prov Viewer* while answering the questionnaire, or without provenance, which answered the questionnaire only based on the game session video. The other seven questions are related to events that emerged during the game and have the same weight with values varying from 0 (wrong) to 1 (correct), depending on the provided answer. A value of 0.5 means the answer was partially correct, meaning that only one item was correctly identified. These questions explore different aspects of the game session, and some questions require a deeper knowledge of it.

The third question asks what made the employee Arden to quit. The forth question is equivalent to the third one, but related to the employee Daniel, since their reasons for quitting were different. Arden left because of lack of payment (morale

decreased due to lack of payment) while Daniel left due to overworking and lack of payment (morale decreased due to low stamina and lack of payment). Either answer was acceptable because we only asked for one reason. The fifth question asks why Tornik had made no progress during a certain period of time. The sixth question asks why Daniel's productivity had a sudden drop from one day to another. The seventh question asks the most contributing factor that allowed finishing the software in time. The eighth question asks the two most contributing factors that caused financial problems after day eleven. The ninth question asks which employee was idle for a period of time.

V. Statistical Analysis

A fundamental part of the statistical analysis of an experiment is the hypothesis test [17]. In the hypothesis test, two hypotheses are proposed and used to evaluate the collected data. However, hypothesis testing involves two types of error: Type-I and Type-II. The Type-I error refers to the rejection of the null hypothesis even when it is true, while the Type-II error refers to the acceptance of the null hypothesis when it is false. These errors depend on the power of the test, which is the probability 1 - β that the test is true if H_0 is false. Consequently, β is the probability of committing the Type-II error. There are parametric or non-parametric hypothesis test. Parametric tests have a greater power, thus produces more accurate and precise estimates. However, parametric tests can only be used if the samples follow a normal distribution. Nevertheless, non-parametric tests do not require normality and are recommended when samples are small [17].

The statistical analysis was performed with the intention of checking the obtained results and verifying if they have any significant difference. The main idea is to compare the results obtained from the questionnaire and the elapsed time of both groups. All tests were done using *R* [18], which is an open-source tool commonly used for statistical analysis.

First, we run a normality test, where the null hypothesis H_0 states that the collected data follows a normal distribution. The alternative hypothesis, H_1, states that the collected data does not follow a normal distribution. Given this, a normality analysis from the obtained data decides between using parametric or non-parametric tests. Thus, we used the Shapiro-Wilk test [19] with the following hypotheses:

$$H_0: Sample\ x_1, x_2, \dots, x_n have\ normal\ distribution$$
$$H_1: Sample\ x_1, \dots, x_n\ does\ not\ have\ normal\ distribution$$

The normality assumption was violated for all obtained results from the experiment with *p-value*< 0.01. It is possible to verify that *p-value* < α since α = 0.05, thus rejecting the null hypothesis. We also visually analyzed the data to consider using robust parametric tests, but the distribution differs significantly from a normal distribution. Therefore, non-parametric tests were adopted for statistical analysis. The non-parametric test used to compare the means was Mann-Whitney, which is also known as Wilcoxon rank-sum [20] test. Although there are other non-parametric tests, such as Chi-2 and Kruskal-Wallis, Mann-Whitney was chosen because it compares two means from two different samples against the same alternative hypothesis, which fits to our experiment design. The next subsection presents the results obtained from Mann-Whitney test to verify if the results, with and without provenance, differ.

1) Comparison of Means

We adopted the following hypothesis in our tests, naming *prov* as the group that used the tool and *replay* the group that did not:

$$H_0: \mu_{prov} = \mu_{replay}$$
$$H_1: \mu_{prov} \neq \mu_{replay}$$

The mean is calculated for each question from the questionnaire and for the duration that each volunteer took to finish it. Table 1 illustrates the mean and the standard deviation of each question for both methods, with green values representing the group with higher mean at each question from the questionnaire.

The *boxplots* shown in Figure 8 summarize the distributions of both approaches (with and without provenance). In these graphs, the boxes represent part of the central distribution, which contains 50% of data. Thus, the data scattering is proportional with the box's height. A black line inside the box represents the median. This way, 25% of data is between the box's edges and the median. The median location indicates if the distributions are symmetrical in the experiments. Lastly, circles indicate outliers. The *boxplots* for each question measures the correctness of the answer given by volunteers, while the last *boxplot* (duration) measures time, in minutes, for answering the entire questionnaire. We opted to only measure the total time due to the difficulty to control the time for each question for each volunteer in big groups of students.

It is possible to assert that there is a difference in mean if the null hypothesis is rejected. The null hypothesis is not rejected if *p-value* is greater than the significance level α. In other words, there is not enough evidence to assert a difference between results. When the null hypothesis is rejected (*p-value*< α), it is necessary to identify which method is superior by analyzing the confidence interval *CI*. If *CI* – α < 0, then $\mu_{prov} > \mu_{replay}$. Otherwise $\mu_{prov} < \mu_{replay}$. By analyzing the *p-values* from Table 2, the usage of provenance analysis provided better results in question 3 and in the time required to finish the questionnaire (duration), while there is not enough evidence to assert difference between results for the other questions (*p-value* > α). Even though both questions 3 and 4 asked about the reason that an employee quit the staff, only one volunteer that answered the questionnaire without provenance identified that the lack of payment was the reason for it.

By comparing the *boxplots* in Figure 8 and the statistical results, it is possible to infer that question 3 yielded better results by using provenance while questions 4 and 5 had equal results. Meanwhile, questions 7 and 8 results were similar but with varying scattering. Even though results are matching with Mann-Whitney test, question 9 has a different behavior due to the small difference from *p-value* to α (*p-value* = 0.07 against α = 0.05). By analyzing the *boxplot* for question 9, the results for using provenance are greater than without provenance.

Table 1: Mean and Standard Deviation for each question

		Q3	Q4	Q5	Q6	Q7	Q8	Q9	Duration
With Prov	Mean	0.5	0.9375	0.1875	0	0.375	0.1562	0.8125	23.1875
	Standard Deviation	0.5164	0.25	0.4031	0	0.5	0.3010	0.4031	4.2461
Without Prov	Mean	0.0625	0.875	0.1875	0	0.25	0.0938	0.5	28.9375
	Standard Deviation	0.25	0.3416	0.4031	0	0.4472	0.2015	0.5162	10.5797

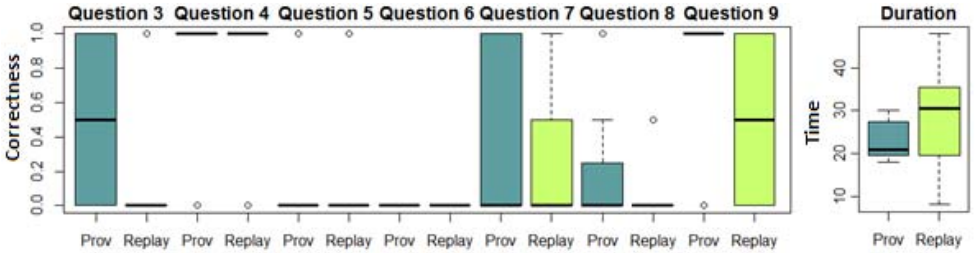

Figure 8: Boxplots from the experiment

Table 2: Results obtained from the Mann-Whitney test

α = 0.05	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Duration
p-value	0.007259	0.5757	1	1	0.467	0.6371	0.07049	0.03595

While without provenance's data is scattered around the maximum and minimum values with the median at the middle, the provenance's median is located at the maximum value.

Lastly, as shown by the Mann-Whitney test, using provenance for analysis provides faster answers than analyzing the game session's replay. This is clearly seen by comparing the medians between both methods and the box's scattering (height) position. The next section details existing threats to the validity of the experiment.

B. *Threats to Validity*

Despite the care in reducing the threats to the validity of the experiment, there are factors that can influence the results. In relation to internal validity, the selection of participants for both groups (with provenance and without provenance) can affect the results because of the natural variation in human performance. Furthermore, the experiment was executed with volunteers, which generally are more motivated for executing tasks. Anyone from the class could choose to be dismissed from the experiment and be released earlier. One possible threat is related to each individual perception of the events that occurred during the video. Lastly, the experiment was the first contact of the volunteers with both the game mechanics (by watching the video) and the tool. Thus, the lack of experience can affect the results, even when minimized by the usage of tutorials. Regarding external validity, we mitigated the discrepancy in experience level by selecting participants from two different classes of the same discipline (Introduction to Computer Programming), which occurs in the first period of undergraduate course in Computer Science at *Universidade Federal Fluminense*.

Regarding construct validity, the questionnaires were composed of several questions to reduce threats related to a lack of knowledge from the game, thus exploring different aspects from it. Another risk is related to people being afraid of being evaluated, thus trying to "look better" by lying. This is the case of how long they took to finish answering the questionnaire. To minimize this, we stated the exact time they began answering the questionnaire and verified the time they finished and delivered the questionnaire.

A threat related to conclusion validity is the reliability of measures. This is dependent on factors like question wording, which may allow for different interpretations, and the graph layout. To minimize the threat, we answered any doubts voiced by volunteers related to the questions or regarding the tool (*Prov Viewer*). It is important to notice that volunteers examined a video of the gameplay session instead of playing it to allow us better control over independent variables. However, in a real situation, they would play the game then proceed to the game flux analysis with provenance.

VI. CONCLUSION

This paper introduces new perspectives on software engineering learning process, leveraging the current state of the art, based on game sessions, to a level where the game provenance can induce deeper analysis and discussions regarding the game session. This knowledge can help on (1) confirming the hypotheses formulated by students, (2) supporting tutors for a better guidance, (3) motivating practical exercises around some case studies, and (4) extracting behavior patterns from individual sessions or groups of sessions.

The provenance graph aids in the understanding of the concepts taught by the game by making explicit the cause-and-effect relationships between entities and actions. The provenance visualization allows the discovery of issues that contributed to specific game fluxes and results achieved throughout the game session. This analysis can be used to improve understanding of the game flux and identifying actions that influenced the outcome [21], aiding the student to understand why they happened the way they did. It can also be used by the tutor to analyze a game session to verify the student's progress by checking his decisions and their consequences in the outcome, identifying concepts that might

not be clear to the student. For the research questions 1 and 3, the results from the experiment indicates in at least one case that analyzing the game session with provenance is beneficial and provides equal or more correct answers than analyzing the game without access to provenance data. It also aids in understanding the underlying influences between events and their effects. In relation to correctly identifying the causes of the events in the game, using provenance provided better statistical results in at least one case (question 3, related to lack of payment), and slightly better results in another (question 9, related to identifying the idle employee). The other cases were not statistically different with the current sample size even when their mean values were greater when analyzing the game session with provenance. Meanwhile, for research question 2, the results clearly show that analyzing the game flux with provenance is faster than analyzing without having access to provenance data, even when using the visualization tool for the first time.

Currently, we do not make automatic inferences from the provenance graph, but let the user decide what he wants to infer. Studies in this area are being made in order to identify information that can be omitted from the user without affecting the overall analysis. Another interesting research is to automatically identify patterns in the game flux and points of interest for the student and tutor. For future work, we plan to work on different graph visualization layouts and introduce the provenance support in other education games. We also plan to run more experimental studies, with players playing the game instead of watching a video, on the usage of provenance in educational games to evaluate the aspects of learnability. We also believe that the ideas discussed in this paper can open a wide range of research in the field of behavior patterns data mining of learning sessions.

ACKNOWLEDGMENT

We would like to thank CNPq, FAPERJ, and CAPES for the financial support.

REFERENCES

[1] A. Dantas, M. Barros, and C. Werner, "A Simulation-Based Game for Project Management Experiential Learning," *Softw. Eng. Knowl. Eng. SEKE*, vol. 19, p. 24, 2004.

[2] A. Drappa and J. Ludewig, "Simulation in software engineering training," *Int. Conf. Softw. Eng. ICSE*, pp. 199–208, 2000.

[3] E. O. Navarro and A. van der Hoek, "SimSE: an educational simulation game for teaching the Software engineering process," *Innov. Technol. Comput. Sci. Educ. ITiCSE*, vol. 36, no. 3, pp. 233–233, 2004.

[4] N. Tillmann, J. De Halleux, T. Xie, S. Gulwani, and J. Bishop, "Teaching and learning programming and software engineering via interactive gaming," *Int. Conf. Softw. Eng. ICSE*, pp. 1117–1126, 2013.

[5] E. Ye, C. Liu, and J. A. Polack-Wahl, "Enhancing software engineering education using teaching aids in 3-D online virtual worlds," *Front. Educ. FIE*, pp. T1E–8–T1E–13, 2007.

[6] T. Kohwalter, E. Clua, and L. Murta, "Provenance in Games," *Braz. Symp. Games Digit. Entertain. SBGAMES*, 2012.

[7] PREMIS Working Group, "Data Dictionary for Preservation Metadata," Implementation Strategies (PREMIS), OCLC Online Computer Library Center & Research Libraries Group, Final report, 2005.

[8] T. Kohwalter, E. Clua, and L. Murta, "SDM – An Educational Game for Software Engineering," *Braz. Symp. Games Digit. Entertain. SBGAMES*, pp. 222–231, 2011.

[9] T. Kohwalter, E. Clua, and L. Murta, "Game Flux Analysis with Provenance," *Adv. Comput. Entertain. ACE*, 2013.

[10] M. D. Ernst, "The groupthink specification exercise," *Int. Conf. Softw. Eng. Educ. ICSE*, pp. 89–107, 2006.

[11] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. V. den Bussche, "The Open Provenance Model core specification (v1.1)," *Future Gener. Comput. Syst.*, vol. 27, no. 6, pp. 743–756, 2007.

[12] Y. Gil and S. Miles, "PROV Model Primer," 2010. [Online]. Available: http://www.w3.org/TR/prov-primer/. [Accessed: 21-Mar-2013].

[13] B. Moret, "Decision Trees and Diagrams," *ACM Comput. Surv. CSUR*, vol. 14, no. 4, pp. 593–623, 1982.

[14] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[15] M. Svahnberg, A. Aurum, and C. Wohlin, "Using students as subjects - an empirical evaluation," *Empir. Softw. Eng. Meas. ESEM*, pp. 288–290, 2008.

[16] G. R. Norman and D. L. Streiner, *Biostatistics: The Bare Essentials*, 3 Pap/Cdr edition. Shelton, Conn: People's Medical Publishing House, 2012.

[17] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

[18] "R." [Online]. Available: http://www.r-project.org/. [Accessed: 26-Mar-2013].

[19] S. S. Shapiro and M. B. Wilk, "An Analysis of Variance Test for Normality (Complete Samples)," *Biometrika*, vol. 52, no. 3/4, p. 591, 1965.

[20] "R Documentation: Wilcoxon Rank Sum and Signed Rank Tests." [Online]. Available: http://stat.ethz.ch/R-manual/R-patched/library/stats/html/wilcox.test.html. [Accessed: 26-Mar-2013].

[21] C. Werner, R. Cepeda, M. Schots, and L. Murta, "How Design Style Relates to the Representational Power of Design Outcomes," *NSF-Spons. Workshop Stud. Prof. Softw. Des.*, 2010.