# – Implementation

## Introduction

In this work, the provenance in games framework is instantiated in the SDM game (KOHWALTER; CLUA; MURTA, 2011) as a proof of concept. The SDM game focuses on introducing Software Engineering concepts and skills to undergraduate students. The new and improved version of SDM presented in this paper includes provenance gathering and analysis, allowing students to visualize their actions and identify steps that lead to successful or unsuccessful outcomes. While the main application of provenance in this paper is over a serious game, we believe that the concepts discussed in this paper are applicable to other kinds of games and useful to support advanced analysis, such as gameplay balancing, events and behaviors data mining, and even storytelling enhancements.

The proposed framework was instantiated in a Software Engineering educational game named *Software Development Manager* (SDM) (KOHWALTER; CLUA; MURTA, 2011). The goal of SDM is to allow undergraduate students to understand the existing cause-effect relationships in software development. As so, the adoption of provenance becomes an important instrument to better support knowledge acquisition, allowing the possibility of tracking mistakes made during a game session.

## SDM

In SDM, which was developed using the game engine Unity3D (HIGGINS, 2010), the player has a team of employees that are used to 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 define the roles 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 quality.

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 uses the employee’s attributes to calculate his performance depending on the role performed. Another element present in the game is specialization, 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. Figure 1 shows a simplified version of SDM’s class diagram focusing on the employee, showing his human attributes, types of specializations, the possibility of training to acquire specializations, and that the employee is affect by other employees that belong to the staff team. It also illustrates the project, its characteristics and requirement.

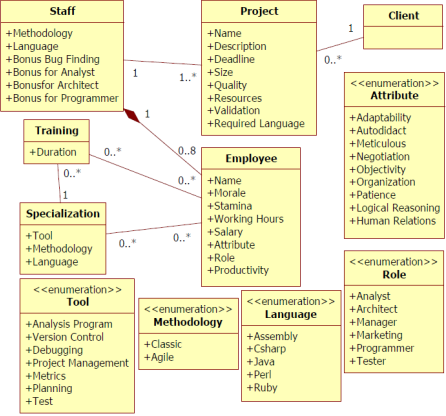


Figure 1: SDM simplified class diagram

### Provenance Gathering

The information structure used in SDM is similar to the one explained at chapter 4. Each project contains a list of the 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 that action will also have a pointer to the action that influenced it. Throughout the game, when actions are executed or new employees are hired, the information of the event is collected and stored for later usage. Actions will go to their respective lists while new employees are added to the project list, creating their own list of actions. Each day passed in the game also records the state of the software development.

Since the information collected is used for the generation of the provenance graph, it contents are mapped to the three possible types of provenance vertex (*activities*, *agents*, and *entities*). This mapping is done according to the data model explained at chapter 4. In the following paragraphs is described the information details extracted from the game and their respective roles in the provenance graph.

Each action executed during the game is represented by an *activity* vertex. The information collected during its execution includes: who executed it, which task and role the employee was occupying, and the current morale and stamina status of the employee that executed the action. The worked hours in the day the action was generated and credits spent to execute the action are also stored. Lastly, the progress made in his task, if any. Besides those, if the action had any external influences, used or altered an artifact (prototypes or test cases), it is stored a link to the action or artifact that affected its execution.

The employees that participated in the development of the software are mapped to *agent* vertex in the provenance graph. The information collected includes the employee’s name, his current staff grade (junior, mid-level, or senior), his level, attributes, and specializations. Lastly, the *entity* vertex in the provenance graph represents one of the three possible artifacts in SDM: Prototypes produced by architects and consumed by analysts; Test Cases produced by analysts, architects, and programmers and consumed by testers; and Project, which represents the instances of the software development progress recorded each day.

The daily project information collected includes the day of its instance, the project’s deadline, how much coding was produced and the code overall quality. It also stores the clients requirements identified and modeled by analysts, how many credits the player had by the end of that day, and the state of each type of bugs in the software. For prototypes and test cases only the day they were created and their names are stored, since actions will contain the information of when they were used. Figure 2 illustrates the information collected in SDM and shown in Proof Viewer according to the vertex’s type.

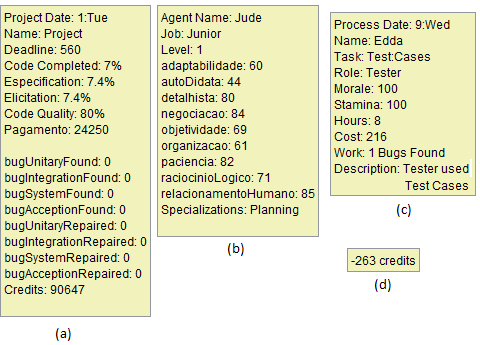


Figure 2: Information data in the provenance graph.

The *entity* vertex representation for the project’s data (a), the employee’s *agent* vertex data (b), and the action’s *activity* vertex data (c).

At the end of a gaming session, the collected data during the game session is exported to an external visualization and analysis tool, the *Proof Viewer*, for provenance analysis. *Proof Viewer* in turn analyzes the data and generates the corresponding provenance graph that represents the game session. The next section describes a game scenario in SDM that is used as an example for describing the *Proof Viewer*.

## Guiding Example

Use experiment 02 as an example.

## Provenance Analysis (Proof Viewer)

How the data is processed in Proof Viewer. Say that it does not uses xml, but there are plans for adapting it to accept xml input format. Developed in JUNG.

### Representations

How the data is represented in Proof Viewer (vertices and edges). Also, how collapsing works.

### Filters

How the filters are used in Proof Viewer.

## Final Considerations

# References

HIGGINS, T. *Unity - 3D Game Engine*. Available: <http://unity3d.com/>. Accessed: 5 maio 2011.

KOHWALTER, Troy; CLUA, Esteban; MURTA, Leonardo. SDM – An Educational Game for Software Engineering. In: 2011 X BRAZILIAN SYMPOSIUM ON GAMES AND DIGITAL ENTERTAINMENT (SBGAMES), nov. 2011, Salvador. *Anais*... Salvador: In: X SBGames, nov. 2011.