A Decision Support System for Game-Based Simulative Environment of Software Development Project Management

Ali Tizkar Sadabadi, *Member, IACSIT*

***Abstract*— Simulations are today very common and are used frequently for education, testing, research, development, gaming etc. With a simulation and a decision support system, certain factors are manipulated depending on what event is being emulated. This report is about how to proceed to develop an environment with the basis of an expert system, which helps project managers to develop a software project. By developing a simulation model, based on a specific software development model, that model could be implemented in a project game, this to give the player the possibility to control and steer certain events in the project and to vary the result of the game.**

***Index Terms*—DSS, expert systems, simulation, software development process.**

I. Introduction

On the previous published paper we relied on just a simulation environment for developing a software project. In this paper we will make a combination of that environment with a decision support system basis to develop not just a simple game but also an assistant for software project manager for getting feedback of their decisions which made in this virtual environment.

An important factor is that it provides insights into complex process behavior. Like many processes, software processes can contain multiple feedback loops, such as associated with correction of defects in design or code. Delays resulting from these effects may range from minutes to years. The complexity resulting from these effects and their interactions makes it almost impossible for human (mental) analysis to predict the consequences. Unfortunately, traditional process analysis does not shed much light on these behavioral issues, and the usual way to resolve them is to run the actual process and observe the consequences. This can be a very costly way to perform process improvement.

While the software industry has had remarkable success in developing software that is of an increasing scale and complexity, it has also experienced a steady and significant stream of failures.

On this paper we develop an environment for software project management as a real environment for project managers to practice and estimate the result of their decisions.

Manuscrip received March 6, 2012, revised March 25, 2012.

Ali Tizkar Sadabadi is with UTM (University Teknologi Malaysia) AIS (advanced Infromatics School).University, Malaysia (email: tsali2@live.utm.my; al_tz2@yahoo.com.)

II. Objectives

This simulation environment must make tradeoffs among faithfulness to reality, level of detail, usability, teaching objectives, and "fun factors". As an example, educational purpose and fun factors may call for visual feedback to be humorous and "over the top", such that the causal effects of decisions can be easily recognized by managers. However, this would contradict the issue of faithfulness to reality. Clearly a delicate balance has to be struck among all design parameters. Beside that we want to add and knowledge-base and an inference engine to the system to be able to learn from previous inputs, mistakes or successes. For this case we decided to choose a case-base system as the environment is such that every recurrence of the application is based on its own. To guide us in creating this balance, we formulated a set of overarching guidelines for the design of Simulation environment, based on both the unique nature of software engineering and lessons learned from previous studies about critical success factors for educational simulations:

- Simulation environment should provide a manager with clear feedback concerning their decisions. Consider, for example, a manager who hires a plethora of employees in order to rush code development. Later, the manager faces a lengthy phase with many bugs being discovered. The simulation environment must explain why this situation occurs (e.g., more employees lead to more parallel work, which leads to more integration problems [1]).
- Simulation environment should be easy to learn, enjoyable, and comparatively quick. Without compromising the level of complexity that is needed to make each simulation unique, a single simulation must neither be too difficult nor too long in order to maintain the interest of students and, thus, its value as an educational tool. The enjoyability of the game is a large part of what will make the lessons learned more memorable [2].
- Simulation environment should illustrate both specific lessons and overarching practices of the software process. Specific lessons include, among others, Brooks' Law (adding more employees to a project that is already late will make the project even later, due to the exponentially increased communication overhead [3]) and the fact that the use of a configuration management system normally improves the development process. Overarching practices concern the choice of different life cycle

models, the tradeoffs involved in many decisions, and the potential for drastic consequences in case of failure.

- A Case-memory which would be the archive of previous inputs. These previous inputs are the results of previous plays of the program which are to be stored in a specific way in a database. Afterwards the player wants to start over, the program would call a set of routines (Case-based reasoning engine) to retrieve a structured knowledge formed recommendations for player. Thus the program needs to be played several times to have a strong knowledge-base for its reasoning engine to feed. At the start when the knowledge-base is empty without previous experiences the program applies a set of self-recommendations or in better way of saying, an instinct or some approximations (a knowledge-base) from the software engineering methodologies defined rules. That's obvious this recommendations are not practical and applicable in a real world industry. So the importance of the Case-based reasoning engine and a knowledge-base is here becomes illuminated and glamorous. The basic idea is like the human growing from childhood to adulthood. This is the way we learn from our environment and practice our skills and apply our perception in the way of series of actions. If they are true we will get positive feedback and if they are false or just a mirage we will get some desperate results.
-

III. Approach

Simulation environment is a single-player game in which the player takes on the role of project manager of a team of developers. The player must manage these employees to complete a particular (aspect of a) software engineering project. Management activities include, among others, hiring and firing, assigning tasks, monitoring progress. In general, following good software engineering practices will lead to positive results while blithely ignoring these practices will lead to miserable failure in completing the project. The difference between previous versions is in this program we provide a wide range of suggestions for the player to get better results from past acts. Thus that means the player needs to go on with the playing environment and let the systems learn and gather systematic and helpful information for the player.

IV. Architecture

A. Simulation Models and Software Methodologies

There are different kinds of simulators, continuous- and discrete simulation models. The continuous model can, strictly speaking, only be applied on an analog computer. The closest, someone can get to a continuous model, is by making the time steps in a discrete model small enough so it will be visualized as a continuous time flow. Fig. 1 shows the simulation model we used for the simulation environment.

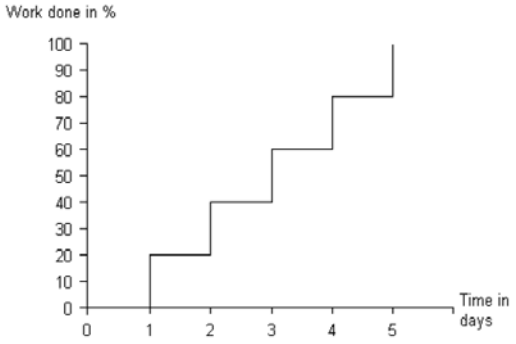

Fig. 1. Discrete simulation model

A particular model was purposed to emulate a waterfall process, we developed the model to reward the player for following the proper steps and practices of the waterfall model and penalize them for doing otherwise. In parallel, we aimed to teach a number of overall lessons about the software engineering process in general. These lessons were taken from a compendium of 86 "fundamental rules of software engineering" [4] that we gathered by surveying software engineering literature. The followings are a few examples of the lessons that are implemented in our model:

- Do requirements, followed by design, followed by implementation, followed by testing.
- At the end of each phase, perform quality assurance activities (e.g., reviews, inspections), followed by correction of any discovered errors.
- Developers' productivity varies greatly depending on their individual moods, and matching the tasks to the skills and motivation of the people available increases productivity.
- The greater the number of developers working on a task simultaneously, the faster that task is finished, but more overall effort is required due to the growing need for communication among developers as in [1]. So this statement is applied to the game based on fast money expenditure and negative mood effects that they will get tired form increased communication channels required.
- Software inspections are more effective the earlier they are performed [5]. The review proceedings are available before the start and also within the game.
- Monetary incentives increase motivation, which leads to increased productivity (but faster expenditures) as in [5].

In addition to these, there are a number of other general workplace issues not specific to software engineering that are included in the model. For instance, employees sometimes get sick, take breaks when they are tired, become less productive when they are tired, and quit when they are upset about something significant (e.g., a pay cut).

B. Expert System and Decision Assistant

The second part of the program consists of its expert system which analyzes the previous cases (results) of the playing and put into the database and afterwards use for some suggestions. The schematic case-base engine would be Fig. 2.

For the reasoning algorithms (retrieving the results) we applied Nearest Neighbor Algorithm for Case Retrieval as following steps:

1. Receive new case as an input to the algorithm.

2. Prepare the data structure to hold case and its associated similarity value with the new case. Set maximum similarity to one and minimum to zero.
3. Begin retrieving all cases from the database one by one serially.
4. For each case retrieved from database, begin:
   i. Set a numeric variable total similarity to 0.
   ii. For each attribute in the attribute list of the cases, loop2
   - If (attribute X of new case is not 'null' and attribute X of retrieved case is not 'null')
   - Then increase the total similarity by $sim(f_X^{NC}, f_X^{RC})$

   iii. Store the cases and their total similarities in the data structure defined in Step 3.
5. Compare the total similarity values of all the cases and arrange them in ascending order.
6. Prepare array of cases with minimum number of similar cases required as input.
7. Return the array as the output of the algorithm.

V. Simulation Environment

The simulation environment is responsible for executing a simulation and creating the user experiences that demonstrate the inherent complexity of the software process. The simulation loop itself operates in the following manner (refer to Fig. 1). The clock drives the simulation by emerging the simulation events at equal time intervals. At every clock tick, the execution component checks which events are to be happened by querying the state management component. It then emerges the respective dialog windows to communicate with the user, and in turn propagates the effects of these dialogs on the environment's variables and the project factors. After this, the user interface is updated and the progressbar is gone forwards that represents the project's development. The results are shown after a phase is completed and the current state of the project simulation and the artifacts are visible through the respective monitors on the right part of the screen. Fig. 2, Simulation's Graphical Environment.

The upper left part displays a virtual office in which the software engineering process is taking place, including typical office surroundings (e.g., desks, chairs, computers, meeting rooms) and employees. Employees "communicate" with the manager (player) through pop-up "bubbles" that appears over their heads. The player can interact with the employees through right click menus (not shown) on each employee that displays a list of available actions (e.g., employ new persons, give a rest, fire, etc.).

The player can control the simulation by stopping or starting again or player could start over a new game. The simulation environment gets the initials values of the project by selecting the project menu and clicking on cocomo and filing the requested fields. With the beginning of the simulation, it predicts the project's progress and values and prompts for failure if it is expected. Even so the player is still able to continue with playing with the current parameters and see the final results.

At the end, the player clicks on the results' button to see the project's results. If the project is led well with high probability it is going to be successful, otherwise the project fails.

Fig. 2. Simulation's Graphical Environment.

VI. Case-Based Reasoning

A general CBR cycle may be described by the four basic processes [6]. An initial description of a problem defines a new case. This new case is used to retrieve a case from the collection of previous cases. The retrieved case is combined with the new case - through reuse - into a solved case. Through the revise process this solution is tested for success, e.g. by being applied to the real world environment or evaluated by a project manager, and repaired if failed. During retain, useful experience is retained for future reuse [7]. The cycle is shown in Fig. 3.

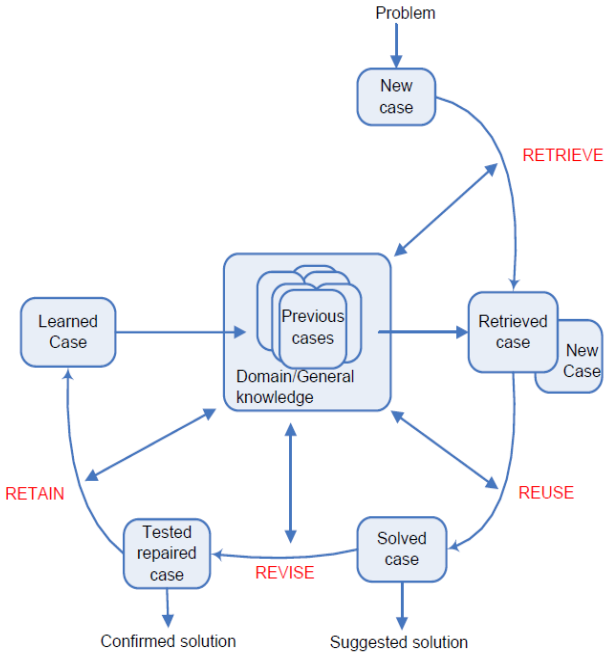

Fig. 3. Case-based Reasoning Life Cycle [8]

The final system for transacting with the player is a recommendation system that has two dimensions: with the player or the interface and with the simulation environment and its results. The interface is something that player would

see and interacts with that. Other parts of this system are hidden from the player and its transactions and procedures (inference engine) are working on the background. Fig. 4 shows the diagram of this system.

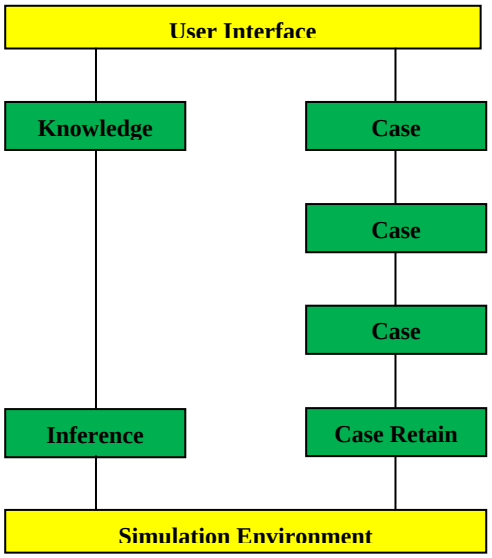

Fig. 4. Shows the diagram of hybrid system

As seen above the diagrams has a little difference from exact case-based reasoning. That is so called Hybrid system [9]. The architectural goal of a hybrid expert system is to improve start of the game when there is nothing in the case memory, this systems strives by augmenting it with case-based reasoning. This augmentation is done by taking the rules as a starting point of problem-solving, i.e. using rules to generate a first approximation to the project and then invoking case-based reasoning to handle exceptions to the rules. The idea is to fine-tune the performance of the rules. Another advantage is that if the case-based reasoning misses a similar case, the architecture will at least have a reasonable default answer generated by the rule-based system.

VII. Related Works

Software engineering educators have invented new and innovative ways of teaching the subject in recent years. Some of these approaches have included software process simulation designed specifically for education. One of the most advanced of these is SESAM, a software engineering simulation environment in which students manage a team of virtual employees to complete a virtual project on schedule, within budget, and at or above the required level of quality. The student drives the simulation by typing in textual commands which can consist of hiring and firing employees, assigning them tasks, and asking them about their progress and the state of the project as in [6]. Although SimSE and SESAM share the purpose of teaching students the software engineering process in a game-based setting, SESAM, unlike SimSE, lacks a visually interesting graphical user interface, which is considered essential to any successful educational simulation as in [2]. Moreover, despite the fact that the SESAM language is highly flexible and expressive, the model building process is learning- and labor-intensive and requires writing code in a text editor. Despite these drawbacks, SESAM's models do provide a source of some well-documented software engineering rules of behavior that we plan to leverage in our simulation environment.

The other is OSS's environment which involves case studies which complement the standard materials. The case study element of the course explores various aspects of software engineering and is presented through an innovative interactive multimedia simulation of a software house Open Software Solutions (OSS) [10].

Fig. 5,6,7,8 show OSS's Screen layout and the reception showing the lift door and the OSS organizational chart and The library on the fifth floor contains several books, journals and manuals related to the course and Meeting with a CIRCE project team, showing the workbook being used for interviewing.

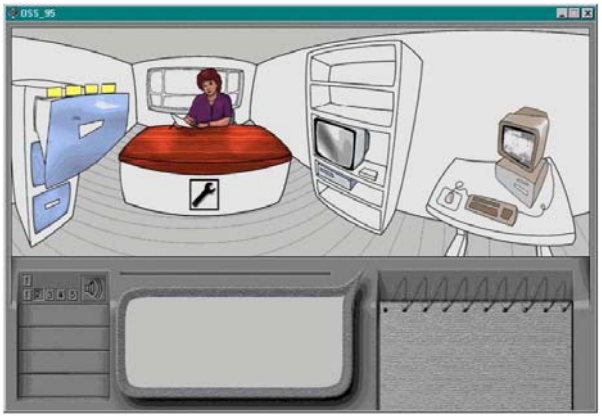

Fig. 5. The overall screen layout is shown in Figure

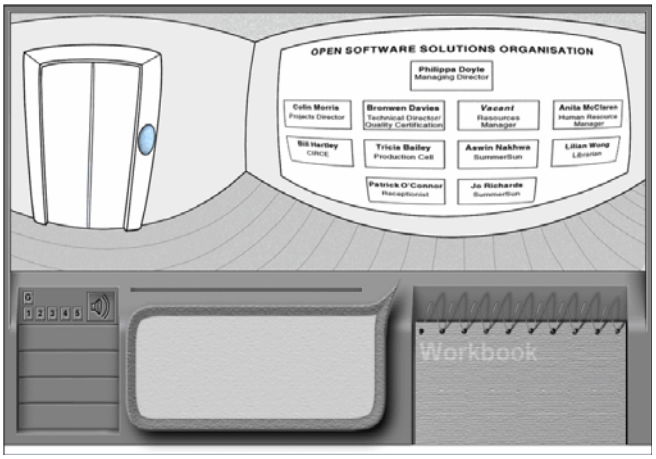

Fig. 6. Part of the reception showing the lift door and the OSS organizational chart

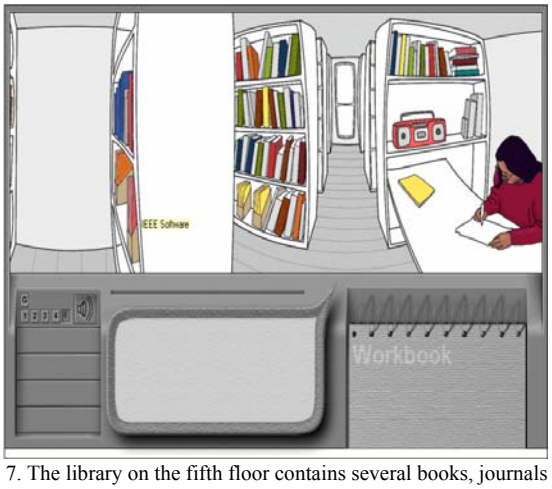

Fig. 7. The library on the fifth floor contains several books, journals and manuals related to the course

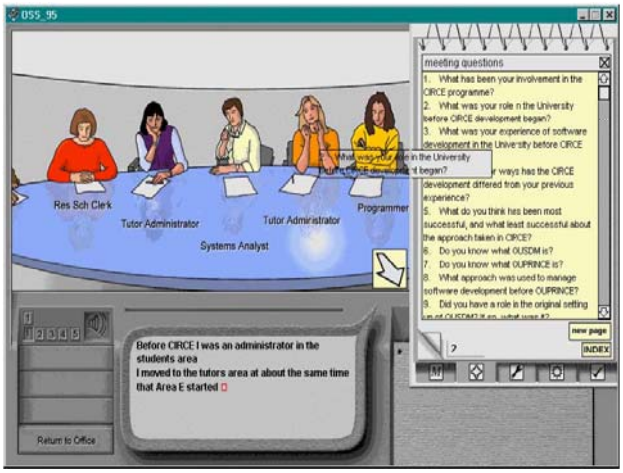

Fig. 8. Meeting with a CIRCE project team, showing the workbook being used for interviewing

We presented previous works as only a simulation environment for software engineering of software development process education. In this paper we discussed about the feasibility of developing a system for software project manager to develop their skills with combination of simulation technique and the power of expert systems to learn and give some useful information for our specific work.

VIII. Summary

We have presented an intelligent simulation environment, an interactive, graphical, customizable simulation game for software project managers. This attempts to address the lack of process realization present in the typical software engineering environments by providing managers with a practical, high-level experience of a realistic software engineering process in an engaging manner.

IX. Conclusion

The system and environment presented as above is a combination of simulation, game technique and expert systems capabilities to develop and provide software manager a proper environment for practicing his skills and ability to manage medium or even colossal projects. This is what to begin with for a manager to present his abilities. The previous works as we mentioned before has the lack of a decision support subsystem for this kind of emulation to advice player to get in the future the better results from past failures or undesired results.

The constructed system exploited computer as an intelligent and deductive instrument. The recommendation system assists the software development process by reminding the theories required and storing the cases of the project results. Results show that it can be used to improve project, separate practice from memorization and encourage different personalities in management. Of the two reasoning techniques, rule-based reasoning and case-based reasoning, implemented in the system, the later one (case-based reasoning) was found to be more effective for software project management although each had its own pros and cons. To summarize, rule-based reasoning exploited theories (defaults by getting approximation of cocomo formulas) and case-based reasoning exploited experiences have been taken before.

For the readers we advise to read the previous version of this project which was just concentrated on simulation and game technique. But after that they will see the difference of these two versions and the advantages of this work.

Acknowledgment

The initial idea of commencement of this project was from my master's supervisor Dr. Suren Khachatryan that encouraged me for this matter instead of other cliché proposals. And also I would like to give my best appreciations to Dr. Tabatabaei who guided and advised me in the way of publication.

References

[1] D. E. Perry, H.P. Siy, and L.G. Votta, "Parallel Changes in Large-Scale Software Development: An Observational Case Study," *ACM Trans. Software Eng. and Methodology*, vol. 10, no. 3, pp. 308-337, October 2001.

[2] M. Ferrari, , R. Taylor, and K. VanLehn, "Adapting Work Simulations for Schools," *The Journal of Educational Computing Research*, vol.21 , no.1, pp. 25-53, 1999.

[3] F. P.Brooks, *The mythical man-month: essays on software engineering*, 2 ed. Boston, MA: Addison-Wesley, 1995, ch. 2.

[4] E. O.Navarro. (March 2005) "The Fundamental Rules of Software Engineering," *SimSE, University of California, Irvine*, [Online] Available : http://www.ics.uci.edu/~emilyo/SimSE/se_rules.html.

[5] J. D. Tvedt, "An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time," Ph.D. Dissertation, Department of Computer Science and Engineering, Arizona State University, Tempe, AZ, USA, 1996.

[6] K. D. Althoff, R. Bergmann, M. Minor, A. Hanft, "Advances in Case-Based Reasoning", in *Proc. The 9th European Conference, ECCBR 2008*, Trier, Germany, September 2008, pp. 44-58.

[7] S. K. Pal and S. Shiu, *Foundations of Soft Case-Based Reasoning*, Wiley Series on Intelligent Systems, 2004, ch. 2.

[8] P.C Chang, C.Y Lai, "A hybrid system combining self-organizing maps with case-based reasoning in wholesaler's new-release book forecasting," *Expert Systems with Applications*, Volume 29, Issue 1, Pages 183–192, July 2005.

[9] R.T.H. Chi, M.Y. Kiang , *An Integrated Approach of rule-based and case-based reasoning for decision support*; Austin, Texas, ACM, 1991, pp. 258.

[10] H. Sharp, and P. Hall, "An Interactive Multimedia Software House Simulation for Postgraduate Software Engineers", in *Proc. the 22nd International Conference on Software Engineering*. New York, USA, 2000, pp. 688-691.

Ali Tizkar Sadabadi was born in Tabriz, Iran. He has got the master degree in the field of Informatics and Computer Systems. He is PhD student at advanced Informatics School, UTM AIS, Kuala Lumpur, Malaysia. His academic background is: 2000-2006 B. Eng (Bachelor of Engineering) from Azad University branch Tabriz, Iran. His major field of study is computer engineering, Software. 2006-2008 M. Eng (Master of Engineering) from SEUA (State Engineering University of Armenia), Yerevan, Armenia.