International Journal on
"Technical and Physical Problems of Engineering"
(IJTPE)
Published by International Organization on TPE (IOTPE)

ISSN 2077-3528
IJTPE Journal
www.iotpe.com
ijtpe@iotpe.com

March 2010 | Issue 2 | Volume 2 | Number 1 | Pages 73-78

# COMPOSITION OF A GAME BASED SIMULATION FOR SOFTWARE DEVELOPMENT PROCESS

**A. Tizkar Sadabadi**

*Department of Computer Science, University of SEUA (State Engineering University of Armenia), Yerevan, Armenia*
*al_tz2@yahoo.com*

**Abstract-** Simulations are today very common and are used frequently for education, testing, research, development, gaming etc. No matter in what area it is used, it is used for the same purpose, to imitate a realistic event. With a simulation, certain factors are manipulated depending on what event is being simulated, like for example at a school for pilots, they use a simulator to imitate the flight. However, this report is not about flight simulators, but about how to proceed to develop a game, which simulates a certain software development 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. Why can simulation enhance traditional software engineering? 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.

**Keywords:** Simulation, Software Development Process, E-Education, CBT.

## I. INTRODUCTION

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. Most of us are familiar with public disasters such as failed Mars landings, rockets carrying satellites needing to be destroyed shortly after takeoff.

We believe the root cause of this problem lies in education: current software engineering courses typically pay little to no attention to students being able to practice issues surrounding the software engineering process. The typical software engineering course consists of a series of lectures in which theories and concepts are communicated, and, in an attempt to put this knowledge into practice, a small software engineering project that the students must develop.

## II. SELECTION OF SIMULATION MODEL

Simulations are made up events that are supposed to act realistic, and when they do, they are a simulation of a realistic event. Simulations are very common today in different working areas. Like the pilots for instance, they do not have to get into a plane right away and learn how to fly, first they simulate a flight on a flight simulator to practice what they have learned during their studies. Then when the "teacher" thinks that the students are ready, they can practice with a real plane. That is why simulators are developed in the first place, to teach people how to do things, before they actually do it in real life.

There are different kinds of simulators, continuous- and discrete simulation models. The continuous model can, strictly speaking, only be applied on an analog computer since the philosophy of the continuous model is that there is a continuous time flow and the simulation is constantly progressed [1]. But since there are no such thing as 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. Figure 1 shows the progress in a continuous simulation model.

The discrete simulator model uses time steps, and can simply be exemplified as a calendar. Days are passing, and during every day, a certain amount of work is accomplished, but the result is only shown after every ended day. This means that during the simulation for one day, nothing is shown. The characteristics of the discrete model, is that it involves time steps and finite number of events, and between those events, nothing happens [2]. Figure 2 shows the progress in a discrete simulation model, and because of the time steps, the curve gets the look of a stairway.

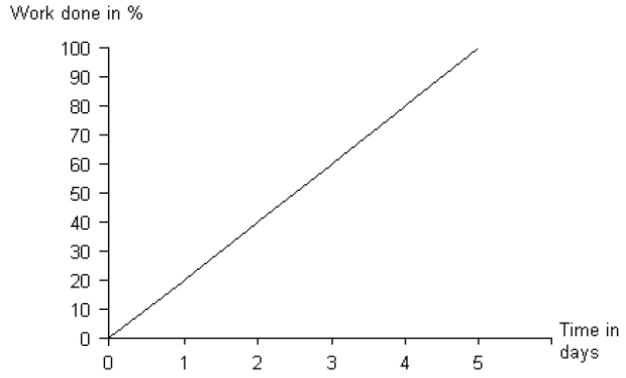


Figure 1. Continuous simulation model

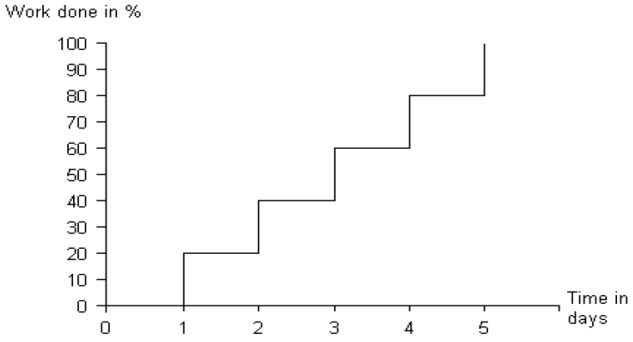


Figure 2. Discrete simulation model

To compare the continuous simulator in the example with the calendar, it would show the work progress during the simulation of every day that gives a straight line as Figure 1. The continuous model could be applied like a discrete model. If the time steps are small enough, the user will not be able to determine the difference between a continuous- and a discrete simulation model.

Simulating different processes in software development is possible if the processes are fully understood by the developer. The article [3], which focuses on the requirement phase, is an example of how it can be done with a discrete simulation model in Figure 3.

This is a part of the model that was developed, and describes the elicitation and documentation of requirements. In this part of the model, there is a time step after every elicited requirement, which is shown in Figure 3 as a loop. The time steps are very small in this model and therefore called a continuous simulation model. This model span over a specific time range, and until the time limit is reached, the loop is simulating the elicitation of requirements. When the time limit is reached, the simulator has produced an amount of requirements which can be further used in a software development project, for example to make a design based on the elicited requirements.

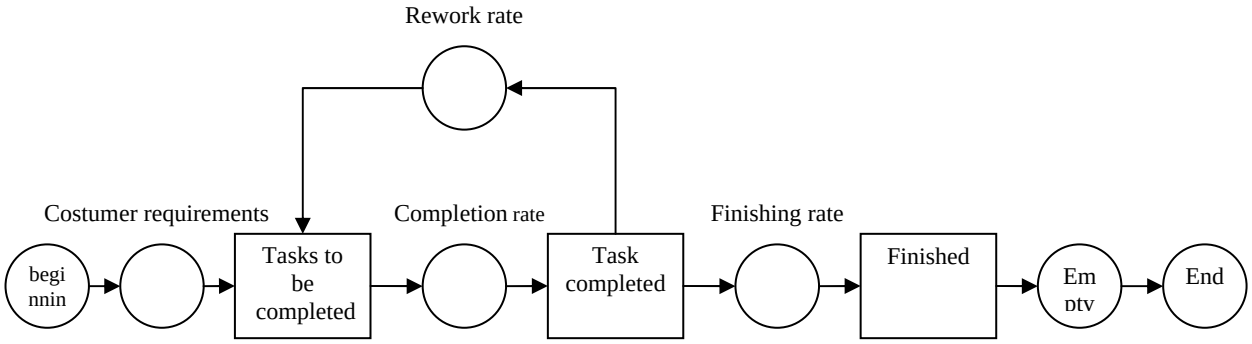


Figure 3. The requirements elicitation part of the simulation model [4]

This continuous simulation model if developed for every phase in a software development project using the waterfall model, could simply be described as one full glass of water and four empty glasses. The full glass of water corresponds to the estimated elicited requirements, and the empty glasses correspond to every phase; requirements elicitation, design, implementation and test. When pouring the water from the first glass, the requirements elicitation, to the next, which corresponds to the design, there is a little less amount of water in the second glass than the first glass, since there are usually some drops left in the first one. This means that there can at most be the same amount of water in the second glass as there was in the first one, but as mentioned before usually a little less. When pouring from the second glass to the third, which corresponds to the implementation, there cannot be more water than in the previous glass. This signifies that if the water from the first glass is not poured correct and maybe spilt, the spilling could be compared with a real software development project as e.g. neglecting a review, and this will have consequences for the further pouring. This leads to that the fifth and last glass will usually have a certain amount less water compared to the first one. The spilling, during the pouring, can be compared to different factors, which can be affecting the result of the project. Take for example if a project member becomes sick, and if the player does not hire a substitute, this could be exemplified as a spill during the pouring.

## III. COMBINATION OF WATERFALL MODEL AND SIMULATION MODEL

In this waterfall model, there are 4 phases to go through; requirement elicitation, design, implementation and test. Every phase is documented and reviewed, and a new phase cannot be begun unless the previous phase documentation is in baseline. When a document is put in baseline, it means that it has been inspected and approved

by the project leaders and project members, and the project is ready to take a further step to the next phase. The documentation in every phase is presented below in Figure 4, and it is important to take in concern that this development model was designed for a specific course, and is not a standard development model.

Every phase produces different documents, which are reviewed before they are put in baseline. In formal reviews, all project members are present and members from the different subgroups inspect the other subgroups document to detect errors. The informal review is similar to the formal, except that the customer is present during the informal review. The different documents in this waterfall model are as follows [5]:

SDP: Software Development Plan
SRS: Software Requirements Specification
SVVS: Software Verification and Validation Specification
STLDD: Software Top Level Design Document
SVVI: Software Verification and Validation Instruction
SDDD: Software Detailed Design Document
SVVR: Software Verification and Validation Report
SSD: Software specification Document
PFR: Project Final Report.

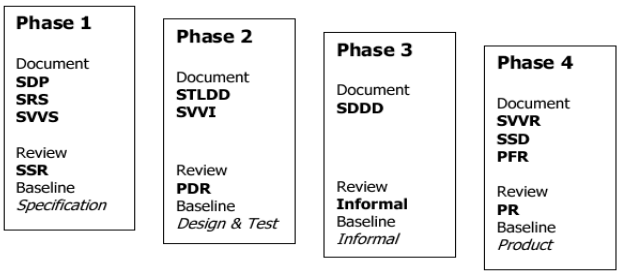


Figure 4. Phases of the waterfall model

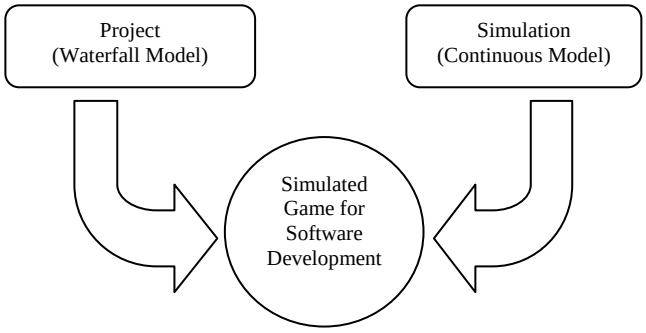


Figure 5. A basic model of the composition of the software development game

### A. Phase 1

In phase 1, requirements elicitation phase, the project leaders produce a software development plan, which will be followed during the whole project. Then every subgroup evoke requirements for their specific feature, and reviews them formally within the subgroup, and documents it in the software requirements specification. System testers do the same thing, they make use-cases from the evoked requirements and documents it in the software verification and validation specification. When all documents are finished, they go through a formal review before they are put in baseline. The formal review includes all project members and all members review the other member's work, like one subgroup reviews the other subgroups work and so on.

### B. Phase 2

In this phase, the design and test phase, the software engineers document all methods, variables and signals, which will be used and implemented, in the software top level design document. The system testers make test instructions to cover every requirement, and document it in the software verification and validation instruction document. They also produce monitor files, for module- and integration testing, and checks that all requirements are covered. When those documents are finished, they go through a formal review, including all project members, before put in baseline.

### C. Phase 3

The third phase is where the implementation is done and the system goes through the final test. The software engineers produce the software detailed design document, which describes the methods in a lower level, code level. The system testers make a final test of the system, called system test. When the document is done and formally reviewed, all documents from all phases go through an informal review. The informal review is like the formal review, only that the customer of the system is present and reviews the work of the project.

### D. Phase 4

In the final phase all documentation is completed and the system has passed all tests. All test documents, and results from the tests, are put into a software verification and validation report, and all design documents are put into a software specification document. The last document that is produced is the project final report, which includes all documents in one, and is written in a language so that software engineers can understand it. Finally, the project report is reviewed by the project team and ready to be handed over to the customer.

A combination of all these parts into one unit is possible if there are clear directives about which models to use. It has been decided, that this prototype will include three factors, which can affect the result of the game. The most important factor is reviews, and the other two are project members getting sick and project budget. A combination of these three factors can give a little reality to the game and give the player the possibility to steer the project. These factors have been implemented into the simulation model, which have been developed for this work. A discrete simulation model was developed with small time steps, this so it will act as a continuous simulator, based on the waterfall models criterion, and could be built up as a game. Figure 6 shows the composition of the different parts.

As in this case, in the project part, the waterfall model has been used, and in the simulation part, the continuous simulation model has been adapted. It is called continuous, since the time steps of the discrete model are small enough to act as a continuous simulator.

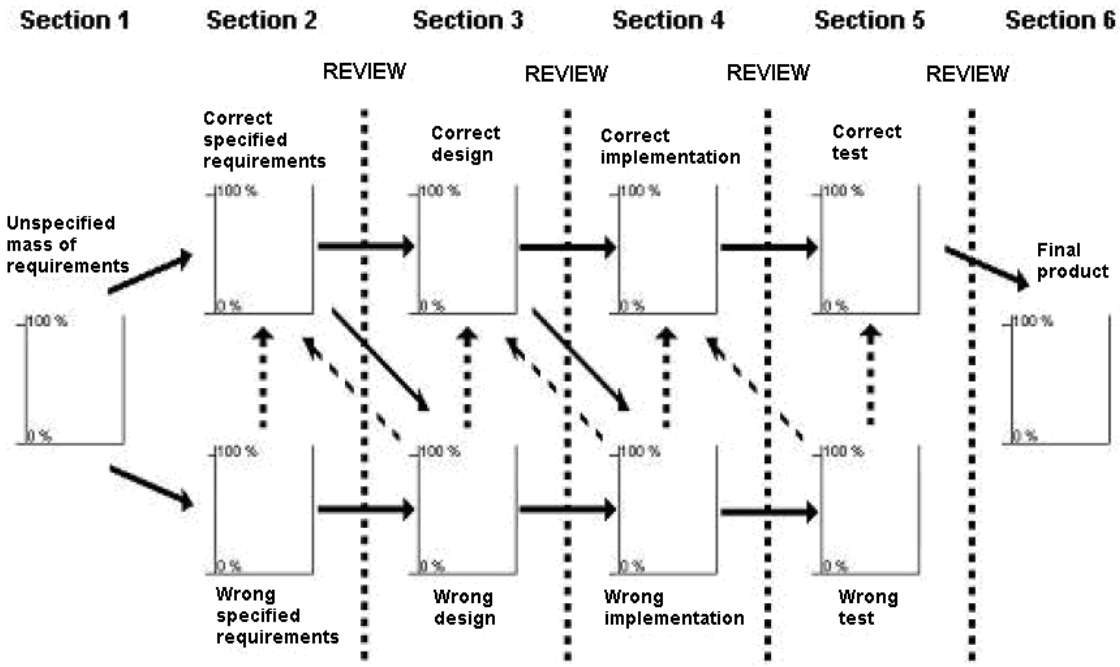


Figure 6. The simulation model, developed for this work

## IV. REQUIREMENTS SPECIFICATION

This will be a one-player game, where the player takes the role as a project leader, PL, for a software development project using a waterfall development model [6]. The PL's task is to lead a software development project and make necessary decisions to finish the project as well as possible. The PL gets an amount of money depending on the size of the project, and has to dispose them right, so he can proceed with the project and get an approved result. There are 4 phases of the game which the PL have to complete, and a phase cannot be initiated unless the previous phase has been completed and the phase document put in baseline. Every phase of the game gives a result in percent compared to the estimated percentage. For example in the requirements phase, the estimated evoked requirements is always 100%, and the result will be close to it, depending on how many reviews are made and if no one gets sick. Below are the different outcomes of the game:

- Project Failed (If quality percentage is below 95%)
- Project Failed (If budget is exceeded)
- Project success (If quality percentage is higher than 95% and budget not exceeded)

Player objectives are as follows:

- Project leader: A person who plays the game.
- Project leader objective 1: Initiate the project by choosing how many functions, 1-5, will be implemented and choose reviews, for every specific phase.
- Project leader objective 2: Choose extra reviews during game.
- Project leader objective 3: If a project member is sick, choose to hire a substitute.

Some guidelines were created to follow a structured plan with intermediate goals. The development model, used for this work, was an incremental prototyping model, which means that everything has been developed from an undefined requirements specification, more like a work description, and additions have been frequently made during the project.

The prototype includes a simulation model, Figure 6, which simulates all four phases in the waterfall development model; requirements specification, design, implementation and test. The same model is used for every phase, and shows the result of developed quantity vs. estimated developed quantity in percent. In the game, the quantity percentage results will become the quality percentage of the developed product at the end of the game.

Figure 6 shows the developed simulation model for this thesis and it is divided into 6 sections. Every section includes one or two tanks, with one start section and one end section. In the start section there is an amount of unspecified requirements in that tank, and it is always 100%. In the next section, which simulates the requirements elicitation, there are two tanks, one for the correct specified requirements and one for the wrong specified requirements, and the estimated elicited requirements is always 100%, customer demands. Section number three is the design phase and contains of two tanks as well, correct design and wrong design, and the estimated design depends on the outcome of the requirements phase. Next section, number four, is the implementation phase, which also contains two tanks, correct implementation and wrong implementation, and here the estimated implementation is also depending on the result of the previous phase, the design phase. Section number five is the test phase, containing two tanks as well, correct test and wrong test, and the estimated test is here depending on the result of the implementation phase. The last section is the final product and determines the quality of the product, depending on the result in the test phase. As described here, all future results, depend on the

results in previous phases. In Figure 6 we can see that there is a broken line after each phase that signifies review, and if chosen, then the broken arrows are used, this means that if there are no reviews selected, then no broken arrows are used either.

When running the simulator, most of the percentage from the start tank is moved to the correct specified requirements tank, and a little percentage is moved to the wrong specified requirements tank. The black arrows show the procedure of the normal movement of percentage between the tanks. During phase one, the player can affect the outcome of the requirements phase by adding a review. If the player chooses to add a review, a little percentage will be moved from the wrong specified requirements tank to the correct specified requirements tank, which is indicated by the broken arrow. This means that the percentage in the correct specified requirements tank increase and the percentage in the wrong specified requirements tank decrease with the same amount. After the first phase is accomplished, there is a certain amount of percentage in the correct specified requirements tank, and that amount constitutes the maximum percentage for the design phase. This means that if the percentage in the correct specified requirements tank is 98 %, the correct design, cannot be higher than that. The simulation proceeds like the previous phase, most of the percentage is moved from the correct specified requirements tank to the correct design, and a little percentage is moved to the wrong design, which is indicated by the black arrows. If the player, choose to add a review, a little percentage is moved from the wrong design tank to the correct design and to the correct specified requirements tank, broken arrow. The percentage, which is moved to the correct specified requirements tank, is automatically moved to the correct design tank. The following phases use the same principle, which means that the correct percentage is decreasing through the whole project, but can be adjusted at bit with reviews. After the last phase, the test phase, is finished, there is an amount of percentage in the correct test tank. This percentage constitutes the quality of the final product. This means that the correction of the tests is included in the review option in the test phase. If the player chooses review in the test phase, it automatically corrects the test cases. One thing in the test section which differs from the other sections is that there is no arrow from correct implementation to wrong test, this because when something is correct implemented, it cannot be wrong in test.

Since this is a prototype, there are only three factors chosen which can affect the outcome of the game, if all factors would be implemented, the game would not be finished within the time limit of the thesis. The factors included are; budget, the choice to make reviews and the choice to hire a substitute if a project member is sick. The review- and sick member factors affect the budget, which is different for every level of the game, and depending on the project leaders decisions, the outcome of the game differs. Levels are different degrees of difficulty in game.

## V. DESIGN

First a paper design prototype was developed for the GUI, to show different positions of the functions in the application. This was made to get new ideas about the interface and its appearance, to make it as simple and understandable as possible. Figure 7 shows the initial window of the first paper design prototype.

The choice of factors that can affect the project result, had to be done before the design, so there would be a structure to follow during the design. As mentioned before, the three factors were:

- Budget
- The choice to make reviews
- The choice to hire a substitute if a project member gets sick.

With the first design sketch as basis, the design of the GUI was experimented with through implementation and consulted spontaneously with a couple of students to get a satisfying appearance

## VI. CONCLUSIONS

Simulation is the best way to bring practical training to classical theoretical education. As that was selected one way to realize this theory. This model isn't so comprehensive and complete for this purpose so this work will be going. Numerical proposes were simple. For this we can imagine other ways for numerical results.

Due to the fact that computer games are very popular these days, the thought of making a simulator game for educational purposes is close at hand. The game is a simulator game, based on software development and was experimented in a prototype with a few features. The used development model for this prototype is a waterfall model.

The used simulation model, is a discrete model, implemented with very small time steps, so it appears as a continuous simulation model, see Figure 6. Since this game was not developed for entertainment, the focus has been directed to the simulator to get a fairly realistic game. To make it fairly realistic, certain factors had to be added to affect the result of the game. In software development projects, there are many factors that can affect the project, and for this experimental prototype, three were chosen;

1. reviews
2. budget
3. The choice to hire a substitute if a project member gets sick.

With those factors, a playable prototype was developed and evaluated for further development. With certain modifications and changes, the prototype could be improved and be a possible tool for illustrative education and explanatory purposes. The target group would in that case be students and future software engineers with some experience in software development. As the prototype is today, it cannot be used for any of these purposes, it has to be modified and improved at certain points.

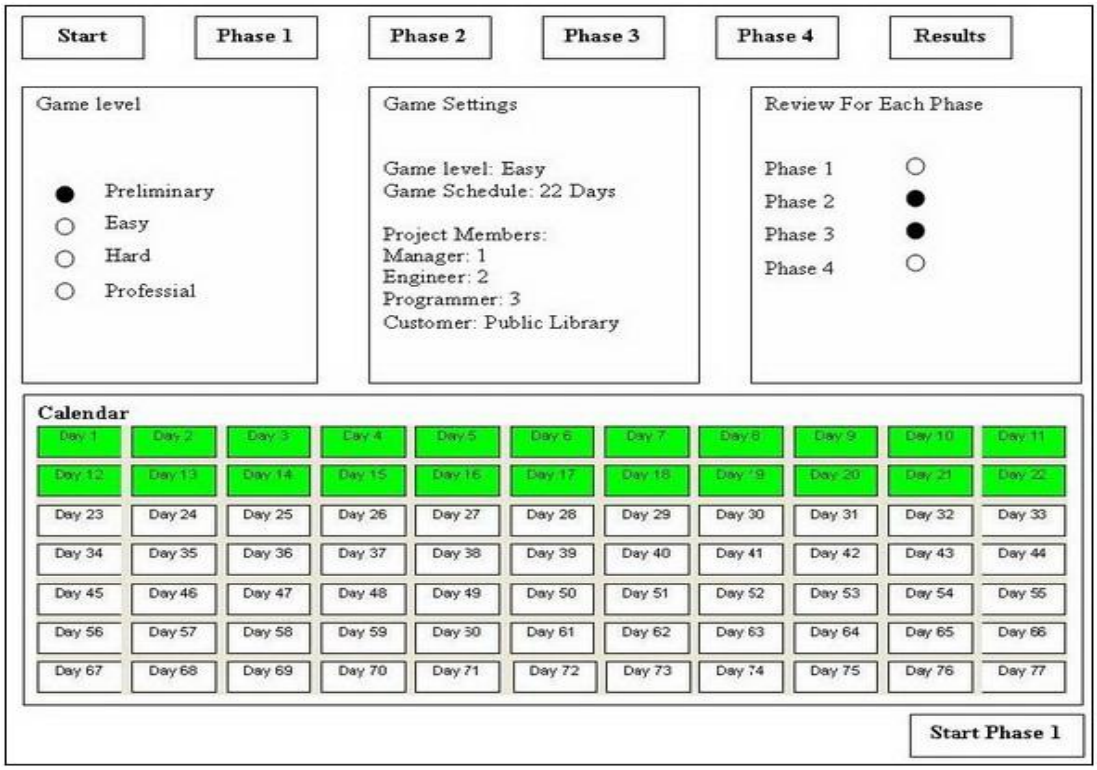


Figure 7. Sketched prototype for simulator's first page GUI


## ACKNOWLEDGEMENT

The great work of Ms Emily Navarro that were a doctoral thesis and other parts for software research at University of California, Irvine, was a great help for developing this project and also with the cooperation of my master thesis's supervisor Dr. Souren Khachatryan that was from another university and spent a valuable part of his time for my project.


## REFERENCES


[1] G. Bellinger "Simulation Types", http://www. outsights .com/ systems/simulation/.

[2] J.F. Ramil, "Introduction to System Dynamics Software Process Modelling", http://www.doc.ic.ac.uk/ ~mml/feast2/papers/pdf/jfr89c_lec1.pdf.

[3] C. Andersson, L. Karlsson, J. Nedstam, M. Host and B.I. Nilsson, "Understanding Software Processes through System Dynamics Simulation: A Case Study", Proceedings of IEEE Conference on Engineering Computer-Based Systems (ECBS), pp. 41-48, Lund, Sweden, 8-11 April 2001.

[4] R. Acosta, C. Burns, W. Rzepka, and J. Sidoran, "Applying Rapid Prototyping Techniques in the Requirements Engineering Environment", IEEE, 1994.

[5] "Programvaru Utveckling för Stora System", Kompendium 1, Projekthandledning Björn Regnell, Claes Wohlin, 1999.

[6] B. Regnell and Cl. Wohlin, "Programvaru Utveckling för Stora System (PUSS)", Compendium 1, Projektledning, V1.6, 1999.


## BIOGRAPHIES

**Ali Tizkar Sadabadi** was born in 1981 in Tabriz, Iran. He is a graduated M.Sc. student and ready for Ph.D. in the field of computer science and specialty of software engineering. His academic backgrounds are B.Eng. degree from Azad University, Tabriz Branch, Tabriz, Iran during 2000-2006 in the major field of Software and Computer Engineering and also M.Eng. degree from SEUA (State Engineering University of Armenia), Yerevan, Armenia during 2006-2008 with the title of master dissertation: "A Survey on Simulation Game of Software Development Process (Software Engineering Simulation by Animated Models)".

Mr. Ali Tizkar Sadabadi was an honorable student and is on Honors and Awards of 1- Obtaining third place in provincial competition in physics laboratory's tournament, 2- Honors in high school graduation, 3- Gaining honorable places in several body readinesses' matches.