ISSN:0975-9646

Economic Cost Model for Software Engineering Simulation

E.Geetha Rani[#1], K.Swarupa Rani[*2], D.Anusha[#3], Dr.M.V.L.N.Raja Rao[*4]

[#1 , #4] *Information Technology,*
Gudlavalleru Engineering College, Gudlavalleru,
JNTUK, INDIA

[#3 &*2] *Information Technology,*
PVPSIT,Vijayawada, jntuk, INDIA

ABSTRACT: The common software engineering education method of theory presented in lectures along with application of these theories in an associated class project is insufficient, on its own, to effectively communicate the complex, fundamental dynamics underlying real-world software engineering processes. We are constructing a new approach to software engineering education that is based on the use of an educational software engineering simulation environment. However, a major challenge in developing such an environment lies in creating an accurate model of the real world upon which the simulation is based. In order for the simulation to be a successful educational tool, this model must be based on an appropriate economic model, must consist of the correct "fundamental laws" of software engineering, and must encode them quantitatively into accurate mathematical relationships, thereby correctly embodying and portraying all of the various factors, dynamics, and cause and effect relationships present in the real-world software engineering process.

Keywords: Software Engineering Education, Simulation

1 GOAL

Given the ubiquitous nature of software in our society, it should come as no surprise that the discipline of software engineering has taken a prevalent role, both in academic research and in industrial practice. In parallel, of course, software engineering education has received significantly increasing amounts of attention as well, evidenced by, for example, a special track at the main conference on software engineering [2], a separate conference dedicated to software engineering education [18], the SWEBOK project [4], special journal issues dedicated to the topic [1, 5], and even the introduction of specialized software engineering degrees [8, 15]. Clearly, all of these efforts are aimed at creating an understanding of the issues involved in teaching software engineering, as well as at sharing approaches to further improve the way software engineers are educated. Despite all of this attention, a remarkable difference remains between the software engineering skills taught at a typical university or college and the skills that are desired of a software engineer by a typical software development organization. At the heart of this difference seems to be the way software engineering is introduced to students: general theory is presented in a series of lectures and put into (limited) practice in an associated class project. While at first this seems to be a reasonable approach, practical, didactic, and timing reasons necessarily lead to the fact that such lectures and class projects often lack an in-depth treatment of the following five issues critical to any real world software engineering project:

- *Software engineering is non-linear.*
- *Software engineering often has multiple, conflicting goals.*
- *Software engineering continuously involves choosing among multiple viable alternatives.*
- *Software engineering involves multiple stakeholders.*
- *Software engineering may exhibit dramatic consequences.*

In essence, all of these issues relate to the overall process of software engineering, which is difficult to teach in lectures, since it remains abstract, and difficult to teach in a class project, since it requires a project of significant size to highlight the issues. Nonetheless, educating students in these issues is essential to creating a full understanding of the depth and complicated nature of software engineering. Simulation is a powerful training technique that has been successfully used in many different settings. Before airline pilots actually fly a plane, they extensively train in simulators. Military personnel practice their decision-making and leadership abilities in virtual-reality simulation environments. In each of these cases, simulation provides significant educational benefits: valuable experience is accumulated without the potential of the dramatic consequences that may occur in case of failure. Moreover, unknown situations can be introduced and practiced, experiences can be repeated, alternatives can be explored, and a general freedom of experimentation and "play" is promoted in the training exercise. Our research project is based on the hypothesis that simulation can bring to software engineering education many of the same benefits that it has brought to other domains. Specifically, we believe that simulation is the ideal platform upon which to teach the above issues. As compared to lectures, simulation has the distinct benefit of showing and teaching students cause and

effect in a practical manner: if they make a wrong decision in the simulation, it will (hopefully) become clear to them because the simulation environment will show them certain undesired effects. As compared to a class project, simulation has the distinct benefit of being much quicker: one does not have to wait days, weeks, or even months to see the effects of a decision, since the simulation environment is able to operate at a faster pace than real life. In essence, simulation allows a practical experience of the software process without the additional, distracting burden of having to produce project deliverables. In order to be an effective educational tool, simulation must be based on a model that accurately embodies the dynamics of the real world process it represents. For a software engineering simulation in particular, this accuracy is attained by successfully communicating each of the five fundamental issues mentioned previously. An interesting observation to make is that these issues generalize to other domains. However, a number of difficulties arise in adopting such cost models for our purposes. In this paper, we highlight some of these difficulties and identify some avenues of addressing them.

2 ARCHITECTURE

Our simulation environment provides the user with a game-like experience: all output is presented in a graphical user interface, which realistically portrays all of the characters, surroundings, artifacts, causes and effects of decisions, and other various details present in a real world software engineering environment. As such, our environment is similar to games like SimCity and The Sims, and builds on many of their lessons learned in providing the desired level of functionality while maintaining a graphical and entertaining environment in which users can learn effectively. Perhaps the most important of these lessons is the fact that, while the user controls the game through the perspective of a single character, other characters behave autonomously and typically interfere with the user in achieving their goals 100 percent. Our simulation environment employs this tactic as well: while a user may control, for example, the character of a project manager, the simulation environment may direct that some of the employees check in sick periodically, or are not as productive as they should be, or spend too much time at the coffee machine talking. Like any other simulation environment, our educational Software engineering environment is based on the basic simulation process shown in Figure 1.

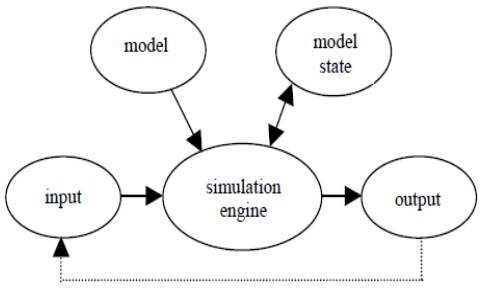

Figure 1: Basic Simulation Process.

At each step in the simulation, input to the simulation engine consists of commands provided by the user of the simulation. The simulation engine uses this input, along with the simulation model and the current states of the model, to, step-by step, calculate the state of the simulation as it progresses. The output is then provided to the encompassing simulation environment, which graphically displays the result.

3 MODEL

A simulation model consists of a set of mathematical and logical relationships that, collectively, represent the rules underlying the behavior of the real-world process it embodies. Any simulation environment is driven by such a model, and our simulation environment is no exception to this rule. In fact, its accuracy and effectiveness in achieving its educational purpose strongly depends on the characteristics of this underlying model. Because of this importance, the creation of the model is a rather challenging process. Specifically, four major questions need to be researched regarding the requirements, design, implementation, and operation of the model. What kind of model is needed? Given the five characteristics of software engineering (non-linear, multiple conflicting goals, multiple viable alternatives, multiple stakeholders, and dramatic consequences) it is clear that the software engineering process can be viewed as a constraint satisfaction problem. To model this kind of problem, a generic mathematical model can be adopted, but several of the approaches developed to model aspects of software engineering with an economic cost model apply as well. As such, we are faced with the question of which model (or integrated set of models) to use to drive our simulations. Two of the most important requirements are that the model is incremental and modular. Instrumentality is model state input output simulation engine needed such that the model can be used on a step-by-step basis, rather than as a "prediction" kind of model that only allows a single run-through (e.g., COCOMO [6], or a probabilistic model [16]). Modularity is needed because we plan to develop many different simulations, which, over time, we expect to integrate in large-scale simulations. Thus, it is required that the partial models we develop can be integrated with relative ease. What are the "fundamental rules" of software engineering and how from where can they be discovered? Like any other discipline, software engineering has many underlying empirical rules. For example, it is well known that adding people to a project that is already late typically makes that project later, due to the increased necessity for communication between personnel. Our simulation environment aims to provide a real-world experience and, thus, its model must be solidly rooted in such real-world phenomena. Unfortunately, the set of rules of software engineering is published in a wide variety of media (software engineering journals and conferences, computer-supported collaborative work journals and conferences, books, trade literature, etc.) and no single source exists in which all are compiled. Therefore, one of the challenges in creating an accurate model lies in researching, identifying, and compiling a list of the fundamental rules of

software engineering. How can the "fundamental rules" of software engineering be encoded into an executable model? Once we have chosen a particular kind of model, several questions about the parameterization of the model follow: What are the constraints and the variables whose values must obey those constraints? What are the constants that influence the values of those variables? What are the equations that embody the cause and effect rules determining the behavior of the model? How are the (often conflicting) overall goals of software engineering and the individual goals of each entity involved in the simulation encoded into the model? As an example, consider the following simulation scenario that illustrates the software engineering "law" which says that skipping the design phase leads to highly problematic integration:

The developers proceed directly from the requirements phase to implementation, skipping the design phase completely. When they begin to integrate, the error rate of the software skyrockets, the quality of the software drops dramatically, and each developer's mood plummets. They must

Spend several months (while the cost meter is ticking away) integrating all of the different developers' pieces of code before the system works.

Expressed qualitatively, this situation is easily described and well understood. However, in order to make this scenario executable in a simulator, a quantitative representation of its behavior, including mathematical equations describing the relationships between all of the different variables and factors involved, is needed. For instance, exactly how many person-months longer does development take when the design phase is skipped? Precisely how many more bugs are present in a piece of software that was developed without a design phase than one that was thoroughly designed before it was implemented? How much does each developer's motivation actually drop as the result of such a situation, and how, in turn, does this affect the resulting productivity of the team? In essence, an exact schema with which to evaluate the precise cost of each action the player can take must be adopted. We intend to leverage information from sources such as COCOMO [6] in creating models that are as close to the real world as possible, neither overplaying nor underplaying the effects portrayed in the simulation. How does the model work? A simulation used for education in particular needs to guide the player in implicit ways in regards to such issues as what steps to take, which decisions to make, and which choices are available for each decision. It also needs to have the ability to initiate the actions of characters in the game that are not controlled by the user, accept input from the user, and somehow balance the interaction between the two. Two challenges lie in accurately and efficiently incorporating this requirement into the actual execution of the model. First, it requires that our model make provisions not only for the overall behavior of the process, but also for the independent behaviors of each individual entity involved in the process. Moreover, the model must consider the interactions between these entities on both an individual basis and in terms of an overall net effect. A model with these capabilities is quite different from cost models introduced so far. Thus, careful evaluation of existing models, as well as considerable extension to one or more of these models, will be required to achieve the necessary functionality.

4 Related Work

This research draws from several related areas, most notably software engineering education, process simulation, games, and economic cost models for software engineering. This section briefly discusses the contributions in each area that are relevant to the construction of our simulation environment

Software Engineering Education It is clear that educational methods in software engineering are still very much dominated by the traditional model of teaching theory in a series of lectures and putting that theory into (limited) practice in an associated class project. Pressured by industry to deliver students who are more in tune with recent advances and new technologies, as well as students who are more adept at understanding the difficulties involved in the software process, numerous variations on this basic method of software engineering education have been developed [7, 10, 14]. Several of these approaches have incorporated simulation, the most advanced of which is represented by SESAM, a simulation environment for software engineering education that has been applied in classroom settings [11]. However, compared to the research we propose, SESAM is limited in functionality. First, SESAM's play is linear in nature, following, in order, each step of the software life cycle. Second, SESAM is text-based and lacks any kind of "fun" graphical user interface. Third, the models developed to date are limited and are typically based on only a few different roles and rules of interaction. Fourth, a player can only play the role of a project manager no controls are provided for any of the other (simulated) characters. Despite these drawbacks, SESAM's models do provide a source of some well documented rules of software engineering, and its simulation engine may be reusable for our needs.

Process Simulation Many software process simulations have been developed and used to analyze the characteristics and behavior of the process being modeled and to predict the effects of process changes [3, 13]. These all operate according to the same basic philosophy of creating a model of a real-world process, choosing a set of input parameters, running the model, and examining the outputs together with traces of the simulation to understand the workings of the environment. Despite the fact that these simulations are passive in that they run without interruption until finished, the models and the rules underlying those models are pertinent to our simulations since they share the purpose of modeling real world phenomena.

Games Simulation games represent a tremendous source of experience that can be leveraged in creating models for an educational software engineering simulation environment. A class of games that is particularly relevant is the one derived from the so-called "adventure games" of the olden days now represented by such popular games as Sim-City, The Sims, Escape from Monkey Island, Myst, Ultima Online, various

MUDs and MOOs, and many others. In these games, players work towards achieving certain, sometimes conflicting goals, by living their "virtual lives" in such a way that they must make tradeoffs in choosing to work towards certain goals while ignoring others, much like the process of software engineering. These games also illustrate many examples of good and effective design that can be used in our simulation environment. They are fun to play, encourage experimentation, usually have an excellent graphical user interface, have immediate as well as time-delayed cause and effect relationships, and bring the player into unexpected, unknown situations that need to be resolved. Moreover, the models upon which these games are based exhibit all of the desired characteristics required for our educational software engineering simulation environment:

- **They are non-linear.** Multiple events happen at the same time; one must frequently interrupt certain activities to tend to others; and generally playing the game in the same way every time will not lead to the same results, due to the presence of several random factors in the simulated characters and events.
- **They involve multiple, conflicting goals.** As explained previously, the games involve optimizing multiple goals that sometimes interfere with each other. Player's actions inherently weigh certain goals as more important than others, and generally lead to certain goals that are attained and others that can only be partially fulfilled.
- **They allow for the exploration of alternatives.** All games allow a player to save the state of the game, in effect providing a checkpoint ability that can be leveraged to explore different directions without committing oneself—simply returning to the saved state allows for exploration of a different alternative.
- **They generally involve multiple stakeholders.** In some games, these stakeholders are represented by the different players that each try to optimize their own results. In other, single-user games, the stakeholders are provided by the game simulation. For example, SimCity has unions and Green Party representatives that the player must keep happy in making decisions regarding city planning.
- **They exhibit dramatic consequences.** Although not real, the graphical illustration of these dramatic consequences (which range from the player actually being killed, to buildings being destroyed by natural disasters, to dirty houses being invaded by rats) has a profound impact on the player. Thus, since these game models exhibit the desired characteristics of our simulations, we intend to leverage these kinds of models in the creation of our environment.

Economic Cost Models for Software Engineering. Several economic models of the software engineering process, based upon such concepts as Net Present Value [12], financial portfolio analysis [9], and Return on Investment [17], have been developed and applied to evaluate various aspects of software development projects. These have all been created mainly for the purposes of either facilitating more accurate software project planning, supporting managers in making decisions about software projects, or predicting the effects of process changes. Each of these models accomplishes its purpose by estimating overall net measurements of the process, such as development time, cost, and quality. The obvious relevancy of this domain to our research lies in our intended adoption of one of these models as a basis upon which to create our simulation model. However, these models in their current state do not fit the needs of our simulation environment, namely, an incremental nature of operation, the capacity to be decomposed into partial models, and the ability to recognize individual entities and their interactions with each other. Nevertheless, it is expected that investigation of these models will yield valuable knowledge that can be used in the creation of our simulation model, and that by incorporating and extending one of more of these models, one suitable for our needs can be developed.

5 CONCLUSION

We are constructing a new approach to software engineering education that integrates software process simulation, simulation games, and economic software engineering cost models into an educational software engineering simulation environment. This environment addresses the problems inherent in the current methods of software engineering education by effectively teaching students the complex, yet fundamental issues and dynamics that underlie the software engineering process. We have begun to take the first steps in building this environment by performing an extensive survey of software engineering journals, conference proceedings, workshop proceedings, and books, as well as literature from other related disciplines, in order to collect the fundamental rules of software engineering. It is this set of rules that will form the basis for our simulation model. Challenges lie ahead in encoding these rules into an executable model, choosing a particular kind of simulation model, and tailoring the simulation to meet the specialized, educational requirements for this particular environment. However, as demonstrated in this paper, their application is not as straightforward as one would ideally like. Nonetheless, it is our belief that adapting one of these cost models is more efficient and will lead to better results than simply building a simulation model from scratch.

REFERENCES

1. *The Journal of Systems and Software*. Vol. 49. 1999: Elsevier Science Inc.
2. *Proceedings of the 22nd International Conference on Software Engineering*. 2000: ACM.
3. Abdel-Hamid, T., *Lessons Learned from Modeling the Dynamics of Software Development.* Communications of the ACM, 1989. **32**(12): p. 1426-1438.
4. Bagert, D.J., et al., *Guidelines for Software Engineering Education Version 1.0*. 1999, Carnegie Mellon Software Engineering Institute: Pittsburgh, Pennsylvania.
5. Balci, O., *Annals of Software Engineering*. Vol. 6. 1998: Baltzer Science Publishers.
6. Boehm, B.W., et al., *Cost Models for Future Software Life Cycle Processes: COCOMO 2.0*. 1995, University of Southern California.

7. Boehm, B.W., et al., *A Stakeholder Win-Win Apprach to Software Engineering Education*, in *Annals of Software Engineering*, O. Balci, Editor. 1998, Baltzer Science Publishers. p. 295-321.
8. Boldyreff, C., *The University of Durham BSc in Software Engineering and Proposed MEng in Software Engineering: A Position Paper*, in *Proceedings of the Thirteenth Conference on Software Engineering Education and Training*, S. Mengel and P.J. Knoke, Editors. 2000, IEEE Computer Society. p. 189.
9. Butler, S., et al., *The Potential of Portfolio Analysis in Guiding Software Decisions*, in *Proceedings of the First Workshop on Economics-Driven Software Engineering Research*. 1999.
10. Dawson, R., *Twenty Dirty Tricks to Train Software Engineers*, in *Proceedings of the 22nd International Conference on Software Engineering*. 2000, ACM. p. 209-218.
11. Drappa, A. and J. Ludewig, *Simulation in Software Engineering Training*, in *Proceedings of the 22nd Internation Conference on Software Engineering*. 2000, ACM. p. 199-208.
12. Erdogmus, H., *Comparative Evaluation of Software Development Strategies Based on Net Present Value*, in *Proceedings of the First Workshop on Economics-Driven Software Engineering Research*. 1999.
13. Madachy, R., *System Dynamics Modeling of an Inspection- Based Process*, in *Proceedings of the Eighteenth International Conference on Software Engineering*. 1996, IEEE Computer Society.
14. Mayr, H., *Teaching Software Engineering by Means of a "Virtual Enterprise"*, in *Proceedings of the 10th Conference on Software Engineering*. 1997, IEEE Computer Society.
15. McCracken, M., et al., *A Proposed Curriculum for an Undergraduate Software Engineering Degree*, in *Proceedings of the Thirteenth Conference on Software Engineering Education and Training*, S. Mengel and P.J. Knoke, Editors. 2000, IEEE Computer Society. p. 246-255.
16. Padberg, F., *A Probabilistic Model for Software Projects*, in *Proceedings of the 7th European Engineering Conference held jointly with the 7th ACM SIGSOFT Symposium on Foundations of Software Engineering*. 1999, ACM. p. 109-126.
17. Raffo, D., J. Settle, and W. Harrison, *Estimating the Financial Benefit and Risk Associated with Process Changes*, in *Proceedings of the First Workshop on Economics- Driven Software Engineering Research*. 1999.
18. Ramsey, D., P. Bourque, and R. Dupuis, *Proceedings of the Fourteenth Conference on Software Engineering Education and Training*. 2001: IEEE Computer Society.