Modeling and Simulation of Software Projects

A. Drappa, M. Deininger, J. Ludewig, R. Melchisedech

Software Engineering Group, University of Stuttgart
Breitwiesenstr. 20 - 22, D-70565 Stuttgart, Germany
drappa@informatik.uni-stuttgart.de

1. Introduction to SESAM

Over the last twenty five years, software development projects have increasingly been plagued by schedule and cost overruns, while product quality is often poor. The term "software crisis" has been coined to highlight this situation. The lack of adequate methods and tools to support software development, and of techniques to simplify project management tasks have been identified as major factors contributing to the "software crisis".
In reaction to this, the software engineering community has attempted to devise a number of analytical and constructive operations to both handle the complexity of software development and assure quality of the resulting products. However, despite an impressive evolution of methods and technology in the past years, neither the software development process nor overall product quality have improved significantly. Part of the reason for this appears to be a serious lack of understanding of the software development process. It is often unclear how to integrate methods and tools in the development process to achieve best results because little is known about the most important influencing factors that determine process and product quality.

Our research project SESAM (**S**oftware **E**ngineering **S**imulation by **A**nimated **M**odels) attempts to address the problems outlined above. With the SESAM project, we pursue the following two main goals.

1) *Building quantitative models of software development projects to gain a better understanding of the underlying processes.*

   Detailed description of software processes is an absolute requirement for informed discussion about effects observable in software development projects. Software projects can be described by a set of simple effects, where each effect influences the course of the process and the quality of the resulting products. The software project is thus determined by simple rules and their effect on the whole process. Our goal is to identify these effects, and to describe them in a quantitative way wherever possible. Even though some of the quantitative effects are currently hypothetical in nature, the formal description provides a sound basis for their validation with empirical data.

2) *Simulation of software development projects, taking the models as a basis to educate future project managers.*

   Teaching project management from text books has been proved to be insufficient, while training on the job is difficult due to the length and costs of software projects. Taking the knowledge captured in SESAM models as a basis, the simulation of software projects allows students to gain reality-like experience without jeopardizing the progress of real

projects. Thus, the simulation offers the opportunity to transfer the available knowledge of software processes to project managers who in turn can apply this knowledge in real industrial projects.

The idea of SESAM was first described in 1989 [Ludewig89]. Since then we have built three consecutive prototypes, and have finished the system SESAM-1 in 1994. This work has produced new results in three important categories:

- more elaborate concepts of SESAM models,
- further development of a language to adequately describe those models, and
- new features needed for the simulation system.

A selection of these results will be presented in the following three sections. Finally, we will summarize our experience with a SESAM model in a project management course held at our department and briefly outline some of our current research activities.

2. Basic Principles and Related Work

The approach we took to develop SESAM is influenced by a number of underlying basic concepts. In the following section, we will outline our general approach, refer to some of the concepts developed by other groups and illustrate how our approach differs from previous attempts to model software development processes.

Building models that can be validated.

Effective teaching of software engineering and project management knowledge by simulation requires that the models used be realistic, provide quantitative information, and be capable of reflecting process and product quality. Thus in SESAM, we have to identify cause and effect relationships between objects involved in the software process and to describe them quantitatively as far as possible. Consequently, the SESAM models were designed to be able to reflect and process quantitative data. Since the software engineering knowledge currently available largely is a collection of "rules of thumb", our models have to be bolstered by empirical data from real projects. Therefore, SESAM models are intended to be detailed (fine-grained) so that results of the simulation can be directly related to real project data.

Learning by trial and error.

The traditional way of teaching project management is to convey one (the best) predefined solution of a given problem to a student. In our experience, teaching is more effective, and learning is much easier, if the student has a chance to try different solutions. Subsequently, he or she can analyze which solution is the best, and why. Therefore, software projects should be simulated by allowing the student to trigger any sequence of actions, driving the project in what he or she thinks is the right direction. Conversely, the simulation system reacts by presenting informal messages reflecting the current state of the project. As a consequence, SESAM models are built to be interactive, i.e. models must accept input from the student, and react appropriately to the actions taken.

Related Work

Quantitative modeling is not a new idea. The pioneering work of Tarek Abdel-Hamid aimed at gaining a fundamental understanding of software project management processes [Abdel-Hamid91]. His results have influenced a number of similar approaches [Levary91], [Carr90], the basic idea of which is to describe and simulate software processes using system dynamics. The drawback of system dynamics models is that they are neither interactive, nor fine-grained. While well suited to describe quantitative aspects, they do not provide means of interaction between model and student for training purposes.

At the present time, software process modeling is an active and growing area of research. An overview is presented in [Curtis92]. Among the principal goals of this research is the construction of Process Centered Software Engineering Environments (PCSEE) offering tools to facilitate product development and integrate information about the whole process. By strongly guiding the process, management tasks can be simplified significantly [Snowdon94]. The construction of PCSEEs also requires detailed description of software processes, and much effort has been devoted to the definition of new modeling approaches and languages [Finkelstein94]. These approaches, however, tend to emphasize a predefined course of the modeled project, and do not provide the flexibility required for our purposes.

3. SESAM – Modeling Approach and Language

We concluded that for our purposes, a new modeling approach was desirable. We set out to define a multi-paradigm language integrating a number of well established concepts.

The following description of the SESAM models is structured into two parts. The static perspective mainly focuses on the description and type definition of objects involved in the software process and possible relationships between these objects. The dynamic behavior is described by a set of generic rules specifying actions and effects which change the state of the simulated software project [Schneider94].

Static perspective – the scheme

To describe the static perspective called the SESAM scheme, the extended entity-relationship notation is used. Types of real world objects, like Developer or Document, and of possible relationships between these objects, like reads or writes, are identified and further described by attributes. Possible attributes of the entity type Developer include for example name, age, or experience. The type Document could be characterized by the attributes name, size, and #_of_faults. The scheme is described graphically. A very simple part of the scheme is shown in figure 3.1.

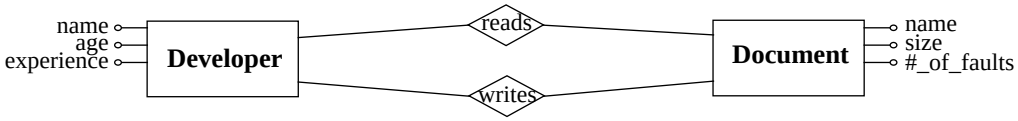

Figure 3.1: Part of a SESAM scheme

Dynamic perspective – the rules

The dynamic behavior is represented by a set of rules. Each rule essentially describes how the state of the simulated project changes with time, or as the result of an action triggered by the project manager. Consider a simple example: The project leader asks a developer to write a document, perhaps the specification. One effect of this action is that as long as the developer is writing the document, the document grows at a speed depending on the developer´s experience. Assume further that the average writing productivity is 3 pages per day, and that experience is a multiplier with a value of less than one for a rookie, and greater than one for a very experienced team member. In SESAM this rule is described as shown in figure 3.2.

The graphical notation for SESAM rules is based on a combination of graph grammars and system dynamics. The basic idea is that the current state of the simulated project is represented by a graph to which graph grammar productions can be applied. Each SESAM rule performs

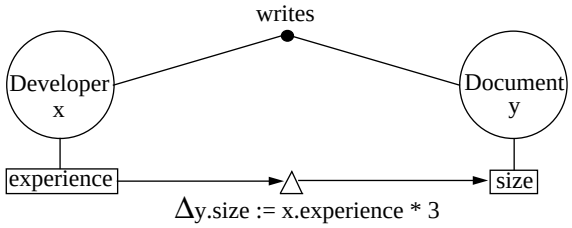

Figure 3.2: A simple SESAM rule

such a graph production, i.e. it specifies which subgraph must be matched so that the rule can be activated. The notation has been extended by system dynamics elements to reflect continuous changes of attribute values with simulation time (see attribute size in figure 3.2). The interaction between model and player is described by events. Rule notation and the formal foundations are discussed in more detail in [Drappa95].

Before simulation can begin, another model component has to be provided: the initial situation. This part essentially defines the initial graph to which the rules can be applied. The initial situation comprises the problem description of a specific project, and instances from the entity or relationship types defined in the scheme. Examples for objects that could be defined in this context are team members or tools that are individually characterized by their specific attribute values.

4. SESAM Simulation

Having formally described the software process, the simulation of a software project can begin. The user of the simulator (maybe a project management student) receives the project description and plans the project according to his abilities. The player then communicates with the system using a very simple interface.

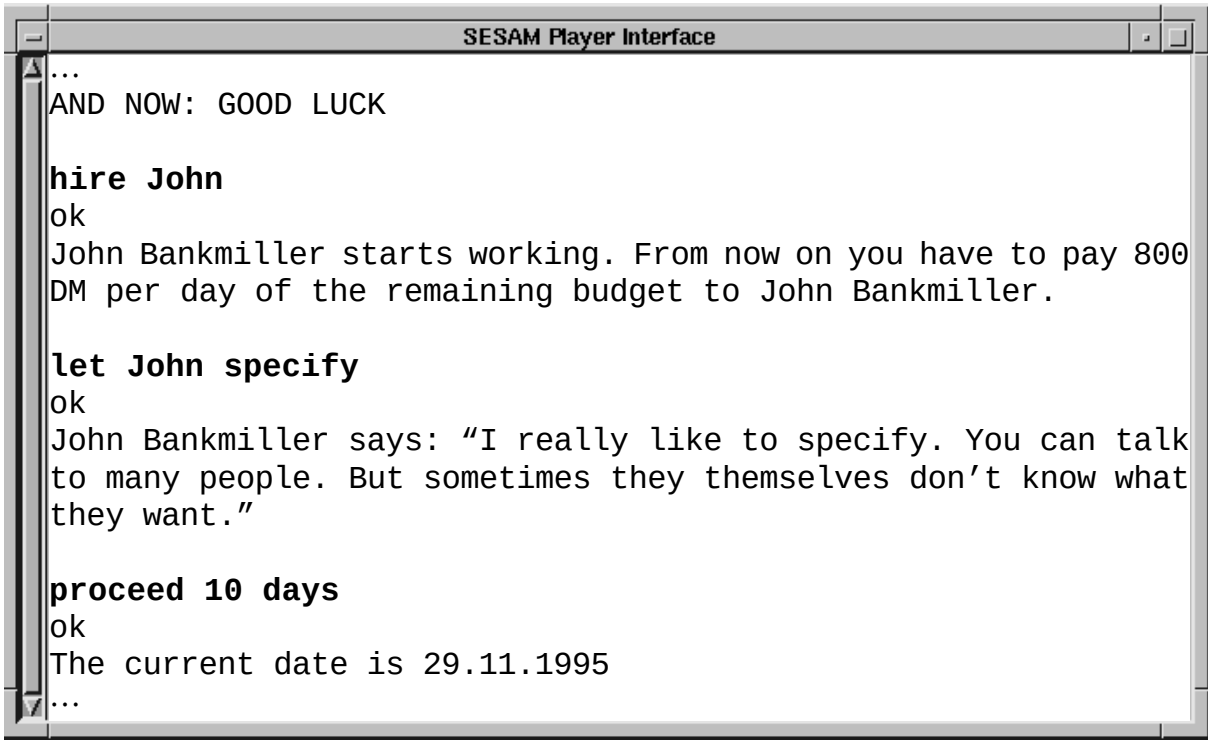

Figure 4.1: The player interface of the SESAM system

Either he or she triggers actions to drive the project in the right direction, e.g. hiring new people or assigning tasks to the team members, or he/she lets time proceed giving staff members a chance to perform their tasks. In return, the player gets reactions from the system. The example shown in figure 4.1 is now discussed in some more detail. The player commands are displayed in bold-typed letters whereas the system replies are in standard type-face. The first action, the player has decided to take is to "hire John". The system replies that "John Bankmiller starts working and that he costs 800 DM per day". After having hired John, the player assigns a task to him. John shall write the specification. He is delighted at this idea and says that he likes specifying very much. The third command shown differs from the previous two. Besides giving instructions and taking decisions, the player is also in charge of the simulation time. Thus he or she is able to determine the time needed for different project tasks.

As shown above, the player receives informal messages from the system depending on the actions taken. These messages mostly do not reveal any internal data. But they may give hints to the player. In the example above, the player can conclude that John Bankmiller is experienced in writing the requirements specification and that he might perform the task well. This reflects the real situation where it can be difficult for managers to get detailed knowledge of the capabilities of their staff members and of the progress of the project in general.

After having finished the simulated project, the course of the project can be analyzed. To facilitate this analysis, the SESAM system provides a so-called analysis component. This component is able to display the attribute values over time. Effects that occured during the project can be visualized and explained. Players learn which action has caused which effect. Subsequently, they can check if this effect has been a positive or a negative one with respect to the overall project. Figure 4.2 shows an example analysis of a simulated project. Assume that the progress of this project is measured in terms of recognized requirements. However, the features the developers have recognized and described in the specification document are not necessarily the same as those which the customer initially required. To deal with this problem, the player of this simulated project has decided to perform a review on the requirements specification. The curve shown in figure 4.2 reveals that the number of requirements is continually growing as long as the developers work on the specification. Then the document is given to the customer who should verify it. The customer tries to find inconsistencies and missing or obsolete functionality. As long as the customer reviews the specification, the number of requirements remains unchanged. When the customer has finished the review he reports about the findings, and the developers start to improve the document. Since the number of requirements decreases significantly, the customer has found much more obsolete than missing requirements.

The analysis component conceptually belongs to the simulation component. The modeling component and the simulation component, however, have been fully separated. This was an important design decision, since models can easily be adapted, extended, and analyzed. At the same time, the complexity of the simulation component is significantly reduced. The simulator mainly activates and deactivates the applicable rules according to the graph structure representing the state of the simulated project.

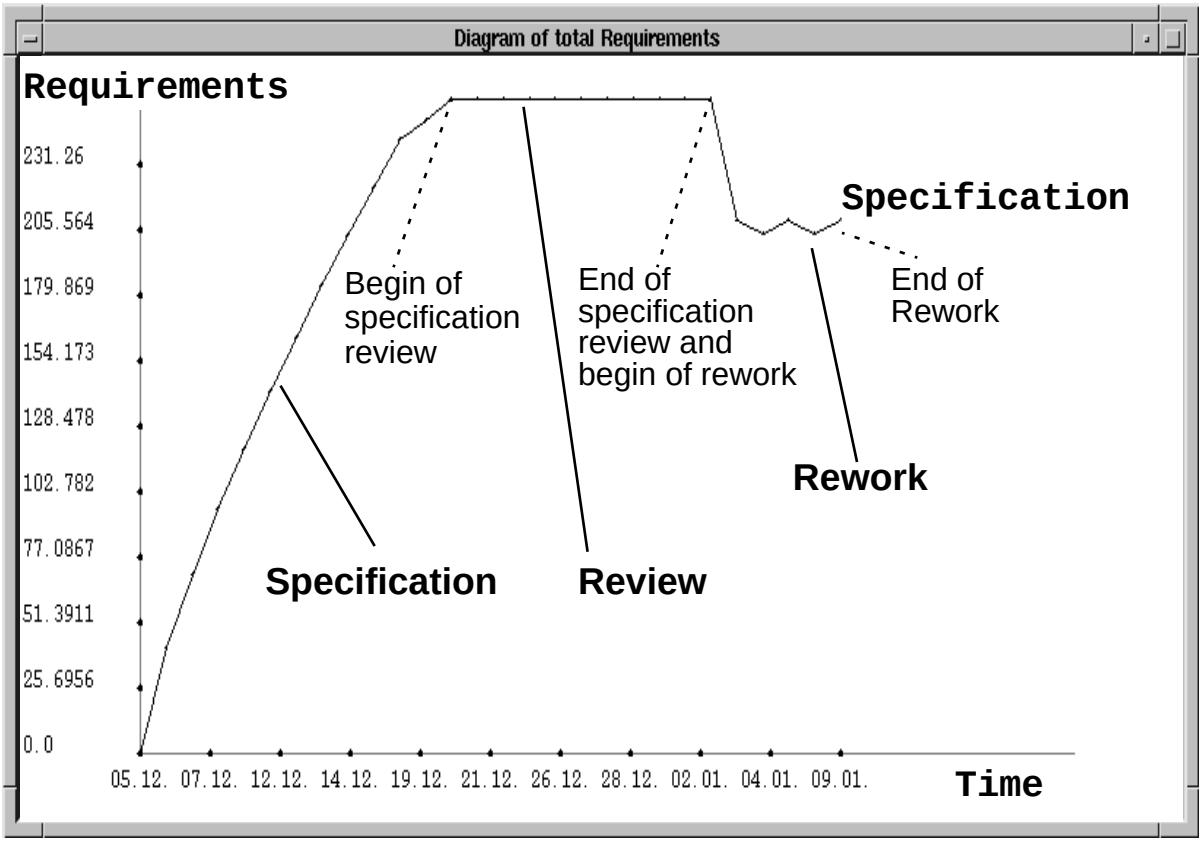

Figure 4.2: An example analysis of a simulated project

5. Experiences and Future Work

To test our modeling and simulation approach, we constructed a comprehensive model of a software process which comprises all phases of the software development lifecycle [Deininger94]. It consisted of a detailed scheme and approximately 100 rules. The model has been successfully applied in an experiment conducted at our department.

In this experiment five groups with two students each had to simulate a "check accountance" project. The experiment took twelve weeks. The students received the project information at the beginning. They had to plan and to manage the check accountance project according to their knowledge and skills. At the end, the success of the project, or its failure, respectively, was assessed using three measures: The needed time, the needed budget, and the conformance of the product with respect to the initial customer requirements. This last measure reflected how many customer requirements have been transmitted into the final product, and how many have been lost or introduced by mistake.

The results of the experiment were convincing. Students gained experiences similar to those real project managers report about. Some groups even felt panic when the simulated project got into some simulated trouble. By revealing their individual management strategies, the students gained deep insight into the software process in general and into the consequences of their individual decisions.

For research purposes however, the model proved to be less adequate. As explained above, the basic metaphor is to model product quality by a number of abstract units which initially represent customer requirements. These requirements have to be transferred from document to document to the final product. Every unit lost, or added without proper reason, decreases the quality of the emerging product. This measure appears too abstract and has no equivalent in reality, which makes it harder to validate the model. We are currently working on a new simulation model that is assembled from several modules. These modules help to reduce complexity of fine-grained models, and they can be analyzed and validated separately. The basic approach we pursue now is to collect quality aspects of software, and to identify suitable metrics to measure these quality aspects. It is important to choose metrics that can be applied to real documents in order to assure the possibility of comparing our models to real projects (the only way to validate the models). Our hope is that eventually, the models should be able to gradually replace the accumulation of rules of thumb now prevalent in the textbooks on software engineering.

References

[Abdel-Hamid91] Abdel-Hamid, T.K.; Madnick, S.E.: Software Project Dynamics: An Integrated Approach. Prentice Hall (Engelwood Cliffs), 1991.

[Carr90] Carr, D.; Koestler, R.: System Dynamics Models of Software Developments. Proceedings of the 5th International Software Process Workshop. 10. - 13. Oktober 1989, Kennebunkport, ME (USA), 1990, p. 46 - 48.

[Curtis92] Curtis, B.; Kellner, M.I.; Over, J.: Process Modeling. Communications of the ACM (35), Nr. 9, 1992, p. 75 - 90.

[Deininger94] Deininger, M.; Schneider, K.: Teaching Software Project Management by Simulation – Experiences with a Comprehensive Model. In: Díaz-Herrera, J. (Hrsg.): Software Engineering Education – 7th SEI CSEE Conference. San Antonio, Texas, USA, January 1994. Proceedings. Springer (Berlin), Lecture Notes in Computer Science No. 750, 1994, p. 227 - 242.

[Drappa95] Drappa, A.; Melchisedech, R.: The Use of Graph Grammar in a Software Engineering Education Tool. Proceedings of the Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation. 28. August - 01. September 1995, Volterra (Italien), 1995, p. 133 - 140.

[Finkelstein94] Finkelstein, A.; Kramer, J.; Nuseibeh, B. (Eds.): Software Process Modelling and Technology. John Wiley & Sons Inc. (New York), 1994.

[Levary91] Levary, R.R.; Lin, C.Y.: Modelling the Software Development Process using an Expert Simulation System Having Fuzzy Logic. Software - Practice and Experience (21/2), 1991, p. 133 - 148.

[Ludewig89] Ludewig, J.: Modelle der Software-Entwicklung – Abbilder oder Vorbilder? Softwaretechnik-Trends (9), No. 3, 1989, p. 1 - 12 (in german).

[Ludewig92] Ludewig, J.; Bassler, T.; Deininger, M.; Schneider, K.; Schwille, J.: SESAM – Simulating Software Projects. Proceedings of the 4th International Conference on Software Engineering and Knowledge Engineering. Juni, 1992, Capri (Italien), 1992, p. 608 - 615.

[Schneider94] Schneider, K.: Ausführbare Modelle der Software-Entwicklung. vdf Hochschulverlag (Zürich), 1994 (in german).

[Snowdon94] Snowdon, R.A.; Warboys, B.C.: An Introduction to Process-Centred Environments. In: Finkelstein, A.; Kramer, J.; Nuseibeh, B. (Hrsg.): Software Process Modelling and Technology. John Wiley & Sons Inc. (New York), 1994, p. 1 - 8.