Problems in Modeling the Software Development Process as an Adventure Game

Jochen Ludewig

Institut für Informatik, Universität Stuttgart

Summary

SESAM (Software Engineering Simulation by Animated Models) is a simulator for practicing the role of a software project manager. Its long term goal is to provide a tool for training CS students. As a research project, SESAM calls for an integrated model of the software development process, reflecting and quantifying many phenomena observed in real software projects.

We are currently using the second prototype, which can already demonstrate some rational behaviour. More important, however, were our observations in the process of constructing SESAM. They shed light on the current state of software engineering, and on the applicability of metrics.

SESAM is being developed in an evolutionary style by the Software Engineering Department (Lehrstuhl) at Stuttgart University; it is implemented in Smalltalk-80 on Unix-Workstations.

This paper concentrates on the fundamental questions raised by the work on SESAM. A more complete description of our work has been published before (Ludewig et al., 1992).

1 Software Process Modeling, and the Concept of SESAM

In the field of software process modeling, two very fundamental questions are still open:

1. How do the relevant factors (like staffing, skill, quality assurance, style of management, etc.) influence the results?
2. Which set of metrics is necessary and sufficient to describe the process, and the emerging product?

It seems that neither of these questions can be answered without the other one, so we try to combine them, and test various answers in an experimental, iterative way. That is the essence of SESAM. We try to build a simulator for education purposes, which can be described as a mixture of "Dungeon", or "Dark Castle", and a flight simulator, as used by future airline pilots. While we do not expect to finish a really useful simulator very soon, our work will continously contribute to the solution of the problems stated above.

Our goal is to provide a tool which can be used as follows:

A player (let's call her Angela) opens the SESAM-system, and is prompted for a number of project-parameters. Then she starts the simulation. The project proceeds like a normal project, with all kinds of difficulties. But Angela will receive only little

information about the actual state of her project as long as she remains passive. She may, however, decide to participate actively, e.g. she may see her employees and order them to do some particular task. Such activities consume her (simulated) time, so she cannot do everything she might like to do.

When the simulated project has been finished, her score is displayed, describing her relative success or failure by several indicators. She can run the project again, with all the (previously hidden) state variables displayed, which will help her to recognize her mistakes. She may also take over control again at some point, in order to try a different path.

Within a few hours, she has had an experience which would take months in reality, not to mention the costs, but would even then not allow for analysis, and a second try.

2 Problems in developing SESAM

After the idea of SESAM was born three years ago (Ludewig, 1989), there was a long period of confusion and irritation. The most difficult points are discussed below.

2.1 The Illusion of Detecting Natural Laws

Scientists, in particular physicists, have managed to reduce their theories to a comparatively small basic set of axioms. In software engineering, we have nothing but a large, and often inconsistent and inhomogenous collection of observations. We must be well aware of the fact that we cannot find any "natural laws of software engineering". Our very ambitious goal is to develop theories which can be used to predict observable phenomena.

2.2 The Lack of Quantitative Relationships

We are still far from being able to describe the software development process by, say, a set of differential equations. There is a number of less or (usually) more vague rules of thumb, of rumours, and of modern proverbs, which may explain certain phenomena. (And in most cases, there are others which explain the opposite.)

Assumed that Brook's law "adding manpower to a late project makes it even later" is correct, what does it say? It does not define a "late project," nor does it indicate to which extent the schedule is prolonged when people are added. It is nothing but the qualitative description of a relationship, very interesting (and fifteen years ago even surprising), but not sufficient for simulating the process, or predicting its success, or failure.

An interesting implication is that there are only very few proper theories in software engineering. COCOMO is one of the rare examples.

2.3 Choosing the Granularity

As we are building a model, the scale is critical. Which level is appropriate for simulating the process? Should we care about single persons, or regard groups as the acting units? Do we handle procedures, modules, or programs? By which factor should we shrink the time scale for simulation? Which granularity of time is required?

For our model, a day is chosen as the basic time unit. No effort is taken to care for precise synchronisation of people (e.g. in case of a meeting). Single persons are the only active units, though many "laws" (like Brook's law, e.g.) refer to groups, or projects. But many other relationships are obviously based on persons (like the cost of salaries, the responsibility for a certain task, or the possibility of intended or unintended changes in the project). If we had a mixed mode simulation, we must anyway convert all state variables to every level. Therefore, our first rule is: The whole is exactly the sum of its parts (including their relationships).

2.4 Hypotheses

As stated in the begin, most of the relationships on which our simulation is based are far from proven truths; they are just hypotheses which may, or may not make our simulator behave sensible. Therefore, we call such relationships hypotheses. While they define the very basic mechanisms of SESAM, they should be integrated in a way which makes it easy to modify, or to replace them.

We use graphs to describe the actual structure of the project; the graph may change when certain events take place, e.g. when a person is given a new task. Hypotheses are described by a graph grammar. Whenever a hypothesis may become applicable, the graph (and the state variables) is scanned for its target pattern. If it is found, the hypothesis docks into the target (like a virus), thus influencing the state transitions.

2.5 What is a Person?

When we simulate a person, we have to determine to which degree the personality should be mapped onto the model. Quite obviously, we have to record every person's abilities for each possible task (analysis, specification, etc.), and also for activities required in each task (like the abilities to communicate, or to take decisions). Other parameters certainly exist, e.g. a person's motivation has a strong influence on his or her productivity. We also have to describe how fast a person can learn, and how fast the person will forget.

But there is no line which separates the personal parameters relevant to the software project from those of purely private character. Every property may influence a person's performance, the distinction of a "public personality" from a "private personality" is fictitious.

3 A few Observations

1. Every system of metrics has to be based on exactly one basic unit in each dimension (e.g. hour, module, person or group or project).
2. When a metric is proposed, a new term is required. Terms which are widely used (e.g. "complexity") should not be used to identify a new function.
3. Even the most recent books on software engineering contain very little information which can be used to construct a project simulator, i.e. they contain very little falsifiable statements, but lots of noise. Boehm's whole work (starting from Boehm, 1973) is a notable exception.
4. If the player wants a single one-dimensional score, we have to map several ratings into one: adherence to deadlines, various aspects of quality, etc. In real projects, such a mapping is missing, though everybody has one in his or her mind. We believe it would be useful to agree upon the mapping in the very begin of every project, as part of the requirements definition.

References

Boehm, B.W. (1973):
Software and its impact: a quantitative assessment. **DATAMATION**, 19, No.5, 48–59.

Ludewig, J. (1989):
Modelle der Software-Entwicklung: Abbilder oder Vorbilder? (in German) **Softwaretechnik–Trends**, 9, 3 (Okt. 1989), 1–12.

Ludewig, J., Th. Bassler, M. Deininger, K. Schneider, J. Schwille (1992):
SESAM — Simulating software projects.**Proc. of the 4th Intern. Conf. on Software Engineering and Knowledge Engineering**. IEEE Comp. Soc. Press Order No. 2830, pp.608–615.