A Software Process Scheduling Simulator

Frank Padberg
Fakultät für Informatik
Universität Karlsruhe, Germany
padberg@ira.uka.de

1 Introduction

To cut development cost and meet tight deadlines in short staffed software projects, managers must optimize the project schedule. Scheduling a software project is extremely difficult, though, because the time needed to complete a software development activity is hard to estimate. Often, the completion of a task is delayed because of unanticipated rework caused by feedback between activities in the process.

In this research demo, we show how to use *process simulation* to support software project managers in scheduling. We present a discrete-time simulator tailored to software projects which explicitly takes a scheduling strategy as input. The simulator makes it easy to experiment with different scheduling strategies. It provides quick feedback about the impact that a particular strategy is likely to have on the progress and completion time of a project. A manager can compare different strategies and choose the strategy which he thinks is best for the next project's setting. In addition, a single step mode of the simulator allows a manager to view the dynamics of a project in detail.

The simulator is an implementation of the probabilistic scheduling model for software projects which we have presented earlier [6, 8]. The scheduling model describes the software process at a high level of abstraction: teams work on software components. Classical process phases such as coding or testing are not modeled explicitly. Still, the model captures much of the dynamics of software projects, representing varying staff skill levels, rework caused by design changes, component coupling, and changing task assignments.

Process simulation is an important technique to evaluate the impact of proposed process changes [5]. In particular, system dynamics simulations have been around for years, some of them addressing the problem of project staffing [1, 2]. By modeling individual teams and components as well as explicit task assignments, our model is much more fine-grained than system dynamics models. In addition, the particular way in which our model describes feedback between activities is novel in both software engineering and operations research [3, 7, 9].

2 Simulator

The simulator is written in the ModL language of the general-purpose simulation tool EXTEND [4]. The input data for the simulator are: the *base probabilities,* the *dependency degrees,* and the *scheduling strategy.*

For each team and component, there is a set of base probabilities which specify how likely it is that the team will need a prescribed amount of time to finish the component, report a high-level design problem, or finish reworking the component after a design change. In practice, the base probabilities must be computed from empirical data collected during previous projects.

The dependency degrees are a probabilistic measure for the strength of the coupling between the components and must be computed from the high-level design of the software. The stronger the coupling is the more likely it is that design changes which originate in one component will propagate to other components, thus leading to rework in these components.

The scheduling strategy (policy) specifies for each possible state of the project which team must work on which component. The scheduling strategy is implemented as a separate block in the simulator; thus, the strategy can be easily replaced. The state of the project includes the development time spent on each component so far, the project duration up to this point, the amount of rework left for each component, and the current task assignment. Plotters can be easily attached to the simulator to trace project state variables during the simulation.

Since the scheduling model is probabilistic, events will occur only with a certain probability at a particular point in time during the project. When the scheduling strategy is fixed, the simulator determines which step the project will take next by "throwing a dice". For example, some component might be finished in the next simulation step, or a design problem might get detected. The dice behaves according to the input probability distributions, taking into account the current state of the project.

Similarly, if a design change occurs in the project the simulator determines the set of those components which are affected by the change by throwing another dice which behaves according to the dependency degrees. This way, in each simulation run one possible full path of the project is simulated.

3 Example

As an example, we use the simulator to study the performance of different *list policies* for a small sample project. A list policy prescribes an order in which the components must be worked on; the next team to finish must work on the next component in the list. Since the completion time for an activity is subject to chance in the simulation, the actual schedule of the project depends on the progress of the project. List policies keep all teams busy during the whole project, which reflects management practice.

The sample project consists of four components and two teams. The base probabilities for the example are chosen in such a way that: team Two has a lower productivity than team One; components A and B have a similar complexity; components C and D require much more effort than components A and B. The dependency degrees are chosen to reflect that components C and D are strongly coupled.

In the sample project, using list policy ABCD will initially assign component A to team One and component B to team Two. Whichever team finishes its task first will work on component C. Finally, the next team to finish will work on component D.

We have implemented list scheduling as a single block where the list to be used is configurable. We ran 500 project simulations for each of the 24 possible list policies and observed the project completion time for each run. TABLE 1 shows the mean project completion time for each list policy.

The scheduling strategy has a significant impact on the completion time of the sample project. On average, list policy DCBA is best and CBAD is worst. List policy DCBA yields a schedule which on average is about 17 percent shorter than for CBAD.

From the histogram for the project completion time of each policy one can see that the *risk* for missing a tight deadline (say, 26) is much higher for CBAD than for DCBA. FIGURE 1 shows the histogram for policy DCBA. Using the full simulation traces, it is possible to give a detailed analysis of the performance of the different policies by studying the actual task assignments and the rework in the simulated projects, see [8].

Table 1. Mean completion times for the sample project under different scheduling strategies.

policy	mean	policy	mean	policy	mean
ABCD	30.2	BCAD	30.1	CDAB	27.0
ABDC	26.4	BCDA	26.1	CDBA	26.9
ACBD	30.3	BDAC	27.8	DABC	29.5
ACDB	26.5	BDCA	26.6	DACB	26.2
ADBC	27.8	CABD	30.9	DBAC	29.4
ADCB	27.2	CADB	28.4	DBCA	26.9
BACD	29.1	CBAD	**31.2**	DCAB	26.4
BADC	27.2	CBDA	29.2	DCBA	**25.7**

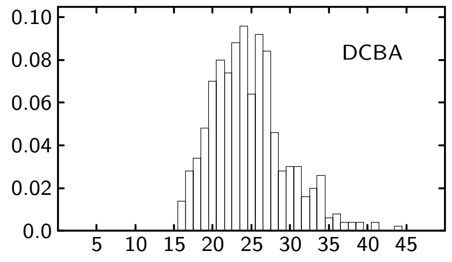

Figure 1. Histogram of the simulated project completion times for policy DCBA.

References

1. Abdel-Hamid, Madnick: *Software Project Dynamics*. Prentice Hall, 1991
2. Collofello e.a.: "A System Dynamics Simulator for Staffing Policies Decision Support", Annual Hawaii International Conference on System Sciences 31 (1998) 103–111
3. Derniame, Ali Kaba, Wastell: *Software Process: Principles, Methodology, and Technology*. Lecture Notes in Computer Science 1500, Springer 1999
4. EXTEND, `http://www.imaginethatinc.com/`
5. Kellner, Madachy, Raffo: "Software Process Simulation Modeling: Why? What? How?", Journal of Systems and Software 46 (1999) 91–105
6. Padberg: "Scheduling Software Projects to Minimize the Development Time and Cost with a Given Staff", Asia-Pacific Software Engineering Conference APSEC 8 (2001) 187–194
7. Padberg: "A Stochastic Scheduling Model for Software Projects", Dagstuhl Seminar on Scheduling in Computer and Manufacturing Systems, June 2002, Dagstuhl Report No. 343
8. Padberg: "Using Process Simulation to Compare Scheduling Strategies for Software Projects", Asia-Pacific Software Engineering Conference APSEC 9 (2002) 581–590
9. Weglarz: *Project Scheduling. Recent Models, Algorithms, and Applications*. Kluwer, 1999