

# Towards Better Schedules for Software Projects

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

## 1 Motivation

Staff is a limited resource in a software project. As a consequence, project managers must perform extensive scheduling. They must assign tasks to the development teams and prescribe an order in which the tasks must be worked on. Scheduling a software project is difficult for a number of reasons. For example,

- tasks depend on each other ;
- certain staff is available only at particular times ;
- the effort required for a task is not precisely known ;
- teams often interact in unexpected ways.

All scheduling aims at meeting a given deadline and budget.

Managers would certainly welcome advice from us researchers about how to find good scheduling strategies. Yet the goal is *not* to equip a manager with an optimal scheduling strategy in the form of a huge decision table which tells him what to do next if the current project status is such and such – a manager simply won't use that. Managers will trust a strategy only if it comes in the form of a rule which they understand and which makes sense to them. It may not be possible to find a strategy which is both rule-based and optimal, so we trade optimality for acceptance here. In addition, it seems likely that how good a particular strategy really is depends on the project under consideration. Therefore, the goal is to equip a manager with a tool for comparing scheduling strategies with respect to time and cost in different project settings. In particular, any strategy should be compared against an optimal one to see how much room there is for improvement. Such a tool could help a manager select the most promising strategy for his next project.

## 2 Modelling

If we want to compare scheduling strategies and find optimal ones, we must be able to quantify the impact of the scheduling strategy on the development time and cost of a project. Therefore, we must have a project estimation model which *explicitly* takes the scheduling strategy as input. Since events occur only with a certain probability at a particular time during a project, the model also must be probabilistic. As a result, an optimal strategy will be *stochastically* optimal.

---

\*supported by a postdoctoral fellowship of the Deutsche Forschungsgemeinschaft DFG at the Graduiertenkolleg Informatik, Karlsruhe

Scheduling is studied extensively in operations research. Many models studied there are deterministic, dealing with scheduling in manufacturing processes, see [1]. The probabilistic models studied there are not what we need, either, because they fail to model general interaction of concurrent process threads as it is typical for teams working simultaneously in a software project, see [2].

In [3], I have presented a dynamic and probabilistic model for software projects. That model can be extended to explicitly take a scheduling strategy as input. Here is a sketch of the new, extended model.

**Time.** Time is discrete in the model. Think of a time slice as corresponding to, say, a week in real time. In addition, for each project there is a deadline for completing the project. If the deadline is exceeded, the project will be cancelled as a failure.

**Phases.** In the model, a project advances through a sequence of phases. By definition, a phase ends when some team completes its component or when a revised version of the software's high-level design is ready. Revising the design might be necessary to fix design errors or because of changes in the requirements. Scheduling decisions take place only at the end of a phase. At that time, staff becomes available again, or re-scheduling might be appropriate because of the design changes. For example, it might be necessary to re-allocate a team to rework a central component that had already been completed before.

**States.** The state of a project changes at the end of each phase. The state of a project by definition consists of three parts :

- a progress vector ;
- an activation vector ;
- a countdown number.

The progress vector has one entry for each component. The entry for a component in the progress vector is defined as the net development time that has been spent working on the component. The net development time is obtained from the total development time by subtracting all rework times spent on that component with fixing high-level design problems. The activation vector has one entry for each component, too. The entry for a component in the activation vector equals one if some team is currently working on the component and zero otherwise. The countdown number is the time left until the deadline will be reached.

**Strategies.** A scheduling strategy is a function which (deterministically) yields an activation vector. The strategy function is applied as the last step during a phase, taking as input :

- the state of the project at the end of the previous phase ;
- a record of what happened at what time during the current phase ;
- a setback vector which specifies for each component how much rework time must be spent on that component because of the latest design changes.

Events that get recorded during a phase are teams reporting that they have completed their component and teams reporting that they have detected a problem with the high-level design. The record of a phase includes the length of the phase. Comparing an activation vector to the previous one shows which components get switched on or off.

The scheduling strategy implicitly built into the original model described in [3] is simple. There are as many teams as components, the teams start working at the same time, and the teams continue to work until all components are completed. Scheduling is more exciting if the number of available teams is less than the number of components.

**Transitions.** The input taken by the strategy function also suffices to compute a new progress vector for the project at the end of a phase. The previous countdown gets decreased by the length of the current phase to get the new countdown. Supplementing the new activation vector returned by the scheduling function yields the new state of the project. Once a scheduling strategy is fixed, the course of a project thus can be described by the sequence of states that the project passes through at the end of its phases.

**Probabilities.** To arrive at a probabilistic model, define the transition probability  $P_{\zeta}(\eta)$  to be the probability for ending the current phase with state  $\eta$  given that the previous phase ended with state  $\zeta$ . To compute the transition probabilities the following input data are required :

- the scheduling strategy;
- statistical data about the courses of past projects;
- high-level design data.

The design data are a measure for the strength of the coupling between the software's components. The stronger the coupling is the more likely it is that design changes originating in a particular component will propagate to other components. For example, when an interface offered by some component gets extended, all components which use that interface must be reworked. The statistical data are probability distributions specifying for each team how likely it is that the team will complete its component, report a design problem, or fix a design error after having worked on the component for a particular time. The statistical data get computed from empirical data collected during past projects. The empirical input data are similar to the data required for the original model in [3].

The resulting probabilistic model is a Markov chain. It is well-known how to compute probability distributions for a project's development time and cost using the transition probabilities.

### 3 Research

I'm currently developing a computer program for the model described above. The hard part is to compute an optimal scheduling strategy for a given project setting. First runs will be available soon. Using the program, a number of interesting research questions can be tackled, including :

- How far away from the (stochastic) optimum are the strategies that managers use in real projects?
- Should we develop components which are strongly coupled to several other components early or late in a project?
- Does the model reflect the results obtained by other researchers about schedule compression?

### References

1. Blazewicz, Ecker, Pesch, Schmidt, Weglarz : *Scheduling Computer and Manufacturing Processes*, Springer 1996
2. Neumann : *Stochastic Project Networks*, Springer Lecture Notes in Economics and Mathematical Systems 344 (1990)
3. Padberg : " A Probabilistic Model for Software Projects " Proceedings ESEC/ FSE 7 (1999) 109 – 126