Linking Software Design with Business Requirements – Quantitatively

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

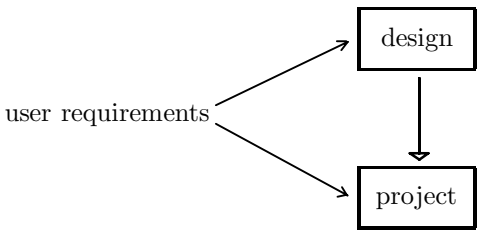
Abstract

If we want to quantify the impact of our design decisions on the delivery date and cost of our projects, we must have a dynamic and probabilistic model for software projects which *explicitly* takes the design as input. Such a model is presented in the paper. In the model, the strength of the coupling between a software's components determines which components have to be reworked when a design problem occurs. The stronger the coupling is the more likely it is that design problems will propagate from the originating component to other components. An example shows how strong the impact of the coupling on the development time can be.

1 Design and Project

The high-level design of a software product has a strong impact on the software's development and maintenance projects. For example, during development each component in the design determines a set of project tasks, and the knowledge and experience required for developing a particular component is a major factor when assigning tasks to the developers. The design is built around the requirements of the users concerning the functionality, reliability and useability of the software. From the user perspective, a design is good if the resulting software meets the requirements.

In some cases, the users' requirements get reflected in the project in a way which can't be seen in the design. For example, though the user interface component is part of the design, testing prototypes of the user interface early during development with the participation of the users, which aims at achieving the required useability, can't be seen in the design.

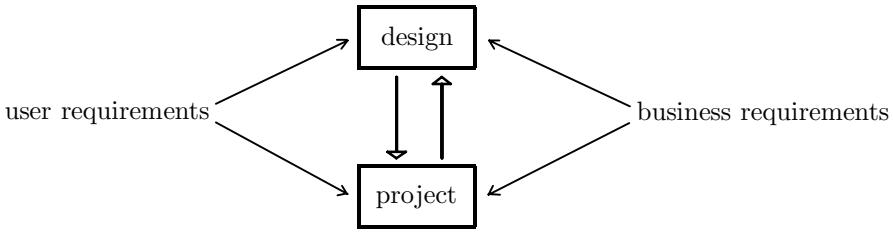


The framework for a software's development and maintenance projects is formed by business requirements. A project must be manageable, there are budget and time limits, and there is only a certain

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

staff available for a project. The business requirements for the projects have a strong impact on the design of the software. For example, to support coordination of the work the design of the software must be structured and the interfaces between the components clearly described. The need to bring out new releases of the software in the future, again with budget, time, and staff limitations, leads to a number of maintainability requirements for the design. To achieve a high degree of reuse in order to support lower cost and faster delivery, the design must be taylored for reuse. Note that reuse also aims at higher reliability of the software, so some quality attributes support both user and business needs. From the business perspective, a design is good if the corresponding development and maintenance projects deliver the software on time and within budget.

In some cases, the business requirements get reflected in the project in a way which can't be seen in the design. For example, choosing a particular schedule because of limited staff availability, or using programming tools to increase productivity can't be seen in the design.



The link between the high-level design of a software and the corresponding development and maintenance projects is not static. During a project, it will become necessary from time to time to revise the design. For example, there might occur a design error which must be fixed, the user requirements might change, or unexpected staff turnover might make it necessary to buy a component instead of developing it and adjust the design accordingly.

When a design change occurs some of the components must be reworked, so their completion is delayed. The closer the components are coupled the more likely it is that changes originating from one 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 best case would be a design with no coupling between the components at all, so changes would remain local and the teams could work independently. The worst case would be a design where the components are coupled so closely that any design change would affect each of the components. How the components are coupled is described by the design. This way the design has another, less obvious impact on the progress of a project. From the dynamics perspective, a design is good if the corresponding projects run smoothly with little interference between the teams.

The impact of a design change need not be limited to reworking some of the components. A change can make some of the components much more complicated or even introduce additional components into the design, causing a re-structuring of the whole project. The times at which design changes will occur during a project and how much impact on the progress of the project the changes will have is hard to foresee when a project gets planned. This contributes a lot to the uncertainty inherent to software projects.

2 Quantitative Modelling

As described above, there is a strong link between the high-level design of a software product on the one hand and the business requirements for the corresponding development and maintenance projects on the other. If we want to arrive at economically founded practical guidelines for designing software, we must have a model for software projects with the following properties.

– 1 – The model must be quantitative. Managers need numbers to base their decisions on.

– 2 – *The model must explicitly take the design of the software as input. We can study a design decision only if it gets reflected in the model.*

– 3 – The model must describe the dynamics of software projects. The delivery date and cost depend on what happens at what time during the project.

– 4 – The model must be probabilistic. Because of the uncertainty inherent to a project, events occur only with a certain probability at a particular time during the project.

Once we have a model with the required properties at our disposal, we can use it to support our design decisions at any time during a project in a "what-if" manner:

- for each of the design alternatives, compute the expected (remaining) development time and cost using the model;
- then perform a cost-benefit analysis of the design alternatives, based on the model output and using concepts from corporate finance such as net present value or return on investment;
- finally choose the design alternative which best meets the business needs.

The first EDSER workshop included several papers about using standard concepts from economics in a software engineering context. For example, Erdogmus [3] studies the impact of rapid development on the net present value of a project. Raffo, Settle, and Harrison [8] study the expected return on investment of adding unit testing to a project. Erdogmus [4] studies how option theory can be used to assess the financial value of flexibility built into a design. Butler, Chalasani, Jha, Raz, and Shaw [2] study how portfolio analysis might be used to allocate a fixed amount of budget among several development techniques. Singer [9] reports that the way in which a product's architecture is managed should depend on the market stage of the software.

There already has been progress in establishing a model with the required properties. Abdel-Hamid and Madnick [1] take a system dynamics approach. They focus on modelling the feedback loops between productivity, workforce, and schedule in a project. The Abdel-Hamid and Madnick model is not probabilistic. The design enters the model only indirectly through some scalar model parameters such as "potential productivity" and "perceived project size". Tvedt and Collofello [10] expand the Abdel-Hamid and Madnick model to study the effectiveness of software inspections. Madachy [5] presents another derived version of the Abdel-Hamid and Madnick model to study inspection-based processes. He mentions that varying his "inspection efficiency" input parameter through simulation according to some probability distribution yields probability distributions for the cost and delivery date as output.

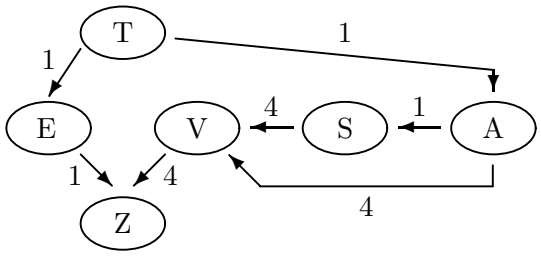
Raffo and Kellner [7] use statecharts to model a waterfall software process. Their model is probabilistic, taking probability distributions for key model parameters such as "productivity" as input. Simulating the model according to the input distributions yields probability distributions for the project effort and duration as output. The design enters the Raffo and Kellner model only indirectly through some scalar model parameters such as "source code size" and "code complexity".

In [6], using a much different approach, I have developed a dynamic and probabilistic software project model which explicitly takes the design of the software as input. The model is not tailored to a specific software development method.

– 1 – The model is quantitative, providing estimates for the development time and the cost.

– 2 – The design explicitly enters the model in several ways.

- The number of teams in the project equals the number of components in the design.
- A project advances through a sequence of phases, where a phase is defined as the time span between two consecutive (re-) designs of the software.
- The strength of the coupling between the components determines which components have to be reworked when a design problem occurs, depending on where the problem occured. For example, suppose that this diagram shows the components in a software design.



The number at the arrow between two components x and y in the diagram is the number of interfaces provided by y which are used by x. From the number of interfaces, the probability that a problem with a particular component (say, V) causes rework in a particular set of components (say, V, S, and A) can be computed (41.4 percent in the example).

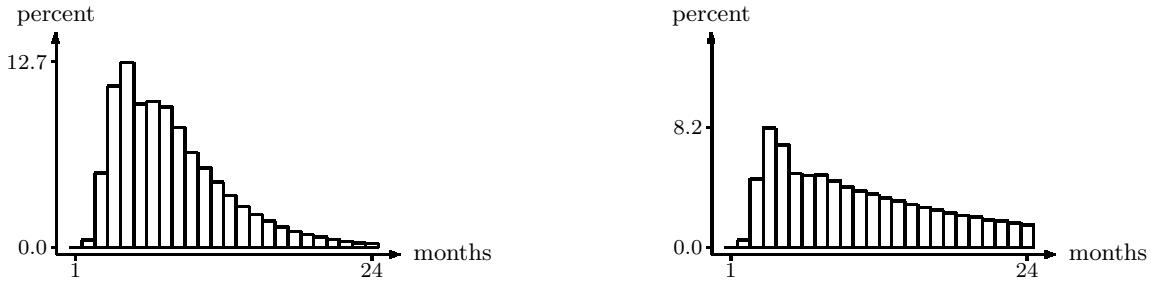
- The complexity of a component is used to select the appropriate productivity distribution for the team working on the component, see number 4 below.

– 3 – The model describes the dynamics of a project. The progress of a team depends on the progress of the other teams, since problems with the design which are detected by one team may cause rework for other teams. The state of the project changes whenever the design of the software changes.

– 4 – The model is probabilistic. For each team, it takes a probabilistic productivity distribution as input. The distribution is a measure of how fast the team has finished its components in earlier projects. The distribution depends on the knowledge and experience of the team, as well as on the complexity of the component. The productivity distributions get computed from empirical data collected during past projects. The strength of the coupling between the components enters the model in a probabilistic way, too, see number 2 above. The model yields a probability distribution for the project completion time as output.

Given the distribution of completion time for a project, it is straightforward to compute an estimate for the risk of exceeding a given deadline, see [6]. The risk of exceeding a given budget limit gets computed similarly. Quantifying the risk of exceeding deadlines or budget limits is a major advantage of probabilistic project models.

To illustrate how the model in [6] reflects the impact of the design on the expected delivery date of a project, consider two projects. The projects have the same number of components and the same team productivity distribution for each component. The projects differ only in the strength of the coupling between the components. The coupling is minimal for the first project and maximal for the second. Minimal coupling means that no component has to be reworked in case of a design problem except for the component where the problem was detected. Maximal coupling means that all components have to be reworked in case of a design problem no matter where the problem occured. Here are the resulting probability distributions for the completion time of the two projects.



The chart to the left corresponds to minimal coupling, the chart to the right to maximal coupling. The expected values for the development time are 8.6 months for the first project and 13.5 months for the second. In the model, the stronger coupling thus leads to a more than fifty percent higher estimate for the second project. Minimal coupling will not occur in a "real world" design and maximal coupling should not, but the example makes clear how strong the impact of the coupling can be.

3 Research

There is a variety of topics left for further research. Two topics that I am working on, using the model described above, are:

- Time to Market. The software industry becomes more and more competitive. Can we deliver faster if we schedule a project based on the coupling between the components? For example, should we develop components which are strongly coupled to several other components early or late in a project? Is a design with weaker overall coupling between the components preferable even if it has more components?
- Object-Oriented Development. The object-oriented paradigm leads to designs which are much different from conventional designs. How strong is the coupling in object-oriented designs? How do the team productivity distributions look like? How does the use of design patterns get reflected in the structure and dynamics of a project?

A dynamic and probabilistic project model which explicitly takes the design as input is just the right tool to study such questions quantitatively.

References

1. Abdel-Hamid, Madnick : *Software Project Dynamics*, Prentice Hall 1991
2. Butler, Chalasani, Jha, Raz, Shaw : " The Potential of Portfolio Analysis in Guiding Software Decisions ", EDSER-1
3. Erdogmus : " Comparative Evaluation of Software Development Strategies Based on Net Present Value ", EDSER-1
4. Erdogmus : " Valuation of Complex Options in Software Development ", EDSER-1
5. Madachy : " System Dynamics Modeling of an Inspection-Based Process "
   Proceedings ICSE 18 (1996) 376-386
6. Padberg : " A Probabilistic Model for Software Projects "
   Proceedings ESEC/ FSE 7 (1999) 108-126
7. Raffo, Kellner : " Evaluating the Performance of Alternative Software Processes Quantitatively "
   Software Process Newsletter (1994) 10-12
8. Raffo, Settle, Harrison : " Estimating the Financial Benefit and Risk Associated with Process Changes ", EDSER-1
9. Singer : " Optimizing Economic Performance in Commercial Software Development Organizations ", EDSER-1
10. Tvedt, Collofello : " Evaluating the Effectiveness of Process Improvements on Software Development Cycle Time via System Dynamics Modeling "
    Proceedings COMPSAC (1995) 318-325