The Economic Impact of Software Process Variations*

Florian Deissenboeck and Markus Pizka

Institut für Informatik, Technische Universität München
Boltzmannstr. 3, D-85748 Garching b. München, Germany
{deissenb,pizka}@in.tum.de

Abstract. The economic benefit of a certain development process or particular activity is usually unknown and indeed hard to predict. However, the cost-effectiveness of process improvements is of paramount importance and the question how profitable certain activities are needs to be answered. Within a large-scale commercial organization, we were challanged with the task to quantify the *economic* benefit of isolated test and development environments. To answer this question we defined a generic process model based on absorbing Markov chains that allows to analyze the economic benefit of software process variations. This model exposes conflicts between process steps and reiterations of development activities and thereby provides a highly flexible tool for the investigation of the effects of changes to a development process on its overall performance. This model was used to predict the impact of isolated testing on the overall effort and duration of projects at BMW. The results obtained correspond well with the perception of experienced developers and gives a detailed explanation for the effects. Besides this, it can be used to analyze various other economic aspects of software development processes and yields an interesting alternative for cost estimation.

Keywords: Software Process Economics, Process Simulation, Industrial Application, Absorbing Markov Chains.

1 Software Process Economics

How does one determine the economic impact of selecting a certain process model? Is XP cheaper than RUP? What are the risks of the waterfall model? Does the spiral model actually yield faster time-to-system? All of these questions are of practical relevance, highly important but very difficult to answer. Surely, in a specific situation we assume that alternative A is faster (cheaper, better, ...) than B based on our invidual experience but the benefit can neither be quantified nor guaranteed.

In contrast to this, the costs of activities are usually clear and precisely documented in bills. For example, it is unclear how much money can be saved in

* Part of this work was sponsored by the BMW Group.

Q. Wang, D. Pfahl, and D.M. Raffo (Eds.): ICSP 2007, LNCS 4470, pp. 259–271, 2007.

a given project setting by spending one additional dollar on model-based development techniques. The same applies for well-established activities like documentation as well as for more specialized methods like requirements engineering with formal methods.

1.1 The Value of Isolated Testing

Within a large scale industrial organization, we were challenged with the task to determine how much time and effort is saved by using isolated test and development environments for IBM mainframe (i. e. PL/I, COBOL) based commercial software projects.

While isolated testing is rather straight forward for UNIX and Microsoft Windows based software projects it is non-standard for mainframe applications since all projects share the same machine with the same infrastructure without having private copies of libraries, databases and so on. To achieve some kind of isloation most IT organizations that develop and maintain mainframe applications create some kind of software solution that enables them to develop and test multiple projects simultaneously in separated *environments* on a single mainframe.

The costs of this solution are usually easy to determine by adding space, CPU time, software licenses and support personnel. However, although it can be argued in a qualitative manner that separate testing and avoiding conflicts is useful, it is hard to quantify the *benefit*. In practice this makes it very hard to argue in favor (or against) such measures and consequently leads to decisions that lack an economically justified basis.

1.2 Approach, Contribution and Outline

Starting from our project partner's concrete questions about the economic benefit of process variations, we formulated a precise research question (Sec. 2) and investigated different approaches to answer it.

Due to a number of reasons (detailed in Sec. 6) we found that an empirical study could not satisfactorily answer this question and therefore developed new concepts to evaluate the economic effect of decisions regarding process variations (Sec. 2). These concepts are based on a probabilistic process model that uses absorbing Markov chains for the process simulation. This model advances existing process models as it renders project risks explicit and precisely describes reiterations of activities (Sec. 3). This model can be used to derive quantitative information on the cost and benefit of specific process activities.

We illustrate this with a study carried out for the BMW Group to determine the economic benefit of isolated test and development environments on mainframes (Sec. 4 and 5). We explain how our approach extends previous work (Sec. 6) and illustrate how the scope of application of the analytical model can be further broadened (Sec. 7).

2 Requirements / Situation

The object of investigation of our study was the development and test processes used by BMW Group's mainframe software development division. At BMW, several 100 software engineers develop and maintain critical business information systems with a total of 85 millions lines of PL/I and COBOL code. The division uses two separated IBM zSeries mainframes for development and operation, whereas our study focused exclusively on the development mainframe.

2.1 Mainframe Software Development

Unlike the more common workstation-based development environments, mainframes do in general not provide developers with isolated environments where they can edit, compile, link and test the code they are working on without interfering with other projects. In fact, if no additional measures are taken, all developers share the same development environment and all test data.

Due to the frequent separation of development and operation spaces of typical mainframe installation this does not pose any problems for the operation of the software, but creates servere problems for the concurrent development and test of multiple projects. Conflicts between projects can occur during almost all activities (e. g. compile, link, test) and affect almost all development artifacts (e. g. source code, libraries, test data). These conflicts are not only frustrating and time-consuming for the developers, but make sound testing almost impossible as test results can not be interpreted properly. For example, if a test case fails, it is not decidable whether it failed because of a bug or because another project changed the test data in the shared data base.

Unfortunately, isolated test spaces cannot be established for mainframes as easily as in ordinary workstation-based environments where every developer can have his own test space on an own workstation.

2.2 The CAP Isolation Mechanism

The BMW Group developed a software-based isolation technique on top of the virtualization mechanism provided by the mainframe.[1] This technique offers projects isolated test and development environments called Caps *(capsules)*. These Caps contain a complete copy of the required development environment including compilers, linkers, job control, and test databases. They thereby enable projects to develop and test in an independent, conflict-free manner until they reach a certain degree of maturity and can be integrated in the main development trunk in a special integration test phase. Caps have the additional advantage of making it easy to *reset* the complete development environment of a project to a specific state.

These advantages, however, come at a price as the initialization, operation and support of a Cap is a non-trivial task that demands significant hardware resources as well as expenses for dedicated personnel.

[1] IBM zSeries mainframes provide a coarse-grained virtualization mechanism.

2.3 Research Question

The qualitative benefit of a CAP can be explained quite easily by explaining how non-isolated development environments create expensive conflicts and contribute to poor product quality due to unreliable test results. It is, however, very hard to compare these qualitative benefits to the known quantitative costs of the CAP mechanism. Therefore the research question of the study we conducted was:

What is the economic benefit of using a CAP *for a software project?*

Note, that although this initial questions focuses on project effort, our study also analyzed the project duration to characterize the crucial time-to-system aspect. However, we cannot report on this in detail due to space constraints.

3 A Probabilistic Process Analysis Model

As explained in Sec. 6, we are convinced that is not feasible to answer the above questions on a quantitative scale by carrying out an empirical study. We therefore opted for an analytical model that abstracts from the problem under investigation and allows us to focus on the impact of CAPs on development effort and time.

This model was inspired by an observation of the analogy between software development processes and concurrent systems theory [1]. Development *activities* are similar to *tasks* executed by an operating system. In a development process the *resources* are not memory and file handles but source code, libraries and test data. Similar to the conflict that arises from a write access to the same memory address in a parallel system, a concurrent change to a program by two different projects produces a conflict in the software development process.

3.1 Probabilities and Risks

These considerations lead to a probabilistic process model that describes a development process as a system of concurrently executing tasks. The tasks of the system are the *activities* of the software process and the processors are humans (developers) executing these activities. Due to the goal of the overall process and limited resources, there are constraints on the order of the activities entailing the need for coordination. The transitions from one activity to possible succeeding activities are labeled with probabilities. Through this, there may also be *loops* in the parallel deterministic automaton describing costly rework in the development process due to failure or incompleteness at a certain stage of the process. The activities and the frequency of there execution define the cost and the duration of the project.

Fig. 1a shows a model of a simplified software process with the typical activities and transitions between them. Unlike other process models this model explicitly describes the loops (cycles) realistically found in software projects. This enables us to e. g. model the alternation between the activities *Implementation* and *Unit Test* that takes place in practice: Developers write some code,

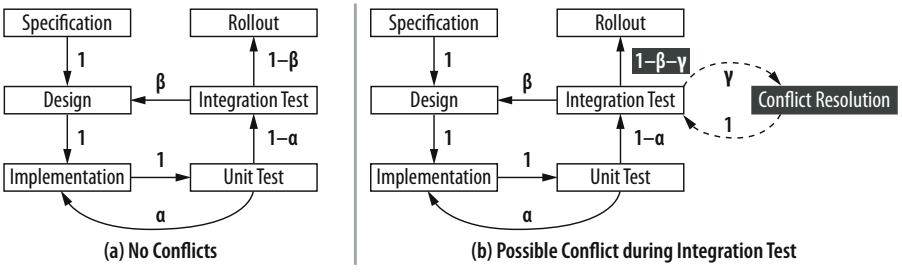

Fig. 1. Example Processes

test it (either manually or automatically) and then go back to implementing more code and/or fix existing code. They do so until they are eventually done with the implementation and all their tests pass. In addition to that, loops allow us to explicitly capture prevalent project risks that are often ignored [2]; e. g. an unlikely, but still possible, transition from the *Integration Test* to the *Specification* could be easily added to the process model. Note that the sum of the probabilities of the outgoing transitions of an activity must always be one.

Fig. 1b illustrates how resource conflicts during specific activities can be elegantly expressed through additional conflict-specific activities and adjusting the transition probabilities accordingly. For example, a conflict with another project during the *Integration Test* does not only reduce the probability that the project can proceed with the activity *Rollout* but requires the execution of the additional activity *Conflict Resolution*.

3.2 Operationalization of the Model

While this model provides an interesting abstraction of a software development process, it is does not answer the question about the benefits stated above, yet. Fortunately stochastics can help here as the process model can be viewed as a stochastic process or, more precisely, as a *discrete Markov chain* with an *absorbing state*.

A Markov chain is defined as a stochastic process with a set of states $S = \{s_1, s_2, \ldots, s_r\}$. The process starts in one of these states and moves stepwise from state to state. If the chain is in state s_i, then it moves to state s_j at the next step with a probability denoted by p_{ij}. This probability does not depend on the state history of the chain [3].

Markov chains are typicalyl represented as directed graphs very similar to the ones in Fig. 1 or as a transition matrix P that denotes the transition probabilities for every state. Working with this matrix, Markov chain theory provides powerful methods to compute a number of interesting properties of the chains. It is, for example, easy to calculate in which state the chain is expected to be after n steps when started in state s_i, or to determine the probability for moving from state s_j to state s_l in k steps.

When modeling a software process there must be an activity that does not have any transitions to other activities and thereby marks the end of the process (*Rollout* in Fig. 1). Translated to a Markov chain model this is equivalent to a terminal state s_i that has exactly one outgoing transition to itself with the probability $p_{ii} = 1$, Such a state is called an *absorbing state* and Markov chains with an absorbing state are called *absorbing Markov chains* [3].

Absorbing Markov chains are a powerful tool for analyzing processes as they provide well defined methods to determine

- the expected total number of steps until the chain reaches an absorbing state as well as
- to calculate the expected number of steps spent in each state.

Without going into the mathematical details we illustrate this for the process shown in Fig. 1a.

For the sample probabilities $\alpha = 0.95$ and $\beta = 0.2$ the absorbing Markov chain analysis yields the following expected number of visits to each state (start state *Specification*): *Specification* is expected to be carried out only once, *Design* and *Integration Test* are expected to be performed 1.25 times, and *Implementation* and *Unit Test* 25 times. The total number of steps before the chain reaches the absorbing state *Rollout* is given by the sum which is 53.5.

Figure 2 shows how different values for the probabilities α and β influence the expected total number of steps in the example process. While values close to 1 lead to an infinite number of steps in both cases, one can see that increasing β raises the number of steps stronger than increasing α as this transition occurs *later* in the process.

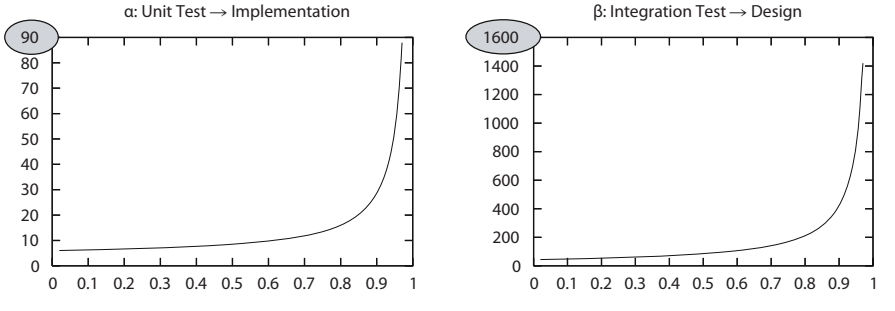

Fig. 2. Transition Probability vs Expected Total Number of Steps

3.3 Total Project Effort and Duration

The expected total number of steps represents a measure for project progress, but it still does not yet fully answer the questions about the total project effort and duration. To achieve this each process activity a is now associated with the average effort $\mathsf{eff}(a)$ and time $\mathsf{time}(a)$ needed for a single execution of the activity. The total effort and duration of a project is given by:

$$\mathsf{eff}_t = \sum_{a \in A} \mathsf{eff}(a) \cdot \mathsf{steps}(a) \qquad \mathsf{time}_t = \sum_{a \in A} \mathsf{time}(a) \cdot \mathsf{steps}(a)$$

where A is the set of all activities and $\mathsf{steps}(a)$ is the expected number of visits to activity a. Note that $\mathsf{eff}(a)$ and $\mathsf{time}(a)$ dependend on the project size.

4 Application of the Analysis Model to Isolated Testing

To apply our approach to analyze the economic benefit of isolated test and development at BMW, three fundamental pieces of information are needed:

1. transition probabilities
2. effort needed to execute for each activity
3. time needed to execution for each activity

As it is not realistic to *correctly* determine this information without investing considering empirical studies, we analyzed the two process variations (Cap and Non-Cap) in a relative manner. We therefore designed a *reference process*, calibrated it with existing empirical data and parameterized it with the probability for conflicts during development and test. Based on this reference process we designed the process models for Cap and Non-Cap development and compared them using the method presented above. This comparative approach allowed us to abstract from concrete values for the transition probabilities as well as the efforts and times needed for each activity.

4.1 Reference Process

Based on existing process descriptions and interviews with project managers as well as developers, we created the reference process model with 13 activities and 18 transitions (not presented here in its entirety due to confidentiality reasons). This model does not contain special isolation-related activities and therefore consists of the usual specification, design, implementation and test activities. It does, however, carefully distinguish between module tests and two levels of integration tests and contains explicit error analysis activities.

Eleven of the 18 transition of the model have a transition probability unequal one. Using existing process analysis data as well as interviews we estimated the probabilities and ensured that the remaining impreciseness does not bias our study results (see Sec. 5).

4.2 Calibration

To determine the effort needed for each execution of the activities, we calibrated the reference process with data from well-known empirical studies.

For example, the Markov chain analysis showed that the activity *Implementation* will be carried out 95.24 times and thereby accounts for 36.78% of the expected total 258.95 process steps. As [4] and other sources point out that implementation usually accounts for $\approx$ 20% of the total development effort, we concluded that the relative effort of each execution of *Implementation* activity in our process is 0.21%. These relative measures of effort were later on used to compare the different processes.

4.3 Parameterization

Obviously the difference between the Cap and Non-Cap development processes is determined by the number of conflicts with other projects that arise during the different activities. We expressed this by introducing the *conflict probability parameter* c and parameterized the process models accordingly. Figure 3 exemplifies this for the *Integration Test* and shows how the conflict parameter c influences the transition probabilities.

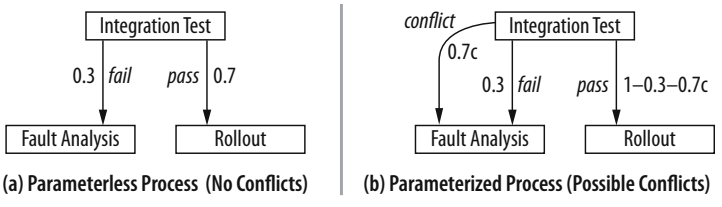

Fig. 3. Process Parameterization

4.4 CAP and Non-CAP Process Models

Based on the previously defined reference process we built specific models for Cap and Non-Cap development. The models differ as the Cap model contains specific Cap-related activities, e. g. *CAP Refresh* and the Non-Cap model explicitly describes conflict resolution activities (Fig. 4).

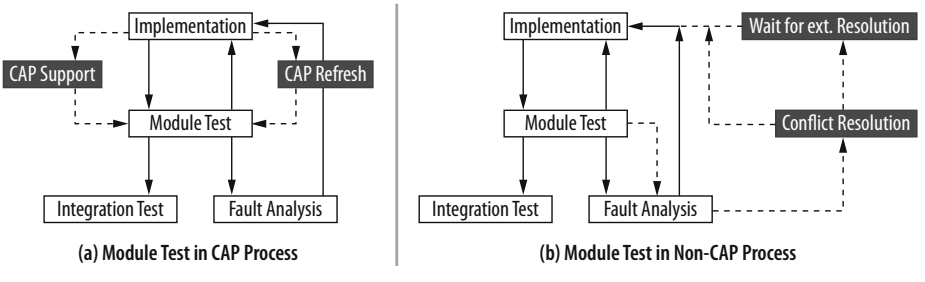

Fig. 4. Differences between CAP and Non-CAP Process (Module Test)

Please note that the Cap process, though isolated, is not fully free of conflicts as conflicts may arise during the *Integration Test* when the project leaves its Cap.

4.5 Relative Project Effort

For both processes the Markov chain analysis was carried out for different conflict probabilities and the total effort was put into relation with the same calculation for the reference process. Figure 5 shows the results in two resolutions. On the

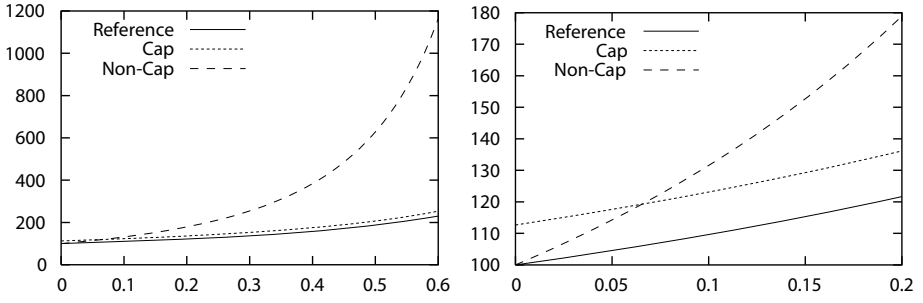

Fig. 5. Conflict Probability vs Relative Effort

left, the total effort for all three processes is shown for the conflict parameter interval $[0; 0.6]$. One can easily see that the efforts for the reference and CAP process behave in a similar way whereas the effort for the Non-CAP process increases much stronger. However, the right side with its finer resolution (interval $[0; 0.2]$) shows that for very low conflict probabilities the effort for the CAP process exceeds the effort for the Non-CAP process.

The results can be explained by analyzing the frequencies of each activity in the three process models. In the CAP and reference process an increasing conflict probability raises only the frequency of the integration test that is performed when the project leaves the CAP. In the Non-CAP process, however, the conflict probability also affects the module test. As the test activities constitute *nested loops* in the process this leads to a much stronger increase of the overall effort. It is also obvious that the CAP process has higher costs than the Non-CAP process for very small conflict probabilities as the cost for creating and maintaining the CAPs occurs independent of the conflict probability. This meets the expectation that CAPs are obsolete if there are no conflicts.

4.6 Estimation of the Conflict Probability

As the results of the process analyses show, the final decision on the economic efficiency of the CAP mechanism depends on the conflict probability parameter c. To determine the conflict parameter we analyzed the average number of dependencies among mainframe programs and examined the number of actual changes of these programs by using the configuration management system. The latter is important as program-to-program dependencies do cause conflicts only if both programs are modified at the same time.

For the analyzed period of one year we found that $\approx 55,86$ *relevant* (i. e. with possible conflict) changes occur for every program every year. Given a work year of 200 days this resolves to 0.279 relevant changes a day. As the reference process predicts about 100 test activities in a year this finally leads to a conflict probability of $0.279/2 = 0.1395$ or 14%.

Note that this does only regard program but *not* data dependencies. For data, the conflict probability is obviously dramatically higher.

5 Results and Discussion

The process analysis and the estimation of the conflict probability leads to the following conclusion:

> *Projects with an average number of dependencies save about* 20% *of total effort through using the* Cap *isolation mechanism as they avoid additional process cycles and conflict resolution activities.*

We therefore recommended to use non-isolated development only for projects with no or very few dependencies. Although we do not have a formal external validation of our results we can say that our results fully correspond with our project partners' experiences. In addition to this this recommendation was already followed before this study was conducted, as project managers intuitively chose isolated development only for projects with zero or few dependencies.

Although it is not detailed here, the difference between Cap and Non-Cap is even stronger with respect to time-to-system.

A new insight gained from this study regards the validity of test results if projects perform tests on shared data. As this drastically increases the conflict probability, enormous efforts are needed to ensure the validity of test results.

The major threats to the validity of these results is the determination of the transition probabilities and the memoryless nature of Markov chains.

Transition Probabilities. To evaluate how strongly different transition probabilities influence the results we performed a sensitivity analysis [5] to determine the transition that has the highest influence on the result. Using the variance-based *Extended FAST Method* [6] we found the transition *Module Test* → *Integration Test* to be not only the most *important* but with an *total order index* of 0.72 about three times as important as the second ranked transition. We therefore focused our analysis on the most important transition probability and found that changes to this probability do of course change the absolute efforts calculated for each process model. They do, however, *not* change the relation between Cap and Non-Cap development processes.

Memorylessness. The memorylessness of Markov chains implies that the transition probability from e. g. *Module Test* to *Implementation Test* and others is always the same, no matter how often the activities have been carried out before. As this might contradict one's intuition, we evaluated the influence of memorylessness by introducing a *process memory* in form of a compound interest function for the activity efforts. By defining a negative interest rate *(reduction rate)* we could simulate a situation where each execution of an activity demands less effort than the previous execution. Repeating the analysis for the two process models with this process memory showed again that the memory does influence the absolute results but *not* invalidate the relation between the Cap and Non-Cap development processes.

6 Related Work

Numerous *empirical studies* were conducted to answer similar process-related research questions, e. g. [7,8,9,10]. In general, empirical research generated highly valuable data that also helped us in calibrating the reference process model. However, empirical studies have a number of drawbacks that rendered them unsuitable in our situation. As it is impossible to replicate the same development project with two different processes (e. g. CAP vs Non-CAP) without changing any other influencing parameter, an empirical study would have to be carried out on *similar* projects. Due to the size and complexity of mainframe software development projects it is very hard to control their *similarity* and to correctly interpret the observations. As this could be overcome only by a significant number of repetitions of such studies, reliable results could be expected only after investing enormous amounts of time and effort [11,12,13,14].

Due to these reasons we chose to use an approach based on *process simulation*. Similar approaches where presented as early as in the 1950ies with the *Critical Path Method (CPM)* and PERT [15]. More recent approaches were presented (among others) by Drappa and Ludewig [16], Madachy [17], Podnar and Mikac [18], Zhang et al. [19] and Mockus et al. [20]. While all of these approaches served as highly valued inspirations, they are either of qualitative nature [19], too specific to their original application [20], do not consider conflict probabilities and project cycles [15,16,18] or were too fine-grained for our purpose [17]. Overviews on process simulation techniques can be found in [21] and [22].

Markov chain-based process simulation models were proposed earlier by Kulkarni and Adlakha [23], Hardie [2] as well as Minh and Bhaskar [24]. Kulkarni and Adlakha focus on the project completion time of PERT networks and do therefore analyze *acyclic* process models only. Hardie specifically includes cyclic process models and concludes that the reluctance to model project cycles is one of the main reasons for flawed predictions. However, he does not use absorbing Markov chains to calculate expected project efforts. Minh and Bhaskar extend Hardie's work by using absorbing Markov chains to determine the expected number of process steps but do not include analysis of project efforts. To our knowledge, neither of the above authors applied their approaches in an industrial context. Padberg [25] presented a model that is based on a Markov decision model to evaluate scheduling strategies. This approach does not model activities explicitly and could therefore not be used to determine the project effort in our case.

7 Conclusions and Future Work

While it is usually easy to determine the costs of specific techniques or methods applied in software development, it is almost always extremely hard to quantify the *economic* benefit of such measures. As decisions for or against such measures should be economically justified, this is a serious problem in today's software engineering practice.

To answer the question about the economic benefit of isolated test and development environments in mainframe software development we developed a stochastic process simulation that explicitly describes project risks and activity reiterations. We demonstrated how this model can be used to compare process variations and found that isolated test environments typically save $\approx 20\%$ development effort in the setting analyzed.

We believe that the incorrect predictions for project time and cost frequently encountered in practice are mainly due to project managers' reluctance to address project risks caused by unplanned reiterations of development activities. Therefore our current and future work focuses on applications of the model in the field of software project cost estimation. In this context we are working on the completion of the tool-suite that allows process design and analysis based on the methods presented in this paper.

References

1. Bacon, J.: Concurrent Systems: Operating Systems, Database and Distributed Systems: An Integrated Approach. Addison-Wesley, Boston, MA, USA (1993)
2. Hardie, N.: The prediction and control of project duration: a recursive model. International Journal of Project Management **19**(7) (2001) 401–409
3. Grinstead, C.M., Snell, J.L.: Introduction to Probability. AMS (2003)
4. Boehm, B.W.: Software Engineering Economics. Prentice Hall PTR, Upper Saddle River, NJ, USA (1981)
5. Saltelli, A., ed.: Sensitivity Analysis. John Wiley & Sons (2000)
6. Saltelli, A.: A quantitative model-independent method for global sensitivity analysis of model output. Technometrics **41**(1) (1999) 39–56
7. Williams, L., Kessler, R., Cunningham, W., Jeffries, R.: Strengthening the case for pair programming. Software **17**(4) (2000) 19–25
8. Phongpaibul, M., Boehm, B.: An empirical comparison between pair development and software inspection in Thailand. In: ISESE '06, ACM Press (2006)
9. Dingsøyr, T., Røyrvik, E.: An empirical study of an informal knowledge repository in a medium-sized software consulting company. In: ICSE '03, IEEE CS (2003)
10. Du, G., McElroy, J., Ruhe, G.: A family of empirical studies to compare informal and optimization-based planning of software releases. In: ISESE '06, ACM Press (2006)
11. Perry, D.E., Porter, A.A., Votta, L.G.: Empirical studies of software engineering: a roadmap. In: ICSE '00, ACM Press (2000)
12. Pfleeger, S.L.: Albert einstein and empirical software engineering. Computer **32**(10) (1999) 32–38
13. Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El Emam, K., Rosenberg, J.: Preliminary guidelines for empirical research in software engineering. IEEE Trans. Softw. Eng. **28**(8) (2002) 721–734
14. Seaman, C.: Qualitative methods in empirical studies of software engineering. IEEE Trans. Softw. Eng. **25**(4) (1999) 557–572
15. Malcolm, D.G., Roseboom, J.H., Clark, C.E., Fazar, W.: Application of a technique for research and development program evaluation. Operations Research **7**(5) (1959)
16. Drappa, A., Ludewig, J.: Quantitative modeling for the interactive simulation of software projects. The Journal of Systems and Software **46**(2–3) (1999) 113–122

17. Madachy, R.J.: System dynamics modeling of an inspection-based process. In: ICSE '96, IEEE CS (1996)
18. Podnar, I., Mikac, B.: Software maintenance process analysis using discrete-event simulation. In: CSMR '01, Washington, DC, USA, IEEE CS (2001)
19. Zhang, H., Huo, M., Kitchenham, B., Jeffery, R.: Qualitative simulation model for software engineering process. In: ASWEC '06, IEEE CS (2006)
20. Mockus, A., Weiss, D.M., Zhang, P.: Understanding and predicting effort in software projects. In: ICSE '03, IEEE CS (2003)
21. Kellner, M.I., Madachy, R.J., Raffo, D.M.: Software process simulation modeling: Why? what? how? Journal of Systems and Software **46**(2-3) (April 1999) 91–105
22. Williams, T.: The contribution of mathematical modelling to the practice of project management. IMA J Management Math **14**(1) (2003) 3–30
23. Kulkarni, V.G., Adlakha, V.G.: Markov and markov-regenerative pert networks. Operations Research **34**(5) (1986) 769–781
24. Minh, D.L., Bhaskar, R.: Analyzing linear recursive projects as an absorbing chain. Journal of Applied Mathematics and Decision Sciences (2006)
25. Padberg, F.: A comprehensive simulation study on optimal scheduling for software projects. In: ProSim '04, IEE (2004)