SOFTWARE PROCESS IMPROVEMENT AND PRACTICE
Softw. Process Improve. Pract. 2004; **9**: 229–238
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spip.208

Dynamic Evolution of Software Processes to Evolve Software Systems during their Development‡

Research Section

Sami Beydeda[1*,†] and Volker Gruhn[2]
[1] *Bundesamt für Finanzen, Abteilung Informationsverarbeitung, Friedhofstr. 1, 53225 Bonn, Germany*
[2] *University of Leipzig, Chair of Applied Telematics/e-Business, Klostergasse 3, 04109 Leipzig, Germany*

A software system, once deployed into its target environment, might need to be modified for various reasons. The reasons might be specific to that software system, such as failures, or, more general, such as changes in the environment in which the software system is embedded. Depending on the reason, a modification might obviously not only be restricted to a particular software system. It might also concern other existing software systems and particularly also software systems under development.

The modification of a software system under development is merely a problem of modifying its development process, also called software process. Such modifications generally cannot be automatically carried out by autonomous process support systems due to the complexity inherent in software processes and in the necessary modifications. It usually needs to be guided by a human process manager. A process support system can, however, offer the human process manager certain services to assist in modifying a software process. One of these services is that of decision support.

This article describes a decision support system developed in the ESPRIT project Process Instance Evolution (PIE). One of the features of the decision support system is an extendable database of decision models, each of which is capable of generating specific information to assist the process manager. One of these models is that of risk analysis. Risk analysis, as used in this context, encompasses assessment of the impact of a possible modification on certain software process attributes before actually changing a software process. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: process evolution; decision support; risk analysis

* Correspondence to: Sami Beydeda, Bundesamt für Finanzen, Abteilung Informationsverarbeitung, Friedhofstr. 1, 53225 Bonn, Germany
† E-mail: sami.beydeda@bff.bund.de

‡ The chair of Applied Telematics/e-Business is endowed by Deutsche Telekom AG.

1. INTRODUCTION

One of the objectives of software engineering research is to obtain means of decreasing time to market of software products. A motivation of this type of research is that nowadays time to market is one of the main competition factors in the software industry. A software company intending to differentiate from its competitors can achieve this by such means. Another motivation to decrease time to market, or, more generally, to decrease the throughput time of a software process is a decrease of process risk. The development of a software system is generally based on certain assumptions concerning the customers, competitors, markets etc. Many of these assumptions have a specific reference date in the future, particularly at the end of development, and can therefore be subject of interim change. Even though the likelihood that such an assumption changes during development generally decreases with the throughput time of the software process, it cannot be totally removed. We thus require the means to tackle changes in assumptions. Such a change in the assumptions of a software process can also impact the software system as the product to be developed. For instance, existing software systems are often changed in response to altered customers' quality requirements, and this in turn can require a software company to change accordingly software systems under development.

The modification of a software system under development is merely a problem of modifying its development process. Such modifications generally cannot be automatically carried out by autonomous process support systems due to the complexity inherent in software processes and in the necessary modifications. A process support system can, however, offer a human process manager certain services to assist in modifying a software process.

Basic services for modifying software processes include the following (Alloui *et al*. 2000):

1. *Monitoring support.* The software process has to be monitored together with its environment in order to detect internal and external problems. A software process-internal problem may be that of staff reduction, while a software process-external problem may be that of changes in customers' quality requirements.
2. *Decision support.* After identifying the need to evolve a software process, solutions eliminating the problems detected have to be generated. For instance, in the case of the software process-external reason given above, a solution could be to insert activities in the software process improving test data generation. Furthermore, these solutions have to be compared with each other according to a specific criterion in order to select the most promising alternative. A criterion suitable for this purpose is, for instance, risk minimization.
3. *Change support.* After having selected a particular alternative, it has to be implemented, requiring actual change of the software process. Generally, the modification necessary is decomposed into single atomic modifications, and the software process is modified according to them.
4. *Evolution strategy support.* The single activities as a whole necessary to modify a software process can be considered as a process, the *change meta-process*. A process support system can support the change meta-process by offering an evolution strategy support.

The objective of this article is twofold. We have seen that in some cases software has to be evolved even during its development and that this can be addressed by evolving its development process. Firstly, this article technically describes a concrete system for software process evolution with emphasis on decision support aspects. This is covered in Section 2. Secondly, the article conceptually considers decision support aspects by explaining a technique that can be used to assess the risk inherent in a possible modification of the development process. This is covered in Section 3. Section 4 shows the application of the risk assessment technique by example and Section 5 finishes the article with our conclusions.

2. GENERIC DECISION SUPPORT

The approach of model-based decision support was developed during a project, *Process Instance Evolution* (PIE) (Cunin 2000, Cunin *et al*. 2001), an ESPRIT Framework IV LTR project supported by the European Community. The main part of the project started in February 1999 and finished in July 2001. One of the objectives of the PIE project was the development of a system supporting the evolution of human-intensive, long-living, and distributed processes.

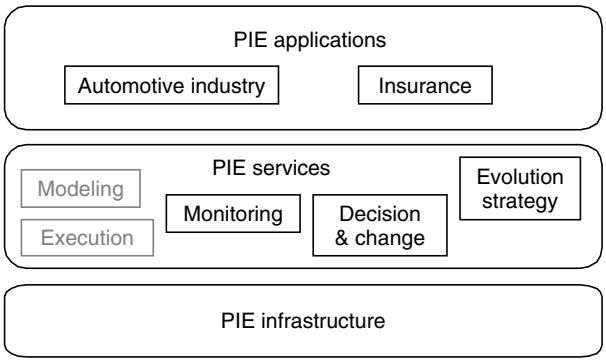


Figure 1. Overall architecture of the PIE system (Dami *et al*. 2001)

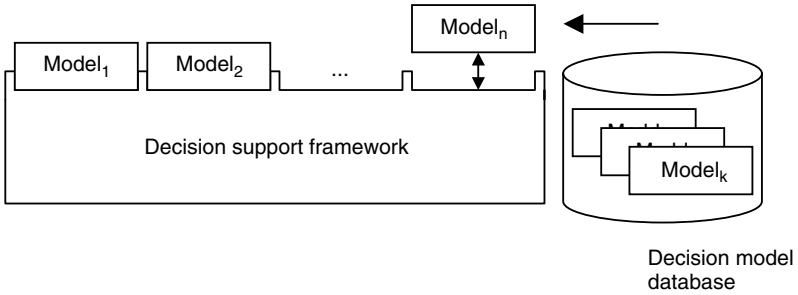


Figure 2. Conceptual architecture of the decision support component

2.1. Overall Architecture of the PIE System

The system developed in the PIE project consists of three levels, as shown in Figure 1 (Dami *et al*. 2001):

1. The top level consists of the application supported by the system. Besides software development, we, in particular, considered applications from the insurance and automotive industry.
2. The middle level, called *service layer*, includes the services provided by the system for supporting the application at the top level. The services mainly consist of *Monitoring Support*, *Decision Support*, *Change Support* and *Evolution Strategy Support*. Furthermore, this level also includes services for modeling and execution (enactment) of processes.
3. The bottom level, called *infrastructure layer*, represents the PIE infrastructure providing general services required by components at the service layer. The infrastructure layer particularly addresses process interoperability and mobility issues, often encountered in federations of heterogeneous and distributed components.

2.2. Decision Support Component

The Decision Support component is designed according to the client/server paradigm. In the following, we explain the conceptual and technical architecture of the Decision Support server and show an example of a Decision Support client.

2.2.1. Decision Support Server

The underlying concept of model-based decision support is to have a generic decision support framework that includes an extendable database of decision models. Examples of such decision models are those for generating alternatives or models for risk analysis. As new requirements can arise during the use of such decision support tools, a feature of model-based decision support is its extensibility; new models can be added to the database if needed.

Conceptually, the Decision Support server maintains a database of models that includes the algorithm explained later as well as other algorithms for decision and risk analysis, as shown in Figure 2. These models can be loaded into the Decision Support server during runtime if needed and can be

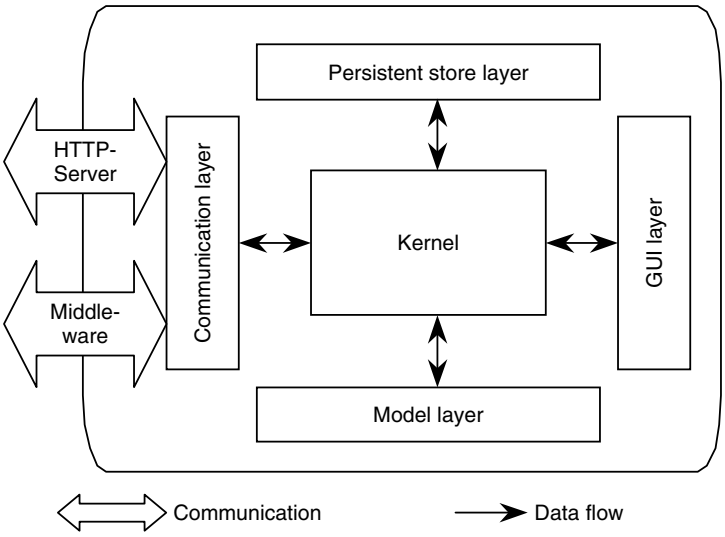


Figure 3. Technical architecture of the decision support server

executed with the necessary data. This data, specified by the corresponding model, can be accessed via the PIE Infrastructure.

The functionality of the Decision Support component strongly depends on the models available through its model database. Existing models not involved in a decision can be exchanged by newer versions and new models can be registered without recompiling or stopping the whole component.

Figure 3 shows the technical architecture of the Decision Support server consisting of five layers, namely, *Communication layer*, *Persistent store layer*, *Graphical user interface layer*, *Model layer* and *Kernel*:

- The *Communication layer* provides the necessary means for the communication of the Decision Support server to clients as well as to other PIE components. It implements interfaces providing different protocols. Clients can connect to the server through RMI either as a Java applet or a stand-alone Java application. The server computes the data related to the current query and sends a response to the client, which displays the data
- The *Graphical user interface layer* contains graphical elements to allow the models creating appropriate graphical user interfaces to display data.
- The *Model layer* implements the conceptual database shown in Figure 2. As the models are represented by Java classes.
- The *Persistent store layer* provides the necessary services for storing data obtained by other PIE components or computed by decisions models. The reason for storing the data computed by models is to facilitate learning. Even though the models available do not employ learning techniques yet, the project manager can use this data as experience for future decisions.
- The *Kernel* mainly executes the models available via the Model layer. It gathers the necessary data using services of other PIE components and invokes the corresponding model with the required data.

2.2.2. Decision Support Client

As mentioned before, services provided by the Decision Support server can be accessed by various types of clients. Figure 4 shows the user interface of a client implemented as a Java applet. This figure shows a client used for decision making in software development. In this particular scenario, two alternatives have been analyzed with respect to throughput time. Two alternatives were available, namely, the alternatives *Planning Amendment* and *Recovery Process*. A probability distribution was computed for each of them and was shown graphically by the client. Furthermore, the client determined that alternative *Recovery Process* entails less risk with respect to throughput time and suggested selecting it according to the *Min Risk* criterion.

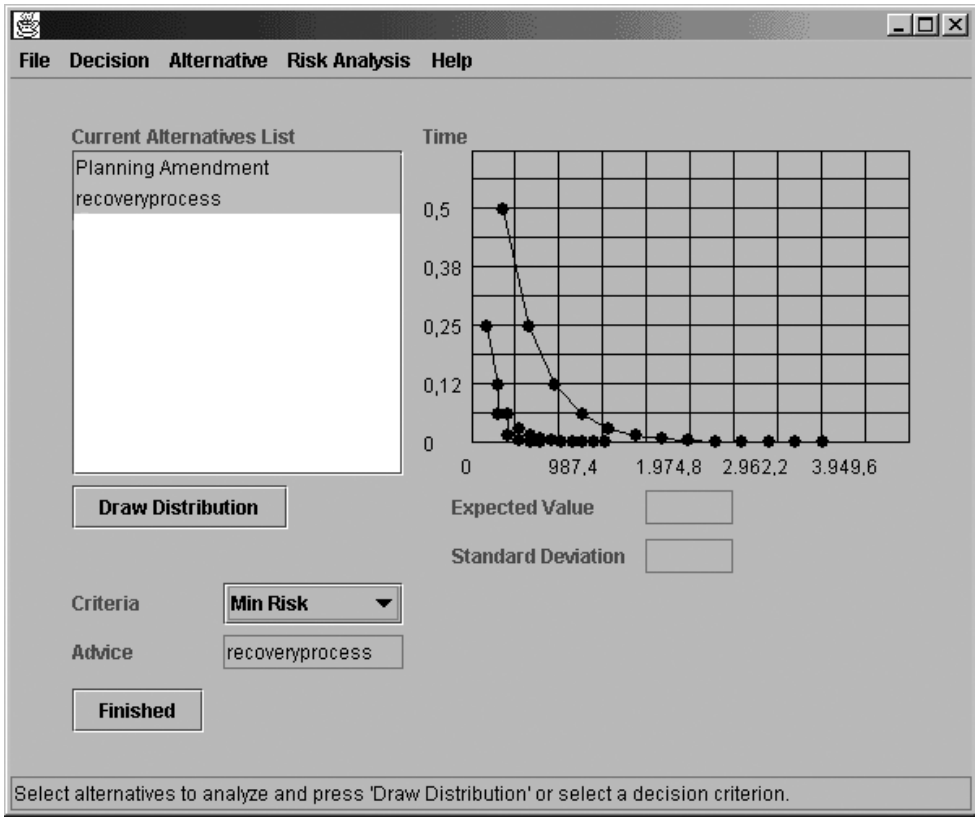


Figure 4. Decision support client implemented as Java applet

3. DECISION MODEL FOR RISK ANALYSIS

The decision model developed for risk analysis can be used for comparing possible alternatives with respect to the risk of two process parameters, *throughput time* and *total costs*. Throughput time refers to the time needed to finish a process. Total costs refer to the costs incurred by the process until its completion. The main concept of our risk analysis is to determine probability distributions of these parameters for the various alternatives based on their process models. These probability distributions can be used to compute simple risk measures such as the *expected value* and the *variance* (Allen 1978) of the distribution, or the more complex *value-at-risk* measure (Jorion 1997) used in finance. Having computed such risk measures, the *utility* (Pratt 1964) of each alternative can be determined, taking into account the preferences of the process manager. A useful side effect of the algorithm is information indicating opportunities for process optimization and also the likelihood of such undesirable situations as a deadlock.

3.1. Related Work

Almost every decision problem is associated with risk due to the uncertainty of the real world. Despite its importance, there is no standard definition of the term *risk* and its interpretation depends on the context (Fishburn 1984, Luce 1980, Jorion 1997). In this paper, we interpret the notion of risk in accordance with Moskowitz and Bunn (Moskowitz and Bunn 1987) as the likelihood of an unfavorable outcome.

Researchers have long considered risk analysis and decision problems under conditions of uncertainty. Bernoulli and Bayes developed a formal framework which was further developed by Ramsey (Ramsey 1931), von Neumann, and Morgenstern (von Neumann and Morgenstern 1947), Arrow (Arrow 1963) and others. The method used in this paper is based on this formal framework.

Unfortunately, existing process support systems do not sufficiently check process models before enacting. This means that after its enactment, a process can enter a deadlock state, causing serious problems van der Aalst 1998, Hofstede *et al*. 1998. The situation becomes even worse after changing a process. Although process changes can be associated with unpredictable side effects and risks, there are few publications addressing this problem. Tarumi, Matsuyama, and Kambayashi (Tarumi *et al*. 1999) have proposed a cycle of Business Process Tactics/Business Process Reengineering/Business Process Simulation with emphasis on simulation. In their approach, simulation is used to evaluate processes and agents before introducing them to the real work place.

3.2. Representation of a Process

The decision model for analyzing a process requires a process model that does not necessarily need to include information concerning data flow. The algorithm rather requires as input a process model that only defines the control flow of the process and additional information describing its current state. Therefore, the graphical representation of the process model used is quite simple, consisting of rectangles and edges between them. Nodes in the graph represent activities of the process. Each node possesses two attributes. One of which represents the costs of the resources required for the execution of the activity. The other represents the time needed to finish the activity under consideration.

Control flow within the process is represented by edges connecting the nodes. Since an activity can have different outcomes and the subsequent activity may depend on the outcome, a node can have more than one outgoing control flow edge. In these cases, each outgoing edge is augmented with a probability to indicate the likelihood of taking a particular branch. Obviously, the sum of the probabilities attached to the outgoing edges of one activity has to be exactly 1, and, in the case of only one succeeding activity, the probability attached to the edge corresponds to 1. In our approach, the state of a process is represented by a set consisting of the currently executed activities. Activities that are not contained in this set are not executed and are considered to be idle. In addition to the two parameters, time and costs of an activity, the set of currently executed activities also maintains two other parameters for each activity. One of these parameters is the probability of entering the activity and the other the costs incurred by the process enactment until entering the activity. Thus, the cost and the probability parameters attached to an exit node in the set of executed activities give the corresponding values for finishing the process at that particular node.

3.3. Process Time Schedule

The technique proposed is based on an algorithm to gather *process time schedule* information. A process instance time schedule is a set consisting of items that indicate the end of process enactment. The items have attributes for total costs and for the probability of finishing at the corresponding time. The algorithm generating the time schedule is given in Figure 5. The basic idea of the algorithm is to consider every possible execution thread of the process simultaneously. Starting with the set of executing activities characterizing the current state of the process, the currently terminated activity is deleted from this set. Its subsequent activities are inserted either in the set of executed or waiting activities, depending on the availability of the required products and resources. Each time the final activity of the process is deleted from the set of executed activities, an item indicating this event is inserted into the time schedule. The item contains information concerning the total costs and the probability of that event. After finishing the execution of an activity, the set of waiting activities is searched for activities that have become ready to start, i.e. the required products and resources have become available. This loop is executed until the set of executed activities is empty, i.e. there are no further threads of execution in the process. At this point, a nonempty set of waiting activities indicates a deadlock situation. The probability attached to each activity in the nonempty set gives the likelihood of the particular deadlock situation.

3.4. Computation of Probability Distributions and Risk Measures

The output of the above algorithm, i.e. the time schedule for a particular process, can be used to determine the probability distribution of both the throughput time and the total costs. In order to obtain the probability for a particular time or cost

```
1  algorithm processInstanceAnalysis;
2  input processModel;
3    setOfExecutedActivities;
4  output processInstanceTimeSchedule;
5    setOfWaitingActivities;
6  begin
7    T = 0;
8    setOfWaitingActivities = ∅;
9    while setOfExecutedActivities not empty do
10     for all activities A ready to start in setOfWaitingActivities do
11       delete A in setOfWaitingActivities;
12       insert A in setOfExecutedActivities;
13     determine that activity A in setOfExecutedActivities finishing next;
14     delete A in setOfExecutedActivities;
15     instId = A_instId;
16     for all B in setOfExecutedActivities do
17       B_t = B_t − A_t;
18     T = T + A_t;
19     if A is an exit node of processModel then
20       m_t = T;
21       m_c = A_c;
22       m_p = A_p;
23       put m in processInstanceTimeSchedule;
24     for all successor S of A in processModel with S_pA_p > p_thres do
25       S_c = S_c + A_c;
26       S_p = S_pA_p;
27       S_instId = instId;
28       instId = newInstId();
29       if S_instId ≠ A_instId then
30         for all B in setOfExecutedActivities with B_instId = A_instId do
31           C = copyB;
32           C_instId = S_instId;
33           put C in setOfExecutedActivities;
34         for all B in setOfWaitingActivities with B_instId = A_instId do
35           C = copyB;
36           C_instId = S_instId;
37           put C in setOfWaitingActivities;
38         if all resources and products are available for the execution of S then
39           put S in setOfExecutedActivities;
40         else put S in setOfWaitingActivities;
41     if processModel does not contain a successor S of A with S_pA_p > p_thres then
42       for all B in setOfExecutedActivities with B_instId = A_instId do
43         delete B in setOfExecutedActivities;
44       for all B in setOfWaitingActivities with B_instId = A_instId do
45         delete B in setOfWaitingActivities;
46   return (processInstanceTimeSchedule, setOfWaitingActivities);
47 end
```

Figure 5. Algorithm for analyzing process instances

value, the time schedule is queried for that value. If the time schedule contains several entries with this particular value, their probabilities are summed up to compute its probability.

After having identified a probability distribution, statistical measures can be used to define appropriate risk measures. As mentioned above, the expected value and the variance (Allen 1978)

of the distribution, or the more complex value-at-risk (Jorion 1997) defined on the basis of the distribution, can be used as risk measures. On the basis of such risk measures, a utility function (Pratt 1964) can be defined, taking into account the preferences and the risk aversion of the process manager. Thus, selecting an alternative involves computing the utility of all available alternatives and then selecting that alternative that dominates all others.

3.5. Process Optimization

The results of the algorithm can be used for three kinds of optimization. Firstly, the set of waiting activities of an optimal process instance should always be empty. Generally, a waiting activity indicates that the throughput time of a process instance can be decreased by synchronizing predecessor activities. Secondly, optimization can be achieved by analyzing the set of executed activities. A process instance can be effectively improved with respect to throughput time and total costs by improving those activities that appear frequently in this set. Thirdly, the parameters of a process instance that determine the shapes of the probability distributions for throughput time and total costs are known. Thus, the shapes of these probability distributions can be formed as intended by altering these parameters. Specifically, this optimization can be automated if symbolic expressions are computed for the various probabilities.

4. EXAMPLE: SOFTWARE TESTING

4.1. Scenario Description

The example used to explain the algorithm is a software process model covering test activities. The process model is given in Figure 6 in the form required by the algorithm. This figure shows the process in its initial form (solid drawn lines) together with an extension (dashed lines), which aims at improving the initial process. Each activity in this figure is augmented with two values: the first (second) indicates the time consumed by the corresponding activity within the initial (revised) process. Obviously, since activity *Determine Test Adequacy* did not exist in the initial version, only one time value is given. Furthermore, probabilities are attached to edges. Again, the first probability gives that of the initial process and the second that of the revised process. The objective of the analysis is to compare these two versions of the process, the initial process and the extended process, with regard to the time required for testing.

The initial process consists of five activities: *Seeding artificial Errors*, *Generating Test Cases*, *Testing*, *Counting Artificial Errors*, and *Debugging*. The initial process starts with seeding artificial errors into the program under test. The objective of seeding artificial errors is to obtain an estimate of the number of errors remaining in the program after testing (Zhu *et al*. 1997). After seeding errors, test cases are generated. In the initial process, test cases are generated randomly, which, although cheap in terms of cost and time, are inefficient in terms of detecting errors. After generating test cases, the program under test is executed with these test inputs and failures are registered. To determine the quality of the test cases, the artificial errors detected are counted. Depending on the number of the detected artificial errors, either test case generation is repeated or the testing process is finished by debugging, i.e. removing all detected faults.

After carrying out several test processes according to the initial model, the tester recognizes that test case generation has to be repeated several times in order to achieve a sufficient quality. The reason is

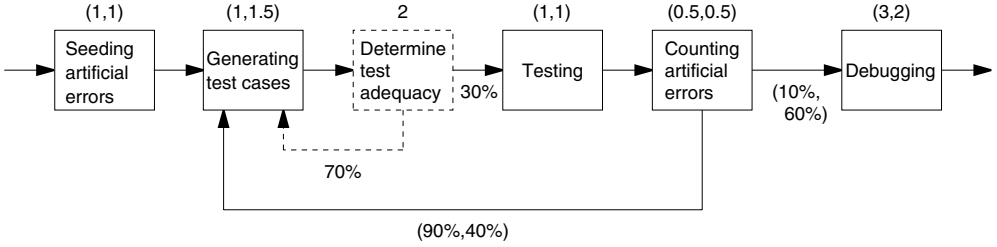


Figure 6. A software testing process

obvious, generating test cases randomly. Therefore, the tester tries to optimize the process by generating test cases systematically by white-box techniques. Generally, white-box techniques require generating test cases on the basis of the source code, which fulfill a certain control or data flow criterion.

But, before changing the process, the tester has to determine which process will lead to savings, at least in time. Even in this simple example, it is almost impossible to predict which alternative dominates the other.

4.2. Determining the Probability Distributions

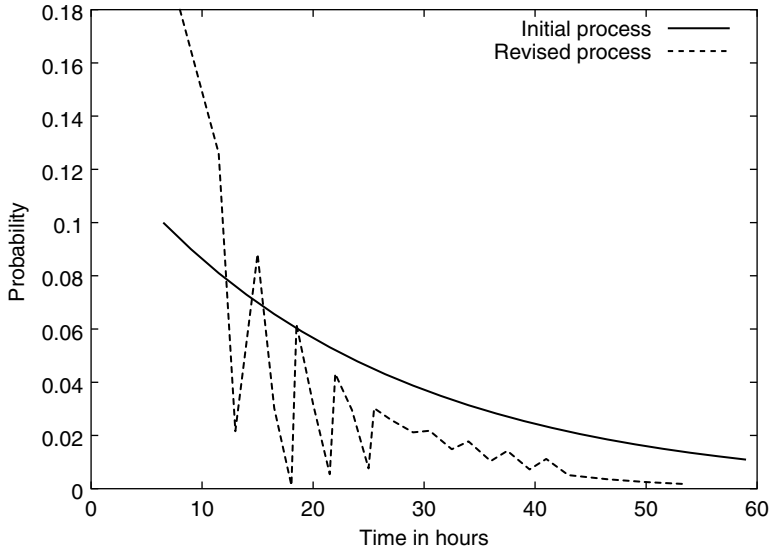
As explained above, the basic idea of the algorithm in Figure 5 is to consider all possible threads of the process enactment simultaneously. Since the number of all possible threads might be very large, only those that can occur with at least a probability p_{thres} are considered. In our case, p_{thres} was set to 0.01 for the analysis of the processes.

After calculating the process time schedules of both processes, probabilities of equal time values are cumulated to a single value. The probability distributions obtained in this way are depicted in Figure 7.

4.3. Using the Risk Measures for Decision Making

In our simple example, the revised process dominates the initial process by shorter throughput times being more likely than in the initial process and high throughput times being less likely as in the other case, as shown by the probability distributions. This is also expressed by the expected value μ and the variance σ^2 of the distributions.

In this simple case, a decision can be induced solely on the basis of the expected value and the variance, since the revised process dominates the initial process in both parameters. However, it is also possible that one alternative dominates the other with respect to one parameter and vice versa with respect to the other parameter. In these cases, preferences of the process manager can be used to calculate the *utility* (Pratt 1964) of an alternative to compare with those of other alternatives. For instance, the process manager might be *risk averse*, i.e. the process manager would prefer the alternative having the least risk, although other alternatives can dominate this alternative with respect to the expected throughput time or total costs.



Alternative	Initial process	Revised process
μ	20.728	14.863
σ^2	181.295	82.365

Figure 7. Probability distributions and statistical measures of the alternative processes

5. CONCLUSIONS

A problem often encountered in practice is that software, even though still being in development, needs to be evolved due to various reasons such as changes in user requirements. The evolution of software under development is merely an issue of software process evolution, and we therefore need the appropriate means to change software processes during their enactment.

We have described model-based decision support as a generic approach to supporting the decision process. The underlying idea of model-based decision support is to have a generic framework that is capable of loading the necessary decision models from an extendable model database and executing them with the required data. The benefit of this approach is that the framework can be adapted to requirements of changing decision processes.

Furthermore, we have explained a decision model for analyzing processes to determine the risk of certain process parameters and to support the process manager in decisions related to process evolution. The technique requires a process model that only defines control flow within the process. Since this information can be easily provided for most processes, the technique is applicable to a wide variety of processes. Furthermore, the technique is automatable. After having entered the process model to be considered, analysis of the process is carried out automatically.

REFERENCES

Allen AO. 1978. *Probability, Statistics, and Queueing Theory*. Academic Press, New York, USA.

Alloui I, Beydeda S, Cimpan S, Gruhn V, Oquendo F, Schneider C. 2000. Advanced services for process evolution: Monitoring and decision support. In *European Workshop on Software Process Technology (EWSPT)*, Springer Verlag: Berlin, 21–37.

Arrow KJ. 1963. *Social Choice and Individual Values*, 2nd edn. Wiley, New York, USA.

Cunin P-Y. 2000. The PIE project: an introduction. In *7th EWSPT European Workshop on Software Process Technology*, Springer Verlag: Kaprun, Austria, 1–5.

Cunin P-Y, Greenwood RM, Francou L, Robertson I, Warboys B. 2001. The PIE methodology – concepts and application. *EWSPT European Workshop on Software Process Technology*, Springer Verlag: Witten, Germany, 3–26.

Dami S, Cunin P-Y, and Francou L. 2001. Operation manual of the PIE system – consolidated version. Technical report, ESPRIT Project Process Instance Evolution (PIE). Techreport number D1.10, http://www.cs.man.ac.uk/ipg/pie/public-e.htm.

Fishburn PC. 1984. Foundations of risk management: risk as a probable loss. *Management Science* **30**: 396–406.

Greenwood M, Robertson I, Warboys B. 2000. A support framework for dynamic organizations. In *7th EWSPT European Workshop on Software Process Technology*, Springer Verlag: Kaprun, Austria, 6–20.

Hofstede A, Orlowska M, Rajapakse J. 1998. Verification problems in conceptual workflow specifications. *Data and Knowledge Engineering* **24**(3): 239–256.

Jorion P. 1997. *Value at Risk: A New Benchmark for Measuring Derivatives Risk*. Irwin Professional Publishing, Chicago, USA.

Luce RD. 1980. Several possible measures of risk. *Theory and Decision* **12**: 217–228.

Moskowitz H, Bunn D. 1987. Decision and risk analysis. *European Journal of Operational Research* **28**: 247–260.

Pratt J. 1964. Aversion in the small and in the large. *Econometrica* **32**: 122–136.

Ramsey FP. 1931. Truth and probability. In *The Foundations of Mathematics: Collected Papers of Frank P. Ramsey*, Routledge and Kegan Paul: London, UK, 156–198.

Tarumi H, Matsuyama T, Kambayashi Y. 1999. Evolution of business processes and a process simulation tool. In *APSEC Asia-Pacific Software Engineering Conference*, IEEE Computer Society Press: Takamatsu, Japan, 180–187.

van der Aalst WMP. 1998. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers* **8**(1): 21–66.

von Neumann J, Morgenstern O. 1947. *Theory of Games and Economic Behaviour*. Princeton University Press, Princeton, USA.

Zhu H, Hall PAV, May JHR. 1997. Software unit test coverage and adequacy. *ACM Computing Surveys* **29**(4): 366–427.