Market-driven approach based on Markov decision theory for optimal use of resources in software development

J. Noppen, M. Aksit, V. Nicola and B. Tekinerdogan

Abstract: Changes in requirements may have a severe impact on development processes. For example, if requirements change during the course of a software development activity, it may be necessary to reschedule development activities so that the new requirements can be addressed in a timely manner. Unfortunately, current software development methods do not provide explicit means to adapt development processes with respect to changes in requirements. The paper proposes a method based on Markov decision theory, which determines the estimated optimal development schedule with respect to probabilistic product demands and resource constraints. This method is supported by a tool and applied to an industrial case.

1 Introduction

In general, requirements hardly remain constant during the lifetime of a software system. Software systems should, therefore, cope with changes in requirements in a cost effective manner. Unfortunately, several case studies have shown that more than 80% of the total cost of software systems is devoted to software maintenance [1]. Adapting software to changing requirements covers a major part of these costs. Changes in requirements not only have an impact on software systems, but also may cause rescheduling of software development processes. For example, if requirements change during the course of a software development activity, it may be necessary to reschedule development activities so that the new requirements can be addressed in a timely manner.

To reduce the maintenance costs often the potential changes in requirements are anticipated as much as possible. Various languages and systems provide techniques to cope with changes within a given context. Each technique assumes that certain aspects of software should be evolvable, while other aspects should be fixed. For example, separation of interface and implementation techniques, such as the model-view-controller pattern [2], assumes that implementation should be evolvable but not the interface. Meta programming techniques assume that *base-level* should be evolvable but not the meta interface [3]. Of course, to support a larger degree of evolution, multiple techniques can be combined with each other. For example, separation of interface and implementation techniques can be combined with meta programming to make interfaces more adaptable, meta programming languages can be made adaptable by introducing meta meta languages, etc.

Changes in requirements not only have an impact on software systems, but also may cause rescheduling of software development processes. For example, it may be necessary to redistribute human resources so that software systems can be delivered in time. This paper proposes an approach for optimum distribution of human resources when facing nondeterministic changes in requirements. The approach is based on the Markov decision model, which constructs probabilistic state-transition diagrams that represent market change scenarios. By estimating changes in market demands, available human resources can be distributed optimally at certain points in time.

2 Unanticipated software evolution

Software evolution usually implies the adaptation of the software system due to certain changes. These changes could include both changes that were expected and those that were not. Software evolution can be supported by planning the expected changes as much as possible. Unfortunately, not all changes can be planned for and remain unanticipated. Nevertheless, unanticipated changes form an important role in software evolution. To support better software evolution it is therefore required that we also provide techniques to cope with the unanticipated changes.

The required modifications that were unanticipated might be caused by different factors such as insufficient information, lack of diligence, lack of required skill, or simply because it was humanly impossible to predict due to the complex and dynamic nature of the problem being solved. By definition, unanticipated software evolution (USE) is not something for which we can prepare during the design of a software system [1]. The adaptations might require implementing the changed requirements in the application or sometimes it could even include adaptation to the changing environment so that the application keeps satisfying the earlier requirements. Although the extent of software evolution can be very broad, in this paper we

© IEE, 2004

IEE Proceedings online no. 20040079

doi: 10.1049/ip-sen:20040079

Paper first received 1st April and in revised form 5th November 2003

J. Noppen and M. Aksit are with the TRESE Software Engineering Group, Dept. of Computer Science, University of Twente, PO Box 217, 7500 AE, Enschede, The Netherlands

B. Tekinerdogan was with Bilkent University, Ankara, and is now with the TRESE Software Engineering Group, Dept. of Computer Science, University of Twente, PO Box 217, 7500 AE, Enschede, The Netherlands

V. Nicola is with Telematics Systems and Services, Dept. of Electrical Engineering, University of Twente, P.O. Box 217, 7500 AE, Enschede, The Netherlands

primarily focus on software evolution due to unanticipated changes in the requirements.

We categorise unanticipated requirements as follows.

2.1 Uncertainty in time of change

In this category, the actual change to the requirements is known at the moment of system design. However, changes in requirements can occur at a different moment in time than was expected (all be it earlier or later). For example, major legislative adaptations on insurance laws may have passed the parliament but the actual enactment of this change in the law can still be unclear. These changes will possibly have a considerable impact on the software systems that the insurance companies use.

2.2 Uncertainty in content of change

This kind of unanticipation means that the actual occurrence of a certain change in requirements is not known, even though the change itself is known. For example, a new insurance law may have been proposed in the parliament without any clarity on whether this law will come to pass or not. In this case, a possible impact of this change on the related software systems is known but it is not clear whether this change will be finally implemented.

2.3 Uncertainty both in time and content of change

In this category, there is no clarity on what will happen. Every change is possible at any given time.

The above classification implies that unanticipation is related both to the content as well as the occurrence of the change to the requirements. As such even if requirements are known in advance the time of occurrence for the requirement change can be fully unanticipated. Including both types of unanticipation will support the software evolution activities. It should be noted that a practical evolution case might be classified under several of these evolution categories.

A number of different approaches have already been proposed that address specific issues related to unanticipated software evolution. These approaches range from extensions of programming languages [4, 5] to adaptable architectures [6] and calculi for describing and reasoning about language properties [7]. The main focus of these approaches is on solving technical issues related to changes that could occur at model, artefact and/or code level. As such, most of these approaches provide solutions to unanticipated software evolution in case of uncertainty in content of changes. To cope with uncertainty in time of changes to requirements we think that unanticipated software evolution must also be considered at the process level. This is because changes in requirements can invalidate process configurations, person hour distributions, etc. As a complementary approach to the existing software evolution techniques we will provide an approach that is specifically targeted at uncertainty in time of occurrence of the requirements.

3 Problem statement

In the following sub-section, first we will present an industrial example case. Second, by using this example, we will explain some problems that are caused by unanticipated changes. These problems will be addressed in Section 4.

3.1 Example: application framework for insurance systems

Consider a software company that produces and maintains software systems to be used by insurance companies. It is assumed that the software company delivers a dedicated software system for each insurance product. In addition, the software company aims to market the same software system to multiple insurance companies worldwide. Insurance companies can tailor these systems according to their specific policies.

Generally, insurance companies have a large customer base, with a variety of insurance products and policies. Of course, the insurance product characteristics are not static but they evolve in accordance with the needs of society. For example, a growing trend in the insurance market is the demand for tailored and personalised insurances.

For the software company, to a certain degree it is possible to deal with the complexity of this evolution by adopting application frameworks, object-oriented and component-oriented techniques. Application frameworks are programs that capture the generic parts of several software systems and encapsulate them in reusable components. Actual software systems can be produced by composition of several of these components [Note 1]. The components are identified by discovering the variabilities and commonalities in the problem domain. For example, Fig. 1 shows a feature diagram of insurance products [Note 2]. A feature diagram represents both the commonality and variability of a product [8]. In this paper, to avoid confusion, products that are sold by software and insurance companies are referred to as insurance software systems and products, respectively.

In Fig. 1, four different parts of insurance products are shown: *insured object*, *coverage*, *payment* and *payee*. Each of these parts can be represented by several entities. For example, an insured object can be a *person* or a *corporation*. The symbols that are depicted in the legend are used as restrictions on the variability. For instance, an insured object cannot be a corporation and a person at the same time. Using this specification, a large variety of insurance products can be defined, ranging from bicycle insurances to tailored insurances for large corporations. In addition, new insurance products may be introduced in the future, possibly using advanced personal computing facilities and the Internet.

3.2 Resource-scheduling problems due to unanticipated changes

Although framework based approaches provide means to deal with problems that may occur when variability, process-scheduling problems may still exist in the case of unanticipated changes. In the following, we present two problems when dealing with evolution of the insurance software systems demands.

3.2.1 Difficulty in prioritising requirements:

Assume that a large set of components have to be implemented in a time span of 2 to 4 years. These components will be used to create several different insurance software systems whenever they are demanded. The components and systems share dependency relations among each other. Therefore, it is crucial to deliver the essential components first, before the insurance companies demand systems that incorporate these components. This

Note 1: Application frameworks differ from product–line architectures in that frameworks are defined at programming language level whereas product–line architectures are specified more abstractly. Application frameworks can be implemented as reusable class hierarchies or as pluggable components.

Note 2: This project has been carried out together with Utopics B.V.

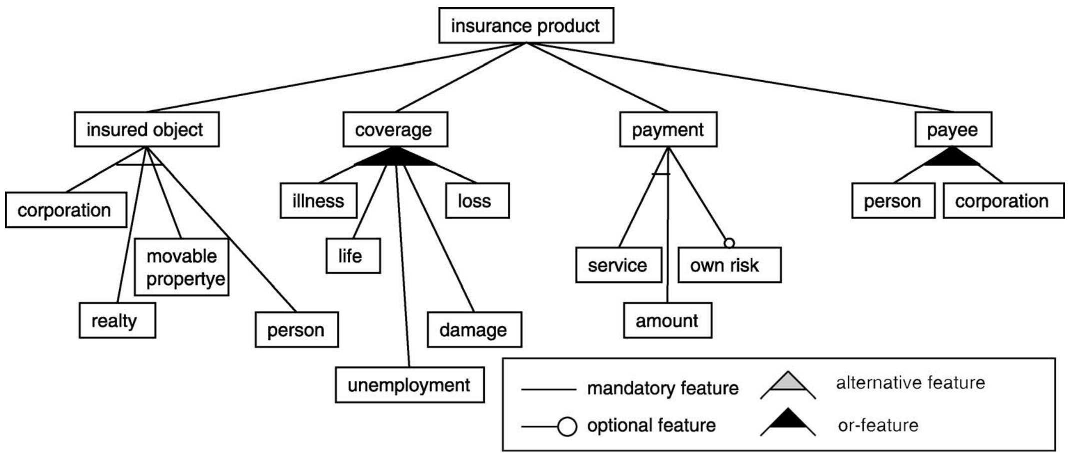


Fig. 1 *Feature diagram of insurance products*

requires giving higher priorities to the components that are demanded first.

A number of software development methods suggest prioritisation of requirements [9]. However, these methods do not provide explicit means for determining the priority values with respect to the changes in demands.

Several publications and tools are proposed for supporting decision-making processes. Most of the approaches are based on the analytic hierarchy process (AHP) method [10–12]. In the AHP method, a pairwise comparison is made between two entities that should be ranked with respect to each other. Subsequently the pairwise ordering is normalised. The AHP method is applied to various kinds of problems such as resource allocation, risk assessment, etc. However, the proposed approaches assume a fixed set of requirements and therefore they are not directly suitable for dealing with unanticipated software evolution.

3.2.2 Configuring software development processes for delivering products in time: After prioritising the components, the project managers should be able to configure the implementation trajectory accordingly. Since there may be many options for the implementation effort, ideally the project managers should be able to compare all the relevant options and select the best configuration that fulfils the timing and resource constraints. For the example case, this means that the expected optimal planning of resources for a given time span should be determined to maximise profit in selling insurance software systems.

Configuring software processes with respect to available resources is not new [13]. However, in contrast to existing methods, also the demands for future systems and changes in requirements must be considered.

4 The approach

4.1 Possible approaches

The following sub-section gives a short description of a number of alternatives for decision making optimisations.

4.1.1 Real options: In [14] the approach of real options is described as a means of supporting decision making in software design. In real options theory each decision that can be made is considered an option and has its own *added value*. With this added value the actual gain of choosing this option is signified. In addition to the options that are identified also scenarios are identified and their change of occurrence. From these scenarios follows an event tree with which it is possible to determine the expected added value for choosing one of the available options. The underlying mathematical theory for the real options approach is based on dynamic programming, also known as Markov decision theory.

4.1.2 Quality function deployment: Quality function deployment [15] is a well known tool for analysis of the importance of decisions in software design. One of the particular uses for this approach can, for instance, be found in resolving conflicting requirements. In quality function deployment a distinction is made between the requirements from the customer side (called the voice of the customer) and the parameters of design (called the voice of the engineer). The relationship between these two voices is described in the relationship matrix. QFD can balance and prioritise the customer needs and design parameters based on the relationship matrix.

4.1.3 Markov decision theory: Markov decision theory is a mathematical theory for determination of optimal decisions at a given point in time. This is done by determination of the decisions that can be made and the definition of scenarios that can occur. For each of the scenarios the probability of occurrence is also described. These two inputs are then combined into a tree structure, which is commonly termed a Markov decision problem. The optimality of the decisions is then calculated based on expectance values of rewards for each decision. Markov decision theory is also known as dynamic programming.

In our approach we have elected Markov decision theory as the underlying mathematical theory since it provides a sound mathematical framework for solving the type of problems described in Section 3. Real options theory is related to our approach since the same mathematical basis is used, and the globally the approaches share similarities. However, our approach offers a more clear-cut solution for resource allocation problems since our application of the mathematical theory explicitly addresses the typical inputs for these problems.

Quality function deployment has a less formal mathematical basis than dynamic programming and as such is easier to understand in its application. However, with the large amount of inputs and the probabilistic character of the problem, QFD does not provide the proper tools for efficient determination of optimal development trajectories.

4.2 Method

The options in scheduling software development processes depend on the possible software system demands.

The following three steps are used to determine the most profitable option:

1. Determine probabilistic estimations for market, cost and profit changes.
2. Specify the possible market scenarios that can occur.
3. Evaluate all possible schedules with respect to the scenarios and select the most cost effective schedule.

This method should be flexible enough to cope with changes in the estimations at any given moment in time.

4.3 Markov decision model

To achieve the objectives stated in the preceding Section, the problem needs to be formalised mathematically. In our approach the optimisation problem is formalised using Markov decision theory [16]. Below, the formalisation is defined for this specific problem.

• Consider a time horizon of length I time intervals (steps). Let i $(1 \le i \le I)$ be the index of the time interval with i steps to go. That is, the time horizon starts at the beginning of interval I and ends at the end of interval 1.
• Let J be the total number of artefact types that may be produced to satisfy market demands during the entire time horizon. Market orders for artefacts are placed at the beginning of each time interval, and are lost if not satisfied by the end of the same time interval. At the beginning of interval $i\,(1 \le i \le I)$, the demand for artefact j is a random number D_{ji} characterised by the (discrete) distribution F_{ji} with a finite mean given by d_{ji}. The price of one unit of artefact j at the end of interval i is given by r_{ji}. Denote by $\mathbf{d}_i$ and $\mathbf{r}_i$ the *demand* and *price* vectors with elements d_{ji} and r_{ji} $(1 \le j \le J)$, respectively.
• At the beginning of interval $i\,(1 \le i \le I)$, artefact $j\,(1 \le j \le J)$ requires a total of w_{ji} person-hours to be completed (i.e. to make it ready for marketing). Denote by $\mathbf{w}_i$ the remaining work vector with elements w_{ji} $(1 \le j \le J)$. Note that $w_{ji} = 0$ if artefact j is completed at some interval before the beginning of interval i.
• At the beginning of interval $i\,(1 \le i \le I)$ we have M_i available person-hours that can be distributed arbitrarily among the set of (uncompleted) artefacts. Let a_{ji} be the fraction of available person-hours allocated to artefact j during interval i, and denote by a_i the *allocation* vector with elements a_{ji} $(1 \le j \le J)$. (Note that for any i, $\sum_{j=1}^{J} a_{ji} \le 1$.) Then the amount of person-hours allocated to artefact j during interval i is given by $a_{ji} M_i$, with $a_{ji} \le w_{ji}/M_i$. Note that if artefact j is completed at some interval before i, then $a_{ji} = 0$. It follows that the total work allocation (cost) during interval i is given by $M_i \mathbf{a}_i^{\mathrm{T}} \mathbf{e} \le M_i$, and that the remaining work on artefact j at the end of interval i (or at the beginning of interval $i-1$) is given by

$$w_{j,i-1} = w_{ji} - a_{ji}M_i = w_{jI} - \sum_{k=I}^{i} a_{jk}M_k, \quad 2 \le i \le I$$

where w_{jI} is the total work required to complete artefact j at the beginning of the time horizon.
• Let the vectors $\mathbf{d}_i$ and $\mathbf{w}_i$ define the state of the Markov decision process at the beginning of interval i. Then, the expected immediate return R_i at the end of interval i depends on the *state* vectors $(\mathbf{d}_i, \mathbf{w}_i)$ and the allocation vector $\mathbf{a}_i$; specifically,

$$R_i(\mathbf{d}_i, \mathbf{w}_i, \mathbf{a}_i) = \sum_{j=1}^{J} \Im(\{w_{ji} - a_{ji}M_i = 0\}) d_{ji} r_{ji}, \quad 1 \le i \le I$$

where the indicator function $\Im(.)$ equals 1 if its argument is true (i.e. artefact j is completed at or before interval i), otherwise it is equal to 0.
• Let $Y_i(\mathbf{d}_i, \mathbf{w}_i)$ be the maximum total expected reward at the end of interval i; then the optimality equations can be written as follows:

$$Y_1(\mathbf{d}_1, \mathbf{w}_1) = \max_{a_1}\left[\left(R_1(\mathbf{d}_1, \mathbf{w}_1, \mathbf{a}_1) - M_1 \mathbf{a}_1^{\mathrm{T}} \mathbf{e}\right)\right]$$

$$Y_i(\mathbf{d}_i, \mathbf{w}_i) = \max_{a_i}\left[\left(R_i(\mathbf{d}_i, \mathbf{w}_i, \mathbf{a}_i) - M_i \mathbf{a}_i^{\mathrm{T}} \mathbf{e}\right) + Y_{i-1}(\mathbf{d}_{i-1}, \mathbf{w}_{i-1})\right], \quad 2 \le i \le I$$

The above recursion can be applied to determine the optimum allocation vectors $\mathbf{a}_i$ for $i = 1, 2, \ldots, I$, respectively.

4.4 Scheduling implementation of framework

In this Section, we will apply our method to the example case. To this end, first insurance software system demands are estimated. Second, an insurance software system demand scenario is presented. Finally, based on this scenario, the optimal alternative in scheduling the implementation of the framework is determined.

4.4.1 Step 1: Estimations:

As explained in the following Sections, at this stage, three kinds of data are defined. First, expected products and sub-products are specified. Second, the insurance software system demands are estimated. Third, the available human resources and costs are specified.

Specification of the product demands: Assume that, according to the marketing study, there will be a substantial demand for *unemployment insurance* systems, *illness insurance* systems and *illness insurance with own risk* systems. These should be assembled through the composition of the necessary framework parts. In Fig. 2, the artefacts *framework parts* and *systems* are represented as rectangles. The directed relations among the rectangles indicate assembly dependence relations. For example, *unemployment insurance* systems are assembled from the framework parts *insured person* and *unemployment coverage*. The framework parts *insured person and unemployment coverage* are, in turn, assembled from *insured object* and *coverage*, respectively.

As shown in Fig. 2, every artefact has the attribute *implementation time*, which indicates the amount of time necessary to implement that artefact. The products have an additional attribute, *selling price*, which indicates the estimated market price of the system.

Market demand scenarios: In general, market demand scenarios cannot be determined with certainty. Therefore probabilistic estimations need to be made. Typically, standard normal distributions are used for this purpose. Consider, for example, the following demand estimations.

In Fig. 3, curves represent from left to right the estimations for market demand for *unemployment insurance* systems, *illness insurance with own risk* systems and *illness insurance* systems, respectively. The curve for *unemployment insurance* systems varies between ~16 and 24 units and concentrates around 20 units. The second distribution, which depicts the demand estimation for *illness insurance with own risk* systems, varies in demand between ~30 and 40 units. For this software system, a demand of 35 units will occur with the highest probability. Finally, the third distribution varies between ~38 and 43 units and concentrates around 40 units. For this example we will take the mean values for ease of computation. However, the

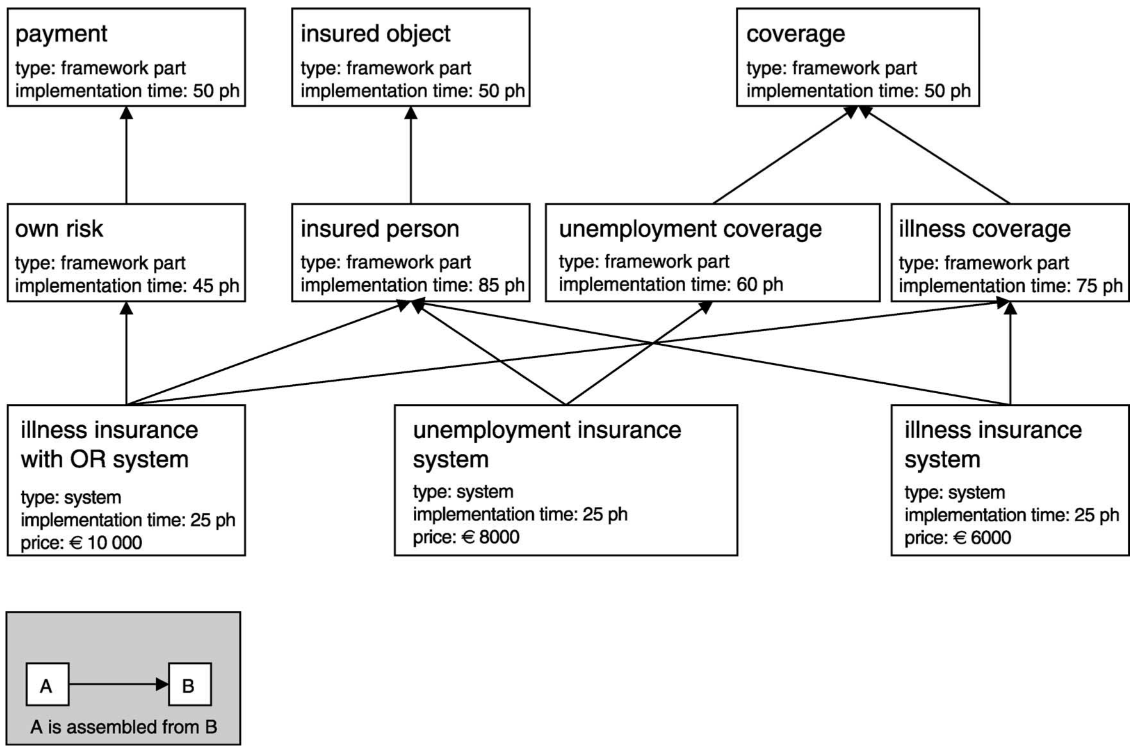


Fig. 2 *Dependencies among insurance software systems and framework parts*

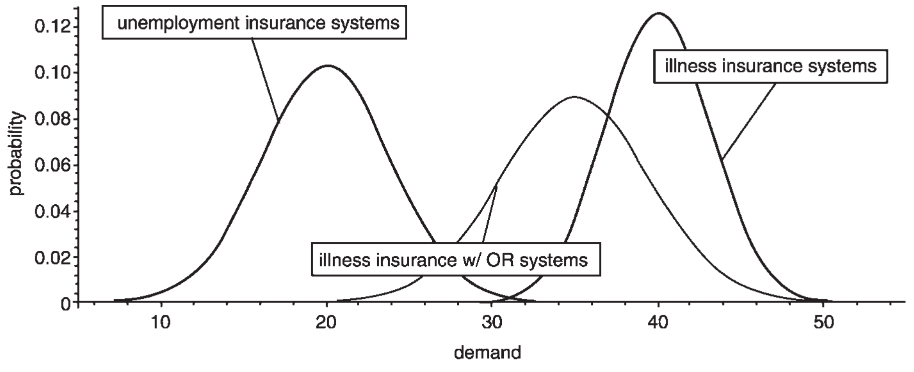


Fig. 3 *Market volume estimations for three kinds of insurance software systems*

computations can be made more accurate by taking the entire deviation interval.

Human resource definitions: Assume that two persons will be available in every week for up to 50 person hours each. It is also required that a person cannot be assigned to multiple tasks in one week. The cost per person hour is fixed to €100.

4.4.2 Step 2: Modelling the product demand scenarios:

To compute the optimum schedule, the Markov decision model as described in Section 4.2 has to be discretised. For the distributions this is done by taking the mean values. The mean values of the distributions of three insurance products are listed in the following:

- illness insurance systems 40 units
- unemployment insurance systems 20 units
- illness insurance with own risk system 35 units

In addition, for this example, we have discretised the amount of person hours that can be allocated as per week and per employee. Since a person can only work on one task per week, the smallest amount of person hours that can be assigned is 50.

4.4.3 Step 3: Selecting the most cost effective schedule:

Finally, all possible schedules are evaluated for every single scenario. For this purpose, schedules are modelled as a *sequence of decisions*, where each decision is taken at the beginning of a week. For instance, in the first week, it is possible to assign 100 person hours to any possible artefact. When a system is completed and there is a demand for it, it can be sold with a profit. Finding the best schedule is an optimisation problem as defined below:

Given

- system demand scenarios with probabilistic change dependencies
- a set of available person hours with a given cost price and minimal contract duration
- a set of framework parts *and/or* systems with expected implementation time
- estimated profit of a system when it is brought to the market

find a schedule which delivers the maximum profit.

The method is supported by a tool, which is presented in Section 5. Finding the optimal schedule is computed automatically and is based on Markov decision theory, which was presented in Section 4.2.

Table 1: Optimum allocation of resources for maximum profit

	coverage	insured object	payment	illness coverage	unemployment coverage	insured person	own risk	illness insurance system	unemployment insurance system	illness insurance/ w OR system	max. expected profit
Week 1	50	50	–	–	–	–	–	–	–	–	€14 000
Week 2	–	–	–	–	100	–	–	–	–	–	
Week 3	–	–	–	–	–	100	–	–	–	–	
Week 4	–	–	–	–	–	–	–	–	100	–	
Week 5	–	–	–	–	–	–	–	–	100	–	

Table 2: Mean values of system demands per week

	Illness insurance system	Illness insurance with OR system	Unemployment insurance system
Week 1	40	35	20
Week 2	30	42	31
Week 3	40	35	20
Week 4	30	42	31
Week 5	40	35	20

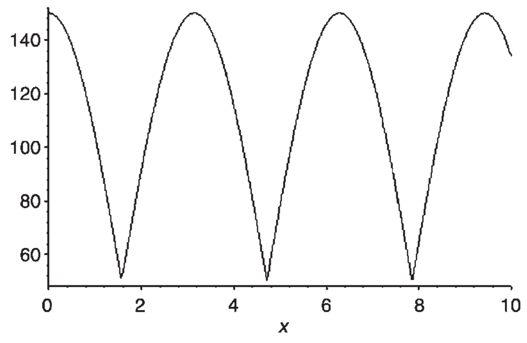


Fig. 4 *Periodic changes in human resources*

Table 3: Optimum allocation of resources in changing system demands and resources

	coverage	insured object	payment	illness coverage	unemployment coverage	insured person	own risk	illness insurance system	unemployment insurance system	illness insurance/ w OR system	max. expected profit
Week 1	50	50	–	–	–	–	–	–	–	–	€25 000
Week 2	–	–	–	–	100	–	–	–	–	–	
Week 3	–	–	–	–	–	100	–	–	–	–	
Week 4	–	–	–	–	–	–	–	–	150	–	
Week 5	–	–	–	–	–	–	–	–	100	–	

Results: The result of the optimisation algorithm can be displayed as a Table. In our example case, the tool provides several equivalent schedules that will result in the same expected profit. Table 1 shows one of the possible schedules.

Each row in Table 1 corresponds to a week in the project time span. The elements of a row contain the allocation of person hours for that week. For instance, in the first week, 50 person hours are assigned to the *coverage* artefact and 50 person hours to the *insured object* artefact. In addition, the model gives an indication of the *maximal expected* profit. Based on the given product demand scenarios and constraints, *unemployment insurance system* should be delivered at the end of the fourth and fifth week. Therefore, the relevant artefacts for these insurance software systems should be implemented first.

4.5 A more accurate market model

The example presented in the preceding Section can be extended in various ways. For example, in a more realistic scenario, the probabilities and available resources may change in time due to illness or holidays. In the following, as an illustrative example we will assume time dependent mean probability values and resource availability functions.

4.5.1 Market demand scenarios: The demands for each insurance software system may change in time. To simulate this, the market demand scenarios should include changing mean demands for every relevant project time step. For example, in Table 2, we make an assumption by modelling changes per week.

4.5.2 Human resource definitions: It is also possible to change the availability of human resources in time. Consider, for example, Fig. 4, which assumes a periodic change in the available human resources. This graph is approximated in the actual calculations by the following function [Note 3]:

$$50 + 50\,\mathrm{Round}(\mathrm{Abs}(2x\cos(t))).$$

In this Figure, it is assumed that the availability of human resources shows a periodic character. The presented changes in product demands and human resources should be considered as examples only. It is possible to use different functions and/or any meaningful data obtained, for example, from past experiences.

Note 3: *Round* represents the function that rounds real numbers to the nearest natural number, and *Abs* the function that returns the absolute value of a given number.

4.5.3 Results: Based on the time varying data, Table 3 displays one of the expected optimal schedules. This Table shows that, based on the assumptions, *unemployment insurance product* is expected to deliver most profit. Therefore, based on the Table, the relevant artefacts for this product should be implemented first. Note that in the first step only 100 person hours of the available 150 are distributed. Apparently within the current situation it is not wise to distribute the additional 50 person hours as it will lower the expected profit.

5 Tool support

The optimisation process is very labour intensive and therefore automatic support is necessary. In addition, tool support is necessary for modelling artefacts, market and available resources. The architecture of our tool is shown in Fig. 5. Here, the models and processes are represented as rectangles and ellipses, respectively.

Artefacts (framework parts and software systems), insurance software system demand scenarios and resources are modelled by the software engineer, marketing analyst and personnel manager, respectively. The personnel manager also defines the goals of the optimisation problem, such as minimisation of cost, maximisation of profit, etc.

The process *decision space generator* retrieves the necessary information from the artefact repository and the process *scenario space generator* retrieves the necessary information from the market model repository. Together they generate the data *scheduling state space*. The next step is the determination of scheduling advice. The process *schedule optimiser* takes *scheduling state* space as input and generates the data *ranked scheduling advice* as output. These data can be presented to the project manager in various formats, for example, as a Table shown in Table 3.

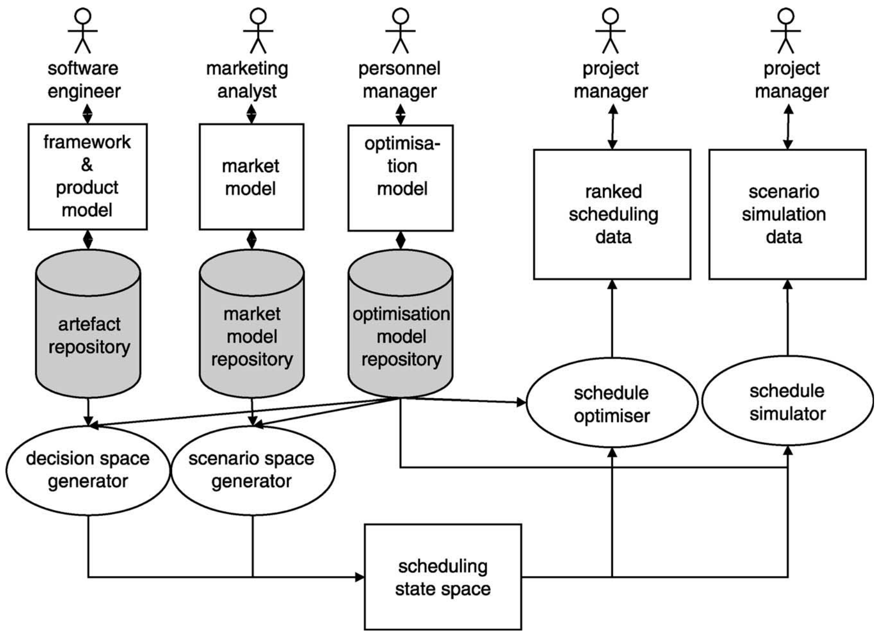


Fig. 5 *Tool architecture*

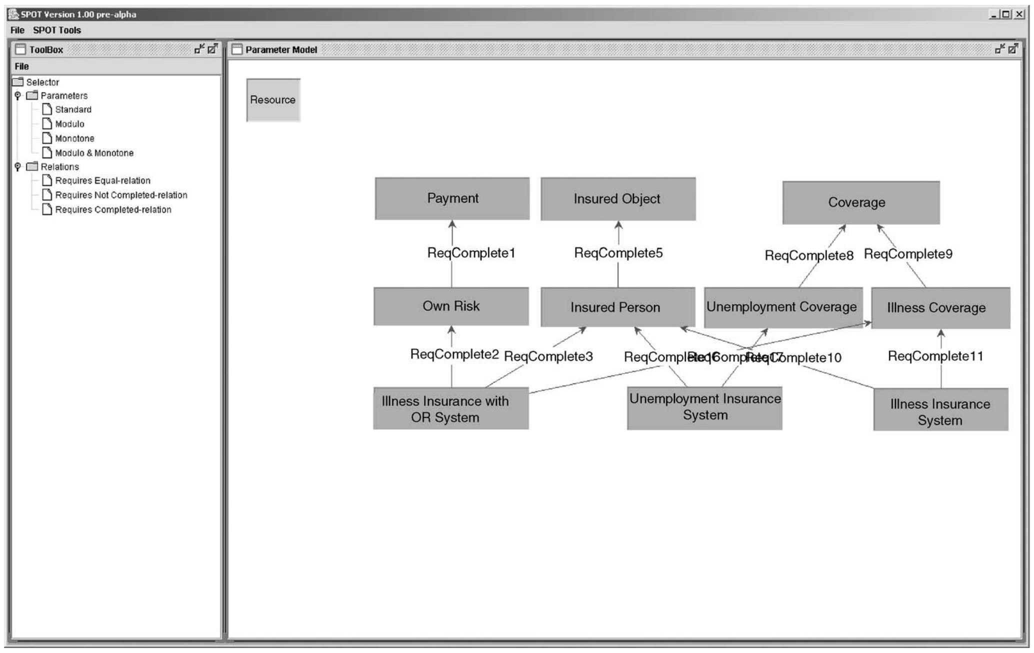
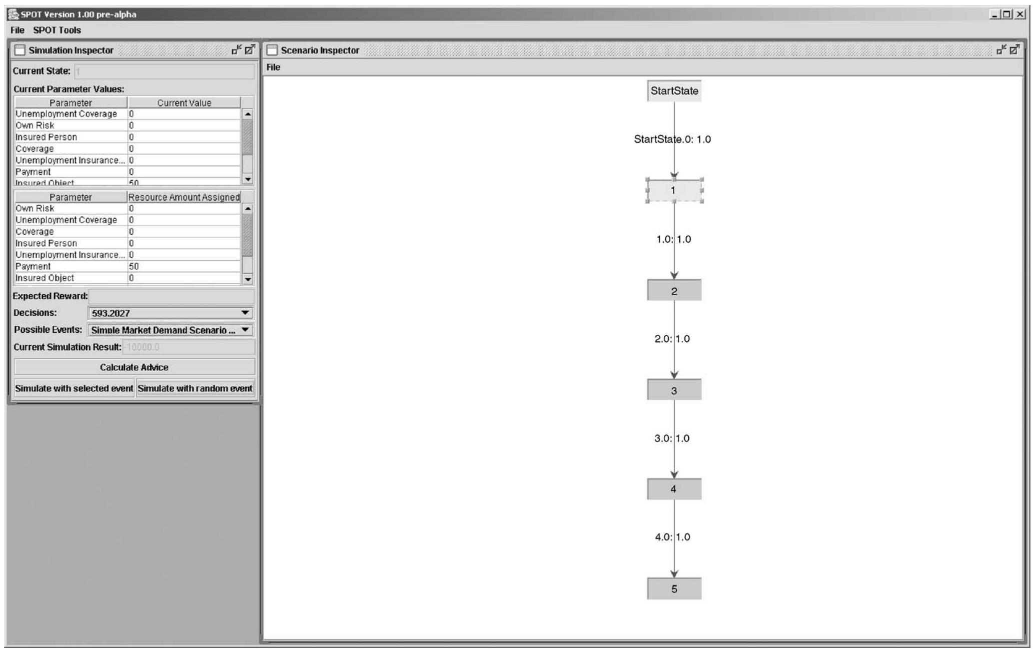


Fig. 6 *Modelling the insurance framework components*



Current State: 1

Current Parameter Values:

Parameter	Current Value
Unemployment Coverage	0
Own Risk	0
Insured Person	0
Coverage	0
Unemployment Insurance...	0
Payment	0
Insured Object	50

Parameter	Resource Amount Assigned
Own Risk	0
Unemployment Coverage	0
Coverage	0
Insured Person	0
Unemployment Insurance...	0
Payment	50
Insured Object	0

Expected Reward:

Decisions: 593.2027

Possible Events: Simple Market Demand Scenario ...

Current Simulation Result: -10000.0

Calculate Advice

Simulate with selected event | Simulate with random event

Fig. 7 *Running a simulation*

In addition, the project manager can run simulations by using the *schedule simulator*, which results in the data *scenario simulation data*.

First of all the *framework and product model* can be defined in the *parameter modeller*. In this tool all the parts can be modelled, which the system engineer can directly influence. In Fig. 6 the component definition of Section 4.3 is modelled using the *parameter modeller*.

As shown in Fig. 6, the sub-window on the left side shows the modelling tools. It is possible to define different types of parameters, corresponding to framework parts and actual products.

During the course of a project, a number of different situations may occur. To increase flexibility, if desired, the scheduling advice can be given on a per situation basis. The tool can simulate the situations that can occur for gaining a better insight into these situations. In Fig. 7 the *simulation tool* is displayed.

As shown in Fig. 7, the simulator tool consists of two sub-windows. The left sub-window is the simulation inspector and displays information on the current state of the simulation, such as profit result, amount of work done, etc.

The right sub-window displays the states that can be reached during the simulation. This graph is the scenario graph that models all possible market situations. The graph in Fig. 7 depicts a graph for which each state has only one possible next state and therefore represents only one scenario. While running the simulation the tool will colour the visited nodes yellow, so in Fig. 7 two nodes have already been visited.

The tool has been implemented on the Java Platform as a part of the SP^{2}OT Tool Suite, and is still under development. More information on the tool suite and the tool itself can be found at: http://janus.cs.utwente.nl/twiki/bin/view/Spot/WebHome/.

6 Evaluation

In Section 3.2, prioritising changing requirements and configuring the software development process accordingly were identified as two important problems. Given probabilistic product demand scenarios and resources, we have presented a method and a set of tools that can help project managers in determining the estimated optimum process development schedule that delivers the best profit. In the following, we evaluate our approach with respect to the following concerns:

- *Support for unanticipated evolution*: In general, unanticipated software evolution focuses on changes in software requirements that were not foreseen during the design of a software system. In our specific case we assume that changes in software requirements are known, but their occurrence is not certain. That is to say, we do not deal explicitly with uncertainty in both content and time of change as described in Section 2. However, our approach can be adopted for phenomenal potential evolution and temporal potential evolution. Our stochastic model explicitly reasons about the occurrence of requirements over a time span; as such, we can deal with uncertainty in time of change. In addition, every requirement is tagged with a probability and the model computes the occurrence and the change of these probabilities, and therefore we can deal with phenomenal potential evolution. Of course, more experimentation is necessary to determine the practical use of our approach.
- *Support for planning the software development process*: The presented approach provides a valuable input to the resource planning process in which the scarce resources have to be planned in a time span. The tool can be used at various levels of details. In addition, the tool provides flexibility in considering new data, for example, which may be available due to better market estimates. The tool will recalculate the optimal schedule from the time that new data are considered relevant. The probabilistic estimations cannot always be guaranteed to be accurate. Nevertheless, our tool may force the managers to explicitly consider various scenarios and experiment with 'what-if' conditions over the scenarios. This may help the managers to gain better insight in planning the software development processes.
- *Tool support*: Optimum resource allocation is a very difficult task to carry out manually because (a) various different input data are required; (b) it requires evaluation of a large space of alternatives, and (c) input data may change

continuously and therefore evaluation has to be repeated frequently. We have developed a set of tools for: (a) defining models, (b) specifying probabilistic estimations, (c) computing optimal schedules at various levels of details, and (d) displaying and inspecting the results in various formats. We have already started experimenting with the tool in different applications. By providing the tool for public use, we hope to obtain the necessary feedback necessary for improving the tool where necessary.

- *Support for vague market characteristics*: In general, short-term market estimations can be made with reasonable accuracy. For a time span longer than 2 years, simple probabilistic estimations will be less reliable. This problem can be partially addressed by using adaptive learning techniques so that estimations can be improved continuously and by defining more accurate market models, for example, based on fuzzy logic techniques.
- *Support for simulation and educational purposes*: One important perspective for our tool is to use it for simulation and educational purposes. Because the tool essentially generates all the possible and relevant situations during a project, the state space can be used to examine specific project situations. In addition, by fixing market expectations at specific time points, scenarios can be simulated, for instance for determining worst-case scenarios. We are currently co-operating with a large software company in adopting the tool for consulting activities and in management courses, etc.
- *Human resource model*: Our approach is sensitive to the accuracy of the human resource model. In addition, it is assumed that every employee can perform any task, if the time permits. A more realistic approach would be to categorise human resources with respect to their skills. This adds an extra level of dependency that needs to be considered when scheduling human resources. We are currently working on a more realistic human resource model, which improves the model properties based on the experiences obtained in the past.
- *Artefact model*: In the current approach, artefacts can only be produced when all the required sub-artefacts have been delivered. By introducing a more fine-grained model, we would like to minimise the dependencies among the artefacts and thereby improve parallel allocation of resources.
- *Scalability of approach through managing state explosions*: To calculate the optimal result, Markov decision models require all relevant states. This can cause very large state spaces and lengthy computations. This is known as the *curse of dimensionality* [17]. To address this problem, within the field of Markov decision theory, various research activities have been carried out. Examples are state space minimisation techniques by assessing the relevancy of states and by making tradeoffs between accuracy of the result and computation size, or optimisation for parallel computing [18–20]. We are currently experimenting on various state-space reduction mechanisms.

7 Background and related work

7.1 Software process configuration management

To cope with the constantly changing customer requirements, software products and software process must be re-configured frequently. Software configuration management aims to manage the evolution of a product. Our approach can be seen as complementary to software configuration management techniques.

Software configuration management is a broad domain and covers the entire life cycle. Further, software configuration management processes can be defined for specific domains as well. For example, in [13] an approach is presented for resource management for distributed multi-agent systems. This approach takes into account a wide range of resource and entity types. By providing different types of relations between entities and resources, the actual situation can be described and optimised accurately. The approach in [13], however, is not aimed to optimise with respect to future changes that are uncertain. Our approach aims to schedule the available resources while considering probabilistic changes in requirements and project context.

In addition to software configuration management, several researchers have focused on the configuration of processes. The basic issues are to choose the right development process and to align process configuration whenever necessary. In [21, 22] it is argued that software development processes should be developed in much the same way as software. The common assumption is that processes should be considered as products and be configured with respect to product quality goals. Our approach can be considered complementary to these approaches. Our focus is, however, to cope with changes in requirements and project context, which is generally missing in the conventional software process configuration approaches.

7.2 Requirements engineering

Prioritisation of requirements with respect to implementation is not new. There have been various research activities on this topic. However, research activities generally focus on fixed requirements; prioritisation is realised after the requirements are determined.

The work described in [23] focuses on prioritisation of software requirements with respect to the quality of decision-making. The basic assumption here is that new requirements should either be accepted or rejected. By analysing and simulating the acceptance/rejectance rate, the decision quality can be improved and only relevant requirements are selected for further consideration.

The difference with our approach lies in the fact that the approach in [23] acts as a filter and results in a set of requirements that should be implemented. The remaining relevant requirements are also considered equally important. However, these requirements still need to be prioritised with respect to implementation and resources, since there still might be dependencies among them. Also the approach focuses on the best requirements set for the next release, while our approach is more aimed at finding optimal schedules for implementation with respect to changes along the time span of a project.

In [10] a methodology is proposed, which prioritises requirements based on a cost–value analysis. For each requirement, the relative cost and value are determined using the analytic hierarchy process. This model then allows comparison of the requirements based on these properties, so that one can be ranked over the other with respect to these two aspects. By considering the requirements of engineers, customers, users and software engineers an accurate requirements model can be made.

The proposed methodology in [10] focuses on prioritisation with respect to value and cost, and with that it aims to satisfy market demands in general. The methodology does not explicitly address the delivery constraints that are imposed during software development process. Also the model does not support analysis of requirements that share dependencies. In our approach this is specifically addressed by the framework dependencies.

7.3 Optimisation techniques

Many different optimisation techniques have been defined, most of them having a mathematical origin. Generally speaking an optimisation model consists of four parts: a subject to be optimised, the options to be considered, a comparison criterion and an ideal situation description (or goal). The possible options are evaluated with respect to their goal using the criterion. Dynamic programming [16, 17] is one of these techniques but many others exist. For instance, there are learning-based optimisation models such as neural networks [24] or genetic algorithms.

Most optimisation techniques are based on evaluation of the available options. Brute force approaches tend to evaluate every single option and then select the optimal one. However, this soon leads to incomputable problems. Therefore most optimisation techniques aim at 'smart' evaluation of options, thus reducing computation complexity. A vital step in the optimisation process is to map the problem to a mathematical formalism. By choosing a smart mapping, the complexity of the problem can be kept at an acceptable level.

8 Conclusions

It should be noted that the management decision problem is indeed a very challenging problem and it is difficult to claim that we have provided the complete and final solution. To fully prepare for unanticipated software evolution seems to be impossible by definition.

Nevertheless we have shown that even unanticipated software evolution can be managed to a certain degree. For this, we have made an explicit distinction between unanticipated software evolution due to uncertainty in content and uncertainty in the time of occurrence of changes to requirements. Conventional software evolution approaches have basically focused on evolution in content of the requirements and several useful techniques have been proposed. As a complementary approach we have focused on coping with evolution of changes to requirements in time.

The presented approach represents the possible development activities as a state space using Markov decision theory. By applying dynamic programming techniques the optimal schedules of software development activities are determined. The required changes are modelled using probabilistic estimations on product demands, profits and available resources. Owing to the large state space it is very hard to manually perform this approach. Therefore, we have implemented a tool which supports the modelling, the selection and the evaluation of the changes to requirements and the allocation of human resources.

The approach and the related tool explicitly forces us to think about priorities and risks of decision making with respect to changes in requirements. To improve the scalability of the approach our future work will include the refinement of the human resources model and the integration of more refined state-space reduction techniques.

9 Acknowledgments

We would like to thank the anonymous reviewers for their valuable feedback on the initial version of this article.

10 References

1 Kniesel, G., Noppen, J., Mens, T., and Buckley, J.: 'Unanticipated Software Evolution'. Proc. European Conf. for Object Oriented Programming (ECOOP), Malaga, Spain, 10–14 June 2002
2 Lethbridge, T., and Laganière, R.: 'Object-oriented software engineering' (McGrawHill, Berkshire, UK, 2001)
3 Yonezawa, A., and Matsuoka, S.: 'Metalevel architectures and separation of crosscutting concerns'. Proc. 3rd Int. Conf. on REFLECTION, Kyoto, Japan, 25–28 September 2001
4 Duggan, D., and Wu, Z.: 'Adaptable objects for dynamic updating of software libraries'. Presented at 1st Unanticipated Software Evolution (USE) Workshop, European Conf. for Object Oriented Programming (ECOOP), 10–14 June 2002
5 Dmitriev, M.: 'Hotswap technology application for advanced profiling'. Presented at 1st Unanticipated Software Evolution (USE) Workshop, European Conf. for Object Oriented Programming (ECOOP), 10–14 June 2002
6 McGurren, F., and Conroy, D.: 'X-ADAPT: an architecture for dynamic systems'. Presented at 1st Unanticipated Software Evoluation (USE) Workshop, European Conf. for Object Oriented Programming (ECOOP), 10–14 June 2002
7 Anderson, C., and Drossopoulou, S.: 'An imperative object based calculus with delegation'. Presented at 1st Unanticipated Software Evolution (USE) Workshop, European Conf. for Object Oriented Programming (ECOOP), 10–14 June 2002
8 Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A.: 'Feature oriented domain analysis (FODA) feasability study'. CMU/SEI-90-TR-21 ESD-90-TR-222, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, November 1990
9 Jacobson, I., Booch, G., and Rumbaugh, J.: 'The unified software development process' (Addison Wesley, Reading, MA, 1999)
10 Karlsson, J., and Ryan, K.: 'Prioritizing requirements using a cost-value approach', *IEEE Software*, 1997 (September/October), **14**, pp. 67–74
11 Salo, A., and Hämäläinen, R.: 'On the measurement of preferences in the analytic hierarchy process', *J. Multi-criteria Decis. Anal.*, 1997, **6**, pp. 309–319
12 Forman, E., and Sally, M.: 'Decision by objectives' (World Scientific, River Edge, NJ, 2001)
13 Podorozhny, R., Staudt Lerner, B., and Osterweil, L.: 'A rigorous approach to resource management in activity coordination' (University of Massachusetts, Amherst MA 01003, USA, 1999)
14 Sullivan, K., Chalasani, P., Jha, S., and Sazawal, V.: 'Software design as an investment activity: a real options perspective', in Trigeorgis, L. (Ed.): 'Real options and business strategy' (Risk Books, London, UK, 1999)
15 Tran, T., and Sheriff, J.: 'Quality function deployment: an effective technique for requirements acquisition and reuse'. Proc 2nd IEEE Software Engineering Standards Symp., Montreal, Canada, 21–25 Aug. 1995, p. 191
16 Puterman, M.L.: 'Markov decision processes, discrete stochastic dynamic programming' (Wiley-Interscience, Hoboken, NJ, 1994)
17 Bellman, R.: 'Adaptive control processes: a guided tour' (Princeton University Press, Princeton, NJ, 1961)
18 Chung, S., Hanson, F., and Xu, H.: 'Parallel stochastic dynamic programming: finite elements methods', *Linear Algebr. Appl.*, 1992, **172**, pp. 197–218
19 Larson, R.: 'Dynamic programming with reduced computational requirements', *IEEE Trans. Autom. Control*, 1965, **14**, pp. 135–143
20 Larson, R.: 'State increment dynamic programming' (American Elsevier, New York, NY, 1968)
21 Osterweil, L.: 'Software processes are software too, revisited: an invited talk on the most influential paper of ICSE 9'. Proc. 19th Int. Conf. Software Engineering, Boston, MA, May 1997, pp. 2–13
22 Osterweil, L.: 'Architecting processes are key to software quality'. Presented at Int. Workshop on the Role of Software Architecture in Testing and Analysis, Marsala, Italy, 30 June–3 July 1998
23 Regnell, B., Karlsson, L., and Höst, M.: 'An analytical model of requirements selection quality in software product development'. Presented at Conf. on Software Engineering Research and Practice in Sweden (SERPS), Karlskrone, Sweden, 24–25 October 2002
24 Haykin, S.: 'Neural networks: a comprehensive foundation' (Prentice Hall, Upper Saddle River, NJ, 1998)