Planning Support to Software Process Evolution

Reidar Conradi*, Minh Ngoc Nguyen, Alf Inge Wang
Norwegian University of Science and Technology (NTNU),
Trondheim, Norway

Chunnian Liu †
Beijing Polytechnic University (BPU),
Beijing, P.R. China.

Abstract

The ability to handle changes is a characteristic feature of successful software projects. The problem addressed in this paper is what should be done in project planning and iterative replanning so that the project can react effectively to changes. Thus the work presents research results in software engineering, as well as transfer of methods in knowledge engineering to software engineering, applying the AI planning technique to software process modeling and software project management. Our method is based on *inter-project experience* and *evolution patterns*. We propose a new classification of software projects, identifying and characterizing ten software process evolution patterns and link them to different project profile. Based on the evolution patterns, we discuss the planning support for process evolution and propose several methods that are new or significantly extend existing work, e.g. **cost estimation of process changes, evolution pattern analysis**, and **a coarse process model for the initial planning- and the iterative replanning process**. The preliminary results have shown that the study of evolution patterns, based on inter-project experience, can provide valuable guidance in software process understanding and improvement.

Keywords: *Software Process Modeling, Software Process Evolution, measurements, planning.*

1. Introduction

Coarsely, we have four kinds of changes/evolution in software processes. First, we have two kinds of changes in a normal development process, namely delayed planning (e.g. of design details), or replanning due to unforeseen events (e.g. changes in requirements or in staffing). Due to the unstructured nature of many of these changes, traditional project management tools are grossly insufficient [6]. A third

*Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), N-7034 Trondheim, Norway. Phone: +47 73 593444, Fax: +47 73 594466, Email: conradi@idi.ntnu.no

†Chunnian Liu's work is partly supported by the Natural Science Foundation of China (NSFC), Beijing Municipal Natural Science Foundation (BMNSF) and Chinese 863 High-Tech Program.

kind of change occur in a later maintenance process, where a stream of change requests are being handled. These changes can be characterized as perfective (50%), corrective (21%) adaptive (25%) and preventive changes (4%) [12]. Lastly, a fourth kind of change is the more long-term *software process improvement (SPI)*, which implies goal-driven and systematic evolution of the entire software process between projects, cf. CMM [21] and Experience Factory [3].

So the scope of research in software process evolution is vast. The particular problem addressed in this paper is what should be done in project planning so that the project can react rationally and effectively to changes. Thus, we focus on the first two kinds of changes, with some emphasis on the fourth (i.e. SPI).

We propose a framework of project profiles, process evolution patterns observed in projects, and the relation between project profiles and evolution patterns. Then we discuss the planning phase in depth, aiming to provide the project manager with method support in configuring a more realistic and flexible project plan. This includes initial planning before the start of the project and iterative replanning during the execution of the project. The method is based on inter-project experience and the evolution patterns. For a new project, a matching baseline project is selected from an Experience Database to predict the potential change patterns that are likely to occur in the new project. Based on this information, the project manager can make more appropriate decisions in the project plan so that the project is better prepared for the potential changes. These decisions will be followed and/or revised, when changes do occur during the execution of the project. We also present some preliminary, experimental results - including an industrial case study - which covers part of this paper.

This work presents not only research results in software engineering alone, but also transfers methods of knowledge engineering to software engineering, applying the AI planning technique to software process modeling and software project management.

2. Related Work

Lehman [11] has presented a classification of software projects and process evolution, listing five evolution entities: Software Releases, Software Systems Under Development, Application Domains, Development Process, and the Process Model. His paper also describes the evolutionary characteristics of each of these entities. However, his discussion is in a very abstract manner.

On the other hand, Nguyen and Conradi propose a functional framework (neither too abstract, nor overly complex in operational details) to classify process evolution and link evolution patterns to different project profiles [19]. The evolution characteristics are: where (origin), why (cause), what (process element), when (phase), how (kind of change), by-whom (modifier). All these characteristics can be subclassed.

The original EPOS Planner [14] exploits a mixture of hierarchical and linear planning [1] to generate parts of the task network in a process model. In [15], Liu and Conradi present an incremental replanning algorithm to deal with "microscopic" changes in the process model. For example, the replanning algorithm can adjust an existing task network to reflect changes in the Product Structure and/or in

task types (called the template process model). But in real processes most changes with major impact on the process quality and improvement are "macroscopic"; see [20]. Our previous work on replanning was also confined to individual projects, and inter-project experience on process evolution was not considered. The (re)planning method presented here is based on inter-project experience and more general evolution patterns.

In the domain of *software experience databases*, we have mentioned the Experience Factory improvement method developed at NASA-SEL and Univ. Maryland. The Experience Factory approach proposes a centralized improvement taskforce in a company, with long-term and company-specific SPI as a goal, and is internally exploiting an experience database. There is also the associated TAME project [4] to exploit reusable process artifacts in an experience database, e.g. to make decisions about subsequent projects within the same organization. Further, much work on experience databases and corporate memory as a vehicle for for organizational learning has been done, especially in the engineering sector. This has often been accompanied by techniques such as statistical analysis, data mining like that provided by rough-sets [22], or AI-oriented methods like case-based reasoning [10]. Finally, there is also work on the use of OO-style patterns to represent reusable classes of projects [8].

Madachy has implemented a knowledge-based system [16] for risk assessment and cost estimation. More recently Konito in [9] proposes a systematic method for risk management, where risk is seen as variance or degree of imprecision. The idea of risk scenarios (the hierarchy of risk factor – risk event – reaction) is adapted in this paper to systematically make planning decisions about how to deal with changes which are likely to occur in the project; see Section .

3. Classification of Process Evolution

3.1. *Classification of Projects*

First we observe that it is not realistic to talk about evolution patterns for software projects in general without linking them to different profiles of projects. Typical *overall-profiles* may cover aspects such as *cost-critical*, *time-critical*, *safety-critical*, *research-oriented* etc. – and with sub-profiles. Figure 1 shows a top-level classification of software projects, called *overall-profiles*, in which a list of items (*purpose, application domain, software product, main concerns, typical changes, main risk*) are used to characterize the overall profiles.

We do not claim that Figure 1 shows a complete list. Also, sub-classification is possible and may be necessary. For example, each of overall profiles A-D in Figure 1 can be further classified according to the following characteristics among others:

Application domain: Telecom, Business-Processing, Military, MIS, Banking, *Novelty*: release *vs.* new software; familiar *vs.* new application domain, and *Software maturity level of the organization.*

In addition to the overall-profile of projects, we have an *operative-profile* with categories such as: *project manager, participants, contractual-constraints, project size, external-suppliers, customers, operating-platform, implementation-languages, etc.* Each part of the operative-profile is assigned a corporate-specific *weight*, rep-

Projects / descr. / Charact.	A	B	C	D
	classical time/budget constrained projects	**time-critical projects**	**safety-critical projects**	**research prototyping projects**
purpose	develop/maintain application software in a particular domain	occupy market of a public software	develop/maintain applic. software in safety-critical domains	test new concepts and techniques, using extensive prototyping
application domain	commerce, industry, medicine, communication, ...	public sector etc.	transportation, military and space, nuclear, real-time control, ...	science, computer science, software technology, process improvement
software product	application software tailored software	system software (languages, OS, DBMS, Network,.....)	application software tailored software	prototype software to test/validate new ideas, algorithms, methods, ...
main concerns	run the project within budget, deliver software in time, user satisfaction	put the product to market as soon as possible	no critical errors, remaining errors are as few as possible	validity of new methods and comparison with existing ones
typical changes	external: requirements, resource internal: review/testing	external: market internal: requirement, design	external: error report internal: testing	internal: specification
main risk	schedule delayed, budget overrun, user complaints	release delays	critical errors causing serious losses of life/money	very low

Figure 1: Classification of Project Overall Profiles.

resenting the relative importance of the part in determining the similarity between various projects of the corporate. For example, a software house XXX may assign a high weight on *contractual-constraints*, as it learned that projects with matching constraints behave similarly.

When we speak of project profile, we mean both overall- and operative-profile. In the following, we assume that the Experience Database has been initialized with data from a non-trivial number (20-50) of historical projects, according to the profile classification described above. Linked to these project descriptions, we have attached template process models that have been used in these projects. Such templates constitute a repertoire of reusable process model parts that can be combined and instantiated into operative process/project models. In such an Experience Database we also store estimation/risk models for project planning, and quality models e.g. about defect densities.

3.2. *Evolution Patterns and their Relation to Project Profile*

In our approach, a *evolution pattern* is recognized by the following five steps:

1. Set up a framework with a hierarchy of characteristics describing various aspects of change. In this paper we use the following top-level attributes, being an extension of those in Minh's PhD thesis [18] (Figure 2):

   - *Source of change*: internal (triggered by feedback), or external (triggered by change of input).
   - *Scope of change*: local (microscopic), or global (macroscopic).
   - *Timing of change*: in early or late phases of the process.
   - *Evolving entity*: application-domain /environment /product /process /process-model.
   - *Forward-action*: actions taken in the forward path of the process.
   - *Backward-action*: actions taken in the backward loop (feedback loop).
   - *Impact on schedule etc.*: big/moderate/small impact on schedule/quality/cost.

2. Keep track of change occurrences over a large number of projects in an organization, record them in a form complying with the framework, and store the data into an Experience Database of the organization.

3. Set up criteria to classify changes. Here two recorded changes with the same attribute values are considered "identical."

4. Retrieve and analyze the evolution data stored in the Experience Database, grouping similar changes with only "small differences" according to some criteria.

5. Each group of *similar changes* with significant number of change occurrences (say, more than 5% of the total number of changes being analyzed) can be seen as an *Evolution Pattern*.

Figure 2 shows some common evolution patterns. For each pattern we give the "normal" value for each of the attributes. Each evolution pattern contains the following information: The *occurrence frequency* of a certain change in the project; The *explicit/implicit reactions* taken to deal with the change; The *impact* of the change on schedule, cost and quality (the risks); And the associated *metrics* (measuring procedure, data attributes, analysis model and analysis results). Naturally, such evolution patterns are also recorded in the Experience Database, and their usage are described in Section on planning.

The relation between project profile and evolution patterns could be extracted from the Experience Database, or a model of this relation could be validated/invalidated by the data in the database. Figure 3 shows an example of such a relation, linking project overall-profiles and evolution patterns.

4. Planning Support for Software Process Evolution

Our method is based on *inter-project experience* and *evolution patterns*.

Each previous project will have its overall/operative-profile recorded in the Experience Base (EB). As we mentioned in Section , items in the operative profile include: *project manager, participants, contractual-constraints, project size, external-suppliers, customers, operating-platform, implementation-languages, etc.*, and each item of the profile is assigned a corporate-specific *weight* to express its importance in measuring the similarity between projects.

A new project is initiated with its own overall/operative- profile. The project manager uses it as the criteria to search the Experience Base for those previously completed projects that are similar to the current project. Project similarity can be discovered by partial matching between the corresponding profile items. These similar projects constitute the candidates to be considered as the baseline for the current project. The best-matching candidate, *i.e.* having the highest sum of the *weighted* attributes associated with the matched measures, is then selected as the baseline project. Of course in practice, the choice of the baseline project is eventually a human decision, but the above method can provide some automatic help in the decision-making process.

Furthermore, the evolution patterns of the baseline project, which were recognized using the method given in Section , were also recorded in the Experience Base (EB). Retrieving this information, evolution patterns become predictable for the current project.

In this section, we will discuss how a project manager can utilize the evolution information of the baseline project to configure a more realistic, flexible plan for the current project. He/she is then expected to be better prepared for potential changes, which can be derived or predicted from the baseline project.

Project planning consists of initial planning and iterative replanning activities.

With our approach, the initial project plan will include the following essential items (besides general descriptions of project objectives, constraints, organization, resource, etc.):

1. *Task network*: This expresses the hierarchy of and dependencies between activities. In the initial plan, only a rough network can be provided. This is then refined/modified during the life-time of the project.

Charact / Values / Patterns	Source of change	Scope of change	Timing of change	Evolving entity	Forward-action	Backward-action (feedback)	impact on schedule,cost, product quality etc.
Pattern I requirement change	external	macro	early phase (usually)	appl.domain, product, process	re-schedule forward path	re-analysis, re-design, re-implem.	early phase: moderate later phase: big
Pattern II market change	external	macro	any time	environment, product	re-schedule forward path	re-analysis, re-design, re-implem.	early phase: moderate later phase: big
Pattern III customer delay	external	macro or micro	early phase (usually)	environment, process	re-schedule forward path	none	risk to schedule: depends
Pattern IV sub-contractor delay	external	macro or micro	later phase of process	environment, process	re-schedule forward path	none	risk to schedule: depends
Pattern V resource re-allocation	external (internal to organ.)	macro	any time	environment, process	re-schedule forward path	none	risk to schedule: big
Pattern VI new development technology infusion	internal	micro	late phase	process, process-model	improvement in forward path	negative feedback to constraint	global impact is constrained by negative feedback
Pattern VII new process technology infusion	internal	macro	early phase	product, process, process-model	improvement in forward path	negative feedback to constraint	global impact is constrained by negative feedback
Pattern VIII internal schedule adjustment	internal	macro or micro	any time	process	re-schedule forward path	none	moderate
Pattern IX review-modify cycle	internal	micro	early phase	product, process	wait till termination of the cycle	re-design, re-analysis	moderate
Pattern X test-modify cycle	internal	micro	late phase	product, process	wait till termination of the cycle	re-implem., re-design, re-analysis	moderate

Figure 2: Classification of Evolution Patterns.

Project / frequency of ocurrence / Evolution	**Type A** cost–critical	**Type B** time–critical	**Type C** safety–critical	**Type D** proto–typing
Pattern I requirement	moderate	high	low	high
Pattern II market	moderate	high	low	none
Pattern III customer	moderate	none	low	none
Pattern IV sub–contractor	moderate	low	low	none
Pattern V resource	low	low	low	low
Pattern VI dev. technology	low	high	low	high
Pattern VII process technology	low	low	low	low
Pattern VIII schedule	moderate	high	high	moderate
Pattern IX review	high	low	high	low
Pattern X testing	high	moderate	high	low

Figure 3: Relation Between Project Overall-Profiles and Evolution Patterns.

2. *Schedule*: The allocation of start/end time and various resources to each activity, and a set of intermediate milestones, (presented by a set of charts).

3. *Cost estimation*: The estimated effort of the whole project, as well as of each activity. In our case, the cost of possible changes should also be estimated.

4. *Evolution pattern analysis*: Based on the anticipated evolution patterns from the baseline project, try to actively change the project profile and/or improve the the process model. If this is not feasible, set appropriate contingencies (extra activities/time/resource) in the project plan to deal with possible changes.

5. *Metrics*: The relevant attributes and associated procedures to collect/analyze them. The actual measures will be used for risk analysis, plan revision, experience learning for future projects, etc.

We first present in Figure 4 our coarse process model (as pseudo-code) of the project planning process, significantly extending the one given in [26]. The planning process has four steps, S1-S4, where we will focus on step S1 "Initial Planning" and S3 "Replanning." The following subsections will describe our methods for each of the plan items, focusing on the core problem: "How to deal with possible changes". The resulting plan will be "better" than the ones which are based on a particular individual's experience or on some algorithmic models. It is "better" because it

is based on the whole organization's experience and is pre-prepared for possible changes.

4.1. *Task Network Layout and Scheduling*

The task network shows the task hierarchy (sub-tasking) and inter-dependencies. Initially, only a rough task network can be created. During project execution (e.g. in the design phase), high-level tasks are gradually decomposed into sub-tasks in a hierarchical way.

The project schedule should fill in start/end time and resource allocation for each task, all milestones representing important stages in the project, and the *critical path* – the longest path in the task network. Scheduling is based on cost estimation which is the subject of the next subsection. The project manager must consider possible scheduling changes. The following guidelines apply:

1. Schedule first as if nothing will go wrong, then increase the time periods and resources for tasks affected by anticipated changes (see Section).

2. Estimates in the initial project schedule must be compared with actual elapsed time in order to revise the schedule for the later parts of the project, even re-partition the later parts to reduce the length of the critical path (see Section).

3. To react to a change during project execution, the layout of the current network may need to be modified (adding/deleting/modifying tasks). In most cases, the modification is a human decision (such as adding an extra review task to enhance reliability), and can only be done manually (with some help of graphical editing tools).

4.2. *Cost Estimation*

Project cost reflects efforts, training, equipment, and so on. Here we only consider the *efforts costs* – the costs of paying project staff according to hours spent and their price.

Initial cost estimation is necessary to negotiate the project budget and the product price, to set the periods and resources for each task in scheduling. Continuous cost recording is later needed to ensure that spending is in line with budget (often being the estimate). Cost estimation for changes is necessary to effectively manage the process evolution.

Our overall method for cost estimation is:

- *Using experience data to help traditional cost estimation*:

  There is extensive literature on this, describing two kernel techniques of software cost estimation: *Estimation by analogy* (often manual) and *Algorithmic cost modeling* (often automatic). Our method is helpful for both. Estimation by analogy is based on completed projects of the organization and in same application domain (the baseline); Algorithmic estimation models such as COCOMO [5] have to be tuned to the need of a user organization using historical project data. In our approach, the baseline project and historical measures

S1. Initial Planning

Define project constraints
Estimate initial project profile
Estimate project cost (whole and for each phase) (Section 4.2)
Define project milestones and deliverables
Generate automatically the top−level task network (Section 4.1)
Make the initial schedule (task break−down, period/resource allocation, Section 4.1 and 4.2)
Evolution pattern analysis: (Section 4.3)
Whenever possible and desirable
Try to change the project profile and/or the process model, adapted to evolution patterns
Otherwise
Put contingencies in the project plan
Make sub−plan for measurements and risk analysis (Section 4.4)

S2. Enact the tasks according to schedule

S3. Continuing, iterative, (re)Planning

while {project is not completed}
Task Decomposition: (Section 4.1)
Decompose high−level tasks during the execution
Whenever possible, use the automatic Planner
Otherwise use Graphic Editor to do so manually
Tracking and Monitoring: (Section 4.4)
Track progress
Collect and analyze the collected measures according to the measurements sub−plan
Analyze Risks: (Section 4.4)
if (the risk is high)
then revise estimate
revise schedule **or**
re−negotiate project constraints/deliverables
end if
Deal with Changes: (Section 4.3, 4.1 and 4.2)
if (a change scenario occurs)
then execute the reaction of the scenario
record the occurrence of the scenario
elseif (unexpected changes occur)
then deal with it based on the manager's judgement
record this new evolution pattern/scenario/occurrence
end if
end while

S4. Experience Packaging and Learning: put back into Experience Database

Figure 4: The coarse Process Model of Project (re)Planning Process.

are both stored in the Experience Database and available to the manager of the new project.

As a minimal example, using the COCOMO model, we have the following estimation steps and results for a project implemented in C++:

1. Calculate the number of *function points* FP, e.g. 150 (estimated in requirement analysis or perhaps in high-level design);
2. Code size $LOC = AVC * FP = 7500$, where $AVC = 50$ for C++;
3. $Effort = 2.4 * LOC^{1.6} = 20$ person-months, where constants 2.4 and 1.6 are tuned using historical data, including documentation and testing;
4. Project duration is $2.5 * Effort^{0.38} = 7.8$ months, i.e. the project will be carried out by a 3 person-team (again, with constants 2.5 and 0.38 are tuned using historical data).

- *Cost estimation of process changes*:
  - The baseline project contains historical cost data for each change pattern. These data can be used to estimate the cost of similar changes in the new project. This method can be applied to the review-modify cycle (`pattern IX`), test-modify cycle (`pattern X`), and other evolution patterns.
  - For a new requirement, COCOMO or other models can also be used to estimate the cost to satisfy the requirement, e.g. expressed as FPs.
  - For requirement modification, the cost depends on the timing of the change occurrence. Modifications occurring in early phase cost little, as the real implementation work has not started yet. Changes occurring in later phases, however, may cause costly profound re-design and re-implementation. Requirements traceability techniques [13] can be used to identify the affected software modules to estimate the efforts of such re-work.

4.3. *Evolution Pattern Analysis*

To achieve real SPI, the project manager should analyze carefully the anticipated evolution patterns from the baseline project, and try to actively change the project profile and/or improve the the process model before the project starts. For example:

- If an evolution pattern indicates intrinsically unstable user requirements, we may choose an incremental development-delivery model with "time-boxing" [24] to replace e.g. the waterfall model employed in the baseline.
- If an evolution pattern indicates a high defect rate, we may introduce more rigorous inspection tasks in the project plan to reduce the risk in testing phase.
- If an evolution pattern indicates high people turn-over, we need to prioritize the over-crowded assignments in the company's pipeline (an improvement of the organization and the project profile).

Sometimes it may not be feasible to change the project profile or the process model. In these cases, there should be some contingency (extra time/resources) decisions in the project plan to be better prepared for anticipated changes. For example:

- Contingency factors (the percentage of extra time/resources) should be decided for those tasks which are affected by the expected change patterns.
- Some resources should be reserved for possible re-work to cope with the anticipated change patterns.
- Some possible extra tasks should be perceived beforehand to deal with changes, for example re-negotiation with the customer about budget, functionality, and deliver date in case of later requirement change.

In the following, we propose a systematic method to conduct the evolution pattern analysis and to make corresponding decisions. Several evolution patterns can be expected for a new project. Each pattern could cause different change events, and each event has alternative reactions and impact on the schedule/quality/cost. To provide a control mechanism for analysis and decision making, we can create and use *evolution scenarios*. In the Riskit method [9], Kontio suggests to classify identified risks into elements (factors and events), and describes plausible reactions for each event, resulting in what he terms a risk scenario. However, his method does not utilize inter-project experience. In our case, an evolution scenario is a path from an evolution pattern to one of its events, and further to one possible reaction. The evolution patterns are anticipated from the baseline project. Thus, it is easier and more natural to break down a pattern into several events, and to provide alternative reactions to each event. All these are guided by the information in the evolution pattern which embodies inter-project experience. Another difference between our method and Riskit is that our proposed decisions are mainly made during the planning phase - before a project starts - while risk analysis in Riskit is mainly carried out during the project execution.

Using evolution scenarios, the main steps in decision making become:

1. For each evolution pattern, perceive the possible change events;
2. For each change event, perceive the alternative reactions.
3. Each path in the above decomposition graph is now an *evolution scenario*;
4. For each scenario, estimate the impact on schedule/quality/cost based on data in the baseline – the actual impact in the baseline project for similar scenario;
5. For each scenario, estimate its probability based on the frequency data in the baseline project;
6. If there are too many scenarios, select the ones with high probabilities and/or significant impact, and make decisions to change the project profile and/or the process model, or define appropriate contingencies to deal with the expected important changes.

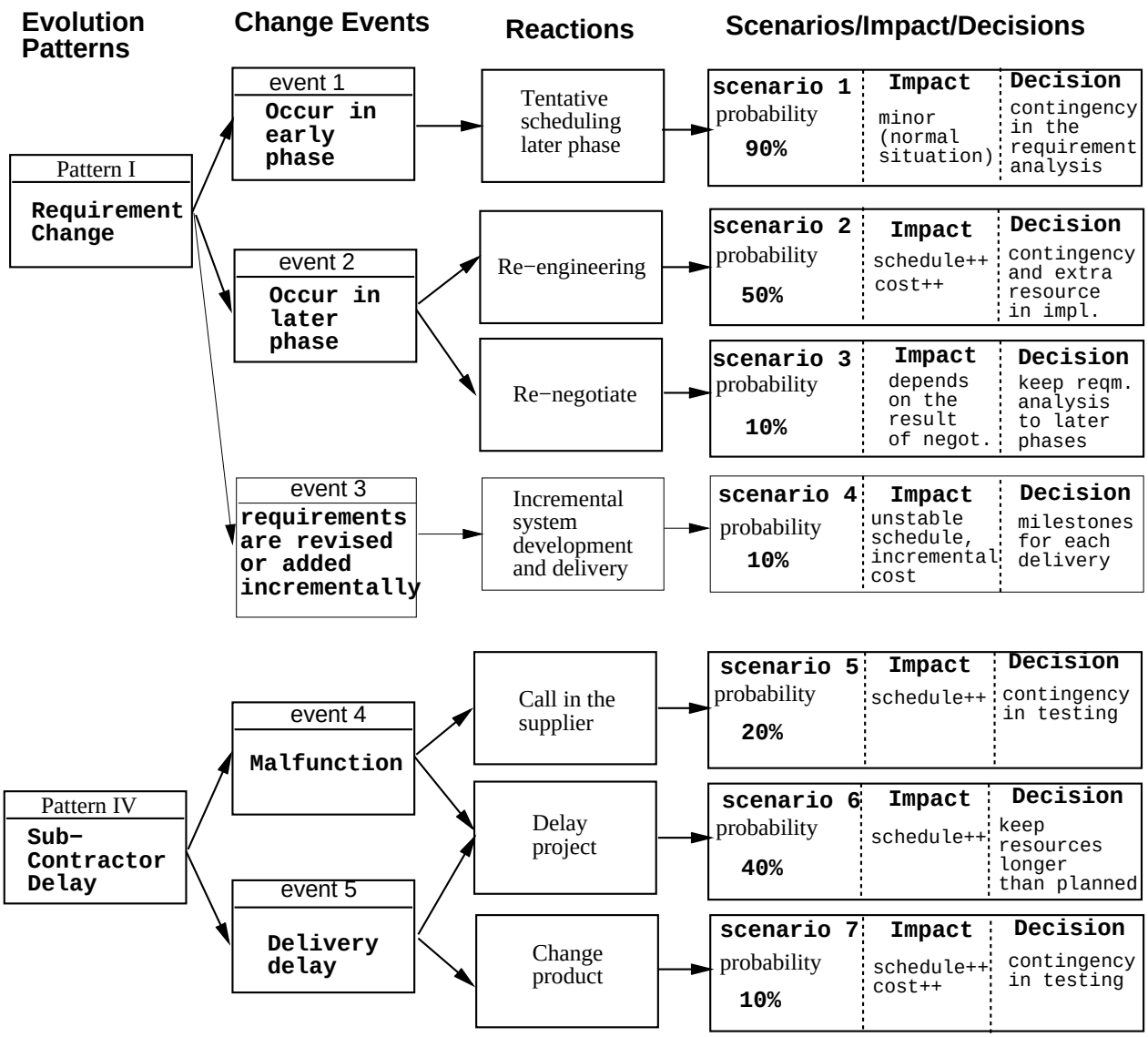

Figure 5: Evolution Scenarios and Decision Making.

Figure 5 shows examples of evolution scenarios and decision making for some typical evolution patterns.

4.4. *Measurements and Risk Analysis*

Systematic and valid process improvement can be achieved only by analyzing the process in a quantitative way, i.e. requiring measurements. For example the Goal-Question-Metric (GQM) method [2] that defines goals for software process improvement before embarking on software measurement tasks. Our method for predictive process evolution relies heavily on software metrics. Thus the project plan should include measurement of the product as well as of the process.

In our case, the process goals for measurements are:

- To monitor projects and their environments for factors that could fail, and to predict risks to schedule, budget, and product quality (risk analysis);
- To revise/refine the project plan to react to (anticipated) changes;
- To track change reactions for their cost and effectiveness;
- To gain experience for future projects; etc.

In the project plan, the following measurement mechanisms should be specified:

- The process/product attributes to measure during project execution;
- The data collection procedures, tools and timing;
- The mathematical/statistical/empirical models used in data analysis;
- The data analysis procedures and their results;
- How to use the results of data analysis, e.g plan revision, risk warning, future learning, and so on.

For a particular project, the project manager should focus on selected issues based on the expected evolution patterns. In the following, we give some examples.

1. Requirement Change – Evolution Pattern I

- *Data Collection and Analysis*:
  Count periodically the number of requirements (initially and number of added /deleted /modified requirements).
- *Risk Analysis*: If the requirements are still being modified, then the schedule is still at risk. If the total number of requirements is stabilizing, the risk is minimal.
- *Reactions*: If the risk to schedule is high, re-negotiate the budget or adopt the incremental system development and delivery approach.

2. Sub-Contractor Delay – Evolution Pattern IV

- *Data Collection and Analysis*:
  The estimated date D_1 of the availability of the external product P (based on its announced arrival time and the model for its acceptance testing) is periodically reported to management. In the project plan, the manager has the planned date D_2 for the availability of P.

- *Risk Analysis*: The risk to the project schedule and cost is proportional to $D_2 - D_1$.

- *Reactions*: If the risk to schedule is high, consider replacing the external product P.

3. Internal Schedule Adjust – Evolution Pattern VIII

- *Data Collection and Analysis*: the effort data (person-hours) is collected by weekly reports to management showing recorded hours for each task. In the project plan, the manager has effort estimates for each phase and the total effort. The actual and estimated effort is periodically compared.

- *Risk Analysis*: Extensive deviation would indicate a potential problem area and risk to the schedule.

- *Reactions*: classical project follow-up.

4. Review-Modify Cycle – Evolution Pattern IX

- *Data Collection and Analysis*: Measure module characteristics (size, complexity, fan-in/fan-out, etc.). Use some model of correlation between module characteristics and module risk level. Applying the model to each module, we can estimate its risk level.

- *Risk Analysis*: To identify "high risk" modules that exceed their estimated or recommended complexity and size.

- *Reactions*: "High risk" modules may need further investigation or even redesign.

5. Test-Modify Cycle – Evolution Pattern X

- *Data Collection and Analysis*: Estimate the total number of defects using industry guidelines, e.g. 7 defects per KLOC. Periodically count the defects found, and estimate the rate at which defects are found using some defect trending model. Then we can estimate how many defects that remain at any time of the project.

- *Risk Analysis*: If too many defects have not been found and the product is released shortly, then the reliability of the product may be unacceptably low and this is risky.

- *Reactions*: Prolong the period initially planned for the testing phase.

Based on different evolution patterns, the project plan should specify appropriate mechanisms for risk analysis and management (data collection, analysis method and models, risk analysis method and models, etc.) and pre-allocate necessary resources for reactions.

5. Preliminary Experiences

Parts of the (re)planning process in this paper has been prototyped, using the EPOS system [18]. Real projects were "simulated" in EPOS, with active process models. The empirical data were taken from a case study conducted with a Norwegian banking software house, called YYY [20]. In that study we classified five historical projects and collected actual change occurrences in these. Organization YYY is ISO-9001 certified since January 1995, and has therefore a documented quality system. The purpose of the case study was to collect actual evolution data based on parts of the framework described in this paper. The preliminary analysis shows that most changes originate from customer delays or requirement changes. Many such changes occur in the design and testing phase, and constitute a major impact in term of extra cost. However, the case study and the initial EPOS implementation covers only part of the work presented in this paper. E.g., the classification system was not fully utilized, and replanning was largely incomplete.

The experience database for YYY has now been re-implemented in ORACLE, and is being populated with data from 30 past projects and tried out on 5 ongoing projects [7]. The gained experiences is being fed back to the organization for making effective improvement decisions. YYY now uses a classification metrics called Project Implementation Profile (PIP) to specify the operating-profile [25], using 10 factors and with 10 values each.

The proposed classification and planning framework will be tried out on a larger scale in the Norwegian SPIQ project in 1997-2001, where SPIQ stands for "Software Process Improvement for better Quality". Most of the 12 participating SPIQ companies including YYY are interested in establishing a corporate Experience Base to improve project estimation and generally to achieve SPI. Five of them are now building up small, web-based experience bases to improve their estimation capabilities. Their initial goal is better cost/time estimation of current projects, and not to manage unanticipated process changes. However, such process changes represent the (un)expected variation in such estimates, and are thus valuable information. In the next round, more up-front planning for process changes can be employed. For instance, one of these companies use five major tollgates in their projects, where they assess progress and (re)plan further activities. This is consistent with the proposed planning support in this paper.

6. Conclusion

The central topic of the paper is the planning support for software process evolution. This work presents not only research results in software engineering alone, but also transfers methods of knowledge engineering to software engineering, applying the AI planning technique to software process modeling and software project management. The main contributions in this work are:

- We propose a new classification of software projects, and several special *overall-profiles* of projects (time-critical, safety-critical, and prototyping) are identified apart from the classical time/budget constrained projects. In addition, several factors to constitute *operative-profiles* are introduced.

- We identify and characterize ten software *process evolution patterns* and link them to different project profiles.

- Based on the evolution patterns, we discuss the *planning support* for process evolution. The following methods are new or significantly extend existing work:

  - Cost estimation for *process changes*;
  - Evolution pattern analysis, and decision making for local contingencies or for more global SPI.
  - A total process for initial planning and iterative replanning, documented by a coarse process model (as pseudo-code).

The presented framework is a bit preliminary and needs detailing towards individual companies and cases. However, its core features have been applied with preliminary, encouraging results.

1. José A. Ambros-Ingerson and Sam Steel. Integrating Planning, Execution and Monitoring. In *Proc. of AAAI'88*, pages 83–88, 1988.
2. Victor Basili, Gianluigi Caldiera, and Hans-Dieter Rombach. Goal Question Metric Paradigm. In *[17]*, 1994.
3. Victor R. Basili, G. Caldiera, Frank McGarry, R. Pajerski, G. Page, and S. Waligora. The Software Engineering Laboratory – an Operational Software Experience Factory. In *Proc. 14th Int'l Conference on Software Engineering, Melbourne, Australia*, pages 370–381, May 1992.
4. Victor R. Basili and Hans D. Rombach. Support for Comprehensive Reuse (on TAME project). *Software Engineering Journal (special issue on Software process and its support)*, 6(5):303–316, September 1991.
5. Barry W. Boehm. *Software engineering economics (on COCOMO model).* Prentice-Hall, 1981.
6. Morten Håker. Evaluering av Prosjektstyringsverktøy, December 1994. 79 p. (diploma thesis). EPOS TR 239.
7. Bent Ingebretsen. Erfaringsdatabase for firma X (in Norwegian), 2 March 1998. 206 p. + Confidential Appendix 14 p., EPOS TR 313 (diploma thesis).
8. Ivar Jacobson, Martin Griss, and Patrick Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success.* ACM Press / Addison Wesley Longman, New York / Reading, Massachusetts, 1997.
9. Jyrki Kontio. A Case Study in Applying a Systematic Method for COTS Selection. In H. Dieter Rombach, editor, *Proc. 18th Int'l Conf. on Software Engineering (ICSE'96), Berlin*, pages 201–209. ACM/IEEE-CS Press, N.Y., March 1996.
10. David Leare. *CBR – Experiences, Lessons and Future Directions.* AAAI/MIT Press, 1996.
11. Manny M. Lehman. Software Evolution. In *[17]*, pages 1202–1208, 1994.
12. B. Lientz, E. Swanson, and G. Tompkins. Characteristics of application software maintenance. *Communications of the ACM*, 21(6), June 1978.

13. Mikael Lindvall and Kristian Sandahl. Practical Implications of Traceability. Technical report, Linköping University, 1996. To appear in Software Practice and Experience, 18 p.
14. Chunnian Liu. An Expert System for Program and System Development. In *Proc. AVIGNON'91, Avignon, France, May 27–31, 1991, Volume 3*, pages 97–110, 1991.
15. Chunnian Liu and Reidar Conradi. Automatic Replanning of Task Networks for Process Model Evolution in EPOS. In *[27]*, pages 434–450, 1993.
16. R.J. Madachy. Knowledge-Based Risk Assessment and Cost Estimation. *Automated Software Engineering*, 2:219–230, 1995.
17. John J. Marciniak, editor. *Encyclopedia of Software Engineering.* John Wiley and Sons, 1994.
18. Minh Ngoc Nguyen. *Framework and Approach for Managing Software Process Evolution in EPOS.* PhD thesis, IDI, NTNU, 22 May 1997. IDI-rapport 97:7, NTNU PhD thesis 1997:73, SU-report 6/97.
19. Minh Ngoc Nguyen and Reidar Conradi. Classification of Meta-processes and Their Models. In *[23]*, 1994. 167–175.
20. Minh Ngoc Nguyen, Alf Inge Wang, and Reidar Conradi. Total Software Process Model Evolution in EPOS. In Richard N. Taylor and Alfonso Fuggetta (Eds.), editor, *Proc. 19th Int'l Conf. on Software Engineering (ICSE'97)*, pages 390–399. ACM/IEEE-CS Press, N.Y., May 1997.
21. Marc C. Paulk, Charles V. Weber, Bill Curtis, and Mary B. Chrissis. *The Capability Maturity Model for Software: Guidelines for Improving the Software Process.* SEI Series in Software Engineering. 640 p. Addison–Wesley, 1995.
22. Z. Pawlak. *Rough Sets – Theoretical Aspects of Reasoning about Data.* Kluwer Academic Publishers, Boston, London, Dordrecht, 1991.
23. Dewayne E. Perry, editor. *Proc. 3nd Int'l Conference on Software Process (ICSP'3), Washington, 187 p.* IEEE-CS Press, October 1994.
24. Felix Redmill. *Software Projects – Evolutionary vs. Big-Bang Delivery.* John Wiley, New York, 1997. 254 p., ISBN 0 471 93343 0.
25. R.L. Schultz and D.P. Slevin. Project Implementation Profile (PIP). *Project Management Journal*, September 1986. Distributed in Compendium in course 92520 Project Organization, 1997 NTNU; translated to Norwegian by Telenor-Novit.
26. Ian Sommerville. *Software Engineering, Fourth Edition.* Addison-Wesley, 649 p., 1992.
27. Ian Sommerville and Manfred Paul, editors. *Proc. 4th European Software Engineering Conference* (Garmisch-Partenkirchen, FRG), *Springer Verlag* LNCS 717, *516 p.*, September 1993.