DRAFT

full paper published in:

Foundations of Computing and Decision Sciences
Vol. 29, No. 1-2, 2004, pp. 115-131

proc. of 5th National Conference on Software Engineering
October 14-17, 2003 Szklarska Poręba, Poland

Paper published in the journal and presented at the conference

RISK IDENTIFICATION PATTERNS FOR SOFTWARE PROJECTS

Jakub MILER*, Janusz GÓRSKI*

Abstract. The paper presents a systematic approach to software risk identification based on risk patterns. The approach assumes explicit modelling of the relevant business process that provides for controlling the scope and ensures the completeness of the analyses. We demonstrate how this approach can be applied using RUP as the reference model of software processes. Then we report on the validation experiment during which we applied our approach to verify completeness of a well known checklist of software project schedule risks. The paper concludes on the results of this experiment and gives our plans for further research.

1. Introduction

The success of software development and/or acquisition projects is not guaranteed. The aim of any software project is to provide the problem stakeholders with a satisfactory solution within the schedule and budget limits. The risk of poor product quality and schedule or budget overruns is high which is confirmed by a number of cancelled, delayed or overpaid projects. Effective management of those risks is presently perceived as one of the most important areas of project

* Department of Software Engineering, Gdansk University of Technology, Narutowicza 11/12, 80-952 Gdańsk, Poland. E-mail: jakubm@eti.pg.gda.pl, jango@pg.gda.pl

management [7]. Current practice of risk management is mostly based on the intuition and personal experience of project managers. The ongoing research effort aims at providing effective support of risk management activities, and in particular at providing for reusing the risk-related knowledge and experience gathered in earlier projects.

In this paper we report on some results related to risk identification. Effective risk identification is the triggering condition of the other risk management activities, and therefore supporting the risk identification phase is especially worthwhile.

Our approach to risk identification, presented in the subsequent sections, is characterized by the following features:

- explicit modeling of *software process* to control the scope of risk identification,
- using *risk patterns* to identify potential risk factors.

First we describe our approach and demonstrate its applicability by some examples. Then we apply our approach in the experiment which objective is to validate a widely known risk identification checklist [5]. In particular we show how our approach helps in finding, in a very systematic way, a number of (apparently important) risks that are missing in [5], at the same time being able to "cover" most of the risks presented in [1].

The results of our experiment demonstrate that the proposed approach is an effective mechanism supporting risk identification at different abstraction levels of process structuring, ensuring that a complete set of relevant risks is being taken into account.

Current approaches to risk identification can be roughly divided into two groups:

- risk identification based on checklists (both questionnaires and risk lists),
- risk identification based on group effort (e.g. brainstorming).

Working with predefined checklists means easy yet tiresome answering numerous questions, not always relevant for a given level of detail. The advantage is that a checklist helps in controlling the scope and (if it is complete) prevents overlooking of some important risks. Group effort like brainstorming emphasizes high human involvement in risk identification without any rigorous scope control.

As for now, many different risk checklists were proposed using multiple taxonomies of risk areas. Some of the checklists are publicly available (e.g. SEI Taxonomy-Based Questionnaire [8], Capers Jones' 60 risk factors [1], Steve McConnell's Complete List of Schedule Risks [5]), while other remain the private property of their proprietary owners (e.g. development companies, consulting offices). Software Engineering Institute (SEI) [9] proposes the taxonomy of a software project together with detailed questionnaire (194 questions). However, the questionnaire is not accompanied neither with a sophisticated method of risk specification nor with a list of risks implied by the answers to the questions. Other researchers who propose their own risk lists, describe a risky situation either as a single statement in a natural language [5], or a structured set of such statements [1].

As far as group approach to risk identification is concerned, a particularly noteworthy contribution is by Jyrki Kontio. He studied the effectiveness of brainstorming in risk identification, and as a result, he provides the participants of a brainstorming session with structuring templates to better specify the discovered risks. Still, for the sake of simplicity of the method, he has used statements in a natural language to express the risks [4].

Our method can be used to help in both approaches to risk identification. In the checklist oriented approach it can be used to reengineer the existing checklists to provide for better scope control and completeness. In the group work approach it can be used to support brainstorming by providing the means to precisely describe risks at different levels of detail and to control the scope of the analysis.

The work presented in this paper is situated within a wider context of our research towards an integrated software environment to support risk management in software projects. We have already developed a pilot implementation of a software tool [6] and carried out a series of experiments to

validate both the support and the underlying theoretical concepts. The summary of our research is available on-line at [2].

2. Introduction

Our ignorance of risks does not imply that they are no longer threatening our undertakings. Early awareness of possible problems forms the basis of successful risk mitigation. Thus the risk identification is always the first phase of the risk management process. Once identified, the risk can be communicated within the project and then analyzed and coped with by undertaking appropriate actions.

The identified risks should be adequately documented. We distinguish two primary components of risk-related information:

- *Risk statement* – description of a particular event that, once materialized, adversely affects the project,
- *Risk context* – description of the position of the risk in terms of project tasks, personnel and products.

The accuracy of risk-related information facilitates risk communication and allows for precise analyses. In particular, the risk statement provides the actual description of the unwanted event or set of circumstances, while the risk context maps that event or circumstances to certain activities in the project work breakdown structure. This way a vague description of a general event can be complemented with the details of a project task resulting in a well-situated definition of a risk. However, the activities themselves cannot be considered as the only elements of a risk context. The context includes also the personnel responsible for the activities together with the source materials and output products. The context elements should be explicitly referred to in the risk statement to allow for focused and precise risk identification and communication.

In order to establish the context for possible risks in a software project we need a model of that project defining the activities, roles and artifacts of the development process as well as their mutual relationships.

Software development processes differ in their structure. Nevertheless, we can distinguish few general meta-model concepts that are valid for structuring any software development process (or, more generally, any business process):

- *Activity* – an action animated by a human(s) in a certain role(s) processing input artifacts into output ones,
- *Role* – a function, responsibility of a human animating the activity (an individual can act in many roles),
- *Artifact* – an item processed by activities (e.g. input materials, documents, tools and output products).

Talking about risk requires explicit division between beneficial and destructive activities, artifacts and roles, and more precisely their qualities. To achieve this, we extend the basic activity-artifact-role meta-model with qualifying entities:

- *Practice* – the practice followed in order to complete the activity,
- *Capability* – experience, skill, ability of a human in a given role,
- *Feature* – quality aspect of a given artifact.

We assume that activities, artifacts and roles can be decomposed recursively. This provides for specifying the software development process to the level of detail that corresponds to the current risk identification perspective.

The meta-model of a software development process is presented in Fig. 1. as a UML class diagram.

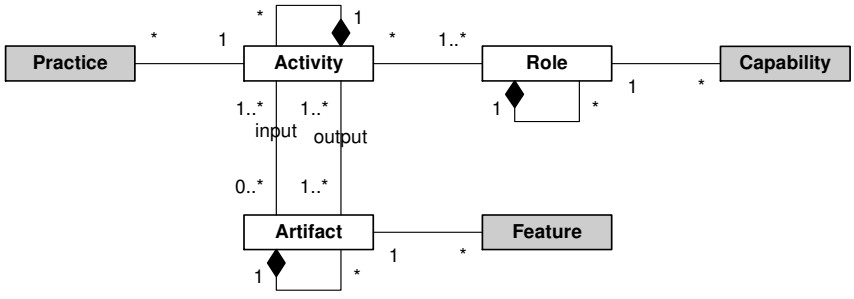

Figure 1. Meta-model for modeling the software project

We can use this meta-model to represent various process models. Once defined, such a model provides the context to describe potential risks. The occurrence of those risks can then be assessed by referring to the actual project (incarnation of the model). It should be noted that a process model is by definition a simplification of the real project and as such it allows only for as detailed specification of risks as the elements defined by the model. However, the process model can be improved to represent better the modeled reality.

In our research, we have selected the Rational Unified Process (RUP) [3] as a referential model of the software project. RUP can be easily represented in terms of the proposed meta-model. It defines directly the activities, artifacts and roles. The workflows, workflow details, artifact sets and role sets can be adapted as activities, artifacts and roles of higher (more general) levels of abstraction of the model. The practices, features and capabilities are extracted from the descriptions of RUP model elements, where they are given in a form suitable for direct importation. We have reengineered the RUP roles within the *Additional Role Set* to extend the *Stakeholder* role that we considered too general for risk context modeling.

At the most general level (level 0), that is not explicitly stated in RUP, we have the following:

Role:
- Personnel *(RUP Any Role)*

Activity:
- Software project

Artifact:
- Software product

At the framework level of customized RUP model (level 1) we have assumed RUP *role sets* as roles, *workflows* as activities, renamed and enhanced *artifact sets* as artifacts, and obtained the following:

Role (*RUP Role Sets*):
- Analyst
- Developer
- Tester
- Manager
- Support staff *(RUP Additional Role Set)*
- External stakeholder *(RUP Stakeholder Role)*

Activity (*RUP Disciplines-Workflows*):
- Business Modeling
- Requirements
- Analysis & Design
- Implementation

- Test
- Deployment
- Configuration & Change Management
- Project Management
- Environment

Artifact (*out of RUP*):
- Business Modeling Documentation
- Requirements Documentation
- System Analysis Documentation
- Design Documentation
- Implementation Documentation
- System
- Test Documentation
- Deployment Documentation
- Deployed Product
- Configuration Management Environment
- Configuration Management Documentation
- Plans
- Management Documentation
- Infrastructure
- Guidelines

At the intermediary level of customized RUP model (level 2) we have directly placed RUP *roles*, assumed *workflow details* as activities and described new artifacts. That way, at level 2 we obtained the following:

Role (*RUP Roles*):
- 8 Analyst roles
- 10 Developer roles
- 1 Tester role
- 7 Manager roles
- 5 Support staff roles *(reduced RUP Additional Role Set)*
    - Course Developer
    - Graphic Artist
    - System Administrator
    - Technical Writer
    - Tool Specialist
- 6 External stakeholder roles *(extended RUP Stakeholder Role)*
    - Customer
    - Client
    - User
    - Supplier
    - Consultant
    - Politics

Activity (*RUP Workflow Details*):
- 56 workflow details / avg. 6.2 per workflow

Artifact (*out of RUP*):
- products of workflow details / about 1-2 per workflow detail

At the most detailed level of customized RUP model (level 3) we have directly positioned RUP *activities* and *artifacts*, and defined the following:

Activity (*RUP Activities*):
- 150 activities / avg. 2.7 per workflow detail

Artifact (*RUP Artifacts*):
- 119 artifacts including:
  - 6 UML models
  - 6 combined artifacts

The adapted RUP model presented above can be extended further as needed by a particular software project. To add new model elements (e.g. activities) one has to set up their position in the hierarchy of that elements (the activities in the example) as well as to define the relationships with existing model elements of other types (in the example, with artifacts and roles). Additionally, expected properties of the new element must be defined (i.e. practices of activities, capabilities of roles and features of artifacts). It may be helpful here to refer to the properties of already existing model elements. Our experiences with the adaptation of the RUP model to our approach show that the further adaptation will not require significant effort. In particular, as the model becomes more detailed, the effort to introduce a new element should decrease because the changes tend to be more localized.

3. Risk identification patterns

Mapped to the model of a software project, a risk event can be represented as a violation of a (expected) relationship in the model e.g. *certain activity does not respect a recommended practice*. Upon examination of all entities and relationships in the meta-model presented in Fig. 1. we can define a complete set of classes of risk events. They are listed below. We follow the convention that an object name is given in bold and followed by its meta-class name (in angle brackets and italic).

Artifact classes of events:
Ar*<artifact>* is not developed
Ar*<artifact>* loses **F***<feature>*

Activity classes of events:
A*<activity>* is not performed
A*<activity>* takes more time than expected
A*<activity>* costs more than expected
A*<activity>* loses **R***<role>*
A*<activity>* loses **Ar***<artifact>*
A*<activity>* loses **P***<practice>*

Role classes of events:
R*<role>* is not assigned
R*<role>* loses **C***<capability>*

The events denoting lack of materialization of a certain model element (artifact, activity, role) were added to take into account possible violations in model implementation. Similarly, the events referring to time and cost are intended to cover the schedule and budget impact.

The meaning of loss is defined as follows:

A*<activity>* loses **P***<practice>* means that **P** is not performed in **A**,

A*<activity>* loses **R***<role>* means that **R** is not played (even if it is assigned to an individual) within the context of **A**,

A*<activity>* loses **Ar***<artifact>* means that **Ar** is not used (even if it has been produced) within the context of **A**,

Ar*<artifact>* loses **F***<feature>* means that **Ar***<artifact>* exhibits **F** less than expected,

R<*role*> loses **C**<*capability*> means that **R**<*role*> exhibits **C** less than expected.

Analyzing the causal relationships between the events of the above classes we can easily identify possible *risk identification patterns*. Such a pattern is shown in Fig. 2. It reads as follows: *if* Event 1 *occurs in the present state then* Event 2 *is likely to occur in the future state.* In other words, the materialization of a cause does not automatically imply the occurrence of the consequence. It rather increases the likelihood of the consequence to materialize itself.

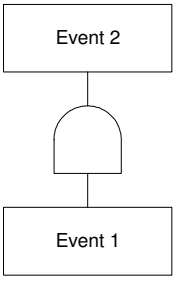

Figure 2. Event tree for cause-consequence risk scenario

To generate all possible risk patterns we have combined all risk event classes with each other within a cause-consequence scheme shown in Fig. 2. Then we have applied a syntactical filter to remove those combinations that did not make any sense regardless of actual elements of a software project model. This way, we have reduced the number of patterns from 100 to 82.

The patterns were also analyzed for meaningfulness of different combinations of actual model elements. Then the restrictions were defined for some of the patterns stating that, for example, activities placed in "if" and "then" part must be different, otherwise the resulting phrase did not make sense. Obviously, that high number of patterns does not mean that the number of risk scenarios identified in a given process model will be huge. When we substitute the actual activities, roles and artifacts to the patterns, only few of them will represent the actual risk scenarios. Moreover, some can even signify positive events in the process. By substituting the real process elements to the patterns we can limit the list only to the meaningful risks in the context of given process model.

Examples of risk identification patterns are given below. In the first pattern, **A1** and **A2** denote two different activities, as with the same activity that particular pattern would be meaningless.

If **A1**<*activity*> is not performed then **A2**<*activity*> takes more time than expected.

If **A**<*activity*> takes more time than expected then **R**<*role*> loses **C**<*capability*>.

If **A**<*activity*> costs more than expected then **A**<*activity*> loses **P**<*practice*>.

If **A**<*activity*> loses **R**<*role*> then **A**<*activity*> takes more time than expected.

If **A**<*activity*> loses **Ar1**<*artifact*> then **Ar2**<*artifact*> loses **F**<*feature*>.

If **A**<*activity*> loses **P**<*practice*> then **A**<*activity*> takes more time than expected.

If **A**<*activity*> loses **P**<*practice*> then **Ar**<*artifact*> loses **F**<*feature*>.

If **Ar**<*artifact*> is not developed then **A**<*activity*> loses **Ar**<*artifact*>.

If **Ar**<*artifact*> loses **F**<*feature*> then **A**<*activity*> loses **P**<*practice*>.

If **R**<*role*> is not assigned then **A**<*activity*> is not performed.

If **R**<*role*> loses **C**<*capability*> then **A**<*activity*> loses **P**<*practice*>.

If **R**<*role*> loses **C**<*capability*> then **R**<*role*> loses **C**<*capability*>.

It cannot be assessed if those patterns represent the meaningful risk scenarios before assigning the actual activities, roles, artifacts, practices, capabilities and features to the variables occurring in the patterns. Some candidate patterns, however, could be rejected based on purely syntactic criteria, an example is given below:

If **Ar**<*artifact*> loses **F**<*feature*> then **A**<*activity*> loses **R**<*role*>.

Despite the concrete objects substituted by **Ar**, **F**, **A** and **R** this pattern combines two highly independent events that are unlikely to be a risk scenario.

The risk identification patterns can be systematically applied to a given process model to identify possible risk scenarios. Examples of risk events and scenarios generated by applying selected patterns to the RUP referential model presented in the previous section are given below. The examples are divided according to the level of detail of the model. In curly brackets we give a natural language description of more complex scenarios.

Level 0:

Software Project<*activity*> costs more than expected

Product<*artifact*> loses **Quality**<*feature*>

Personnel<*role*> loses **Motivation**<*capability*>

Level 1:

If **Business Modeling**<*activity*> loses **Capture a common understanding of current business domain and model target business**<*practice*> then **Business Modeling Documentation**<*artifact*> loses **Conformity to business**<*feature*>

{Business modeling documentation does not represent well the business due to poor understanding of business domain and lack of modeling}

If **Business Modeling Documentation**<*artifact*> loses **Conformity to business**<*feature*> then **Product**<*artifact*> loses **Quality**<*feature*>

{Wrong understanding of business domain results in low quality product}

Level 2:

If **Identify Business Processes**<*activity*> loses **Conduct a workshop, where the goal is to decide on terminology and outline what the business use cases and business actors will be**<*practice*> then **Business Process List**<*artifact*> loses **Completeness**<*feature*>

{Lack of business definition workshop results in incomplete identification of business processes}

Project Manager<*role*> is not assigned

{Project manager quits the project}

Level 3:

Monitor Project Status<*activity*> loses **Employ adequate measurement**<*practice*>

{Inadequate measurement}

It may be noticed that this risk event is also recalled by Capers Jones [1] as risk no. 28.

If **Business Glossary**<*artifact*> loses **Clarity**<*feature*> then **Business Use-Case Model**<*artifact*> loses **Consistency of terms**<*feature*>

{Ambiguous glossary of business terms results in inconsistent business model}

If **Test Ideas List**<*artifact*> loses **Representation of all quality risks**<*feature*> then **Test Case**<*artifact*> is not developed.

{Wrong scope of testing in terms of target quality restricts some beneficial tests}

Given the set of potential risk scenarios we can find out which of them actually take place in the project. This can be generally achieved in two ways: questionnaires and brainstorming. Both techniques can benefit from using our method of risk identification patterns. The questionnaires can be simplified and standardized and the form of questions to be asked would be, for example:

Does **A***<activity>* involve **P***<practice>*?

Does **R***<role>* have **C***<capability>*?

Does **Ar***<artifact>* possess **F***<feature>*?

As for the brainstorming sessions, the participants can start with a very general definition of a software project (e.g. level 1 of the RUP model) and then explore the areas where the most risks are identified. Throughout the session, the risks are documented by applying risk patterns to the specific elements of the process model.

Later on, the risk identified with both techniques must be evaluated to select that intended for mitigation (e.g. threatening the project the most, the cheapest to mitigate). The subject of probability and severity evaluation of a pattern-based scenario is beyond the scope of this paper resting however within the area of our further research.

4. The experiment

4.1. Experiment design

To validate our approach empirically, we have carried out an experiment. The experiment involved one of the risk identification checklists available publicly, namely Steve McConnell's 'Complete List of Schedule Risks' [5]. This checklist catalogs 109 risk factors in 12 areas of a software project such as: schedule, product, personnel or customer. The factors are expressed in the form of phrases in natural language.

We have defined two distinct goals of the experiment.

- The first focused on validation of the completeness of McConnell's checklist using our method of risk patterns.
- The second focused on systematic application of our method to the RUP-motivated extension of the process model underpinning the McConnell's checklist.

The experiment plan comprised 5 steps:

1. Extraction of the process model (i.e. activities, artifacts, roles, practices, features and capabilities) that underpins the McConnell's checklist,
2. Expressing all the risks in the checklist in terms of that process model using risk patterns,
3. Systematic application of the risk identification patterns to the revealed process model in order to check if we can find important risks that are not present in the McConnell's checklist,
4. Extending McConnell's process model with selected representative elements of the referential RUP model,
5. Further application of the risk identification patterns to the extended process model and identification of the missing schedule risk factors in new areas.

As there is no explicit process model underlying the Steve McConnell's checklist (at least it is not known to the authors of the paper), such a model (implied by the checklist) was built in Step 1. Having extracted the process model, we applied our risk identification patterns, first to express the risks already defined by McConnell (Step 2) and then to determine new risks from the model (Step 3). These two phases fulfilled our first objective of the experiment – we could check if risk identification patterns can add meaningful risks to the checklist. For the second objective, we have

extended the original process model (Step 4) by the elements suggested by RUP (and not present in the model) and once again applied the risk identification patterns to the resulting process model (Step 5).

4.2. Detailed experiment results

The main results of the experiment were obtained from steps 3 and 5, however each step of the experiment brought some interesting results that can be reported separately. The following sections detail the results of each step.

4.2.1. Step 1

In this step we explicitly defined a somewhat hidden model of the development process referred to by McConnell's risk factors. We have described the model in terms of our meta-model (Fig. 1). Due to space limitations we cannot present this model here. Instead, in Table 1 we give the summary of its complexity. Particular items of the model will be used later on in exemplary risk statements in the paper.

Table 1. Statistics of the process model extracted from the McConnell's checklist

Model element	Count	Remarks
Role	11	2 levels of detail
Activity	26	4 levels of detail
Artifact	20	2 levels of detail
Capability	37	for 9 roles
Practice	31	for 11 activities
Feature	53	for 20 artifacts

4.2.2. Step 2

In step 2, applying our risk patterns, we redefined the risks already present in the checklist. This step may seem futile as it uses the information extracted from the very checklist to reconstruct it back. Nevertheless, this way we could verify that our risk pattern language is expressive enough to rephrase the McConnell's risks.

Finally, all 109 risks were translated according to the risk patterns, but due to obvious limitations of this paper we can provide only some examples. The following paragraphs list the examples of McConnell's risk factors expressed with our risk identification patterns (first we recall the original definition giving its position in the McConnell's list in square brackets and then we rephrase it in our risk pattern language):

[9] Excessive schedule pressure reduces productivity

If **Project**<*activity*> loses **Avoid excessive schedule pressure**<*practice*> then **Personnel**<*role*> loses **Productivity**<*capability*>.

[24] Management places more emphasis on heroics than accurate status reporting, which undercuts its ability to detect and correct problems

If **Management**<*role*> loses **Capability to promote accurate status reporting**<*capability*> then **Management**<*role*> loses **Ability to detect and correct problems**<*capability*>.

[29] Development tools do not work as expected; developers need time to create workarounds or to switch to new tools

If **Development Tools**<*artifact*> loses **Reliability**<*feature*> then **Development**<*activity*> takes more time than expected.

[33] End-user does not buy into the project and consequently does not provide needed support

If **End User**<*role*> loses **Commitment**<*capability*> then **Project**<*activity*> loses **Acquire support from end users**<*practice*>.

[48] Contractor does not buy into the project and consequently does not provide the level of performance needed

If **Contractor**<*role*> loses **Commitment**<*capability*> then **Contractor**<*role*> loses **Productivity**<*capability*>.

[52] Vaguely specified areas of the product are more time-consuming than expected

If **Requirements**<*artifact*> loses **Clarity**<*feature*> then **Development**<*activity*> takes more time than expected.

[53] Error-prone modules require more testing, design, and implementation work than expected

If **Module**<*artifact*> loses **Reliability**<*feature*> then **Testing**<*activity*> takes more time than expected and **Design**<*activity*> takes more time than expected and **Implementation**<*activity*> takes more time than expected.

[56] Development of the wrong user interface results in redesign and implementation

If **Product**<*artifact*> loses **Adequate user interface**<*feature*> then **Design**<*activity*> takes more time than expected and **Implementation**<*activity*> takes more time than expected.

[82] Conflicts between team members result in poor communication, poor designs, interface errors, and extra rework

If **Project**<*activity*> loses **Resolve conflicts between team members**<*practice*> then **Communication**<*activity*> takes more time than expected and **Design**<*artifact*> loses **Quality**<*feature*> and **Module**<*artifact*> loses **Quality of interface**<*feature*>.

[91] Sabotage by project management results in inefficient scheduling and ineffective planning

If **Management**<*role*> loses **Benevolence**<*capability*> then **Schedule**<*artifact*> loses **Efficiency**<*feature*> and **Development Plan**<*artifact*> loses **Effectiveness**<*feature*>.

[97] Necessary functionality cannot be implemented using the selected code or class libraries; developers must switch to new libraries or custom-build the necessary functionality

If **Reused Code**<*artifact*> loses **Adequate functionality**<*feature*> then **Implementation**<*activity*> takes more time than expected.

[108] Half-hearted risk management fails to detect major project risks

If **Risk Management**<*activity*> loses **Obtain personnel commitment for risk management**<*practice*> then **Risk List**<*artifact*> loses **Coverage of major risks**<*feature*>.

4.2.3. Step 3

In this step we identified some additional schedule risks not included in McConnell's checklist. In this step we focused just on the process model identified in Step 1 without extending it by any new areas that would imply new classes of risks. All the risks identified in this step were already present in the process model used inexplicitly by McConnell but were omitted in his checklist.

Due to the resource limitations we were not able to apply all the patterns to the entire model. Such a complete analysis would be best performed with the help of a tool. Nevertheless, within 2

hours of analyses we were able to find 14 missing scenarios that express significant schedule risk factors in a software project. Some of them are listed below.

Project<*activity*> loses **Government Regulations**<*artifact*>.

{Project neglects legal issues of government regulations}

If **Communication**<*activity*> takes more time than expected then **Project**<*activity*> loses **Resolve conflicts between team members**<*practice*>.

{Ineffective communication channel results in poor conflict detection and resolution}

If **Project**<*activity*> loses **Acquire support from end users**<*practice*> then **Product**<*artifact*> loses **Adequate user interface**<*feature*>.

{Wrong product user interface is developed without the support from end users}

Project<*activity*> loses **Contractor**<*role*>.

{Project does not outsource work where it could and should}

If **Requirements**<*artifact*> loses **Stability**<*feature*> then **Contractor**<*role*> loses **Commitment**<*capability*>.

{Contractor withdraws from the project because of very unstable requirements changing the contract}

4.2.4. Step 4

In this step, we added some example new elements to McConnell's process model. Those extensions were inspired by RUP. We added new practices, features and capabilities to already defined activities, artifacts and roles as well as to newly introduced model elements.

Table 2 gives the summary of the additions.

Table 2. Statistics of the extensions to the original process model

Model element	Count	Remarks
Role	2	
Activity	2	
Artifact	3	
Capability	4	for 3 old, 1 new role
Practice	4	for 2 old activities
Feature	5	for 3 old, 1 new artifact

4.2.5. Step 5

In this step we identified some schedule risk factors concerning new areas of the software project. They were obviously not present in McConnell's checklist as they emerged from new model elements "borrowed" from RUP. It is worth mentioning that the new risk factors did not differ in detail from the ones already reported by Steve McConnell. In other words, the identifying capability of our approach resulted from broadening the scope of the model rather than going down into the very details.

As in Step 3, the limited resources forced us to search only for some meaningful examples. Thus, after additional 2 hours of analyzing we identified additional 14 risk scenarios. As in Step 3, we present below only few examples of discovered schedule risk factors.

If **Training Materials**<*artifact*> is not developed then **End User**<*role*> loses **Capability to accept product**<*capability*>.

{End users do not accept the product because of lack of training materials}

If **Configuration & Change Management**<*activity*> is not performed then **Requirements**<*artifact*> loses **Completeness**<*feature*> and then **Product**<*artifact*> loses **Conformity to user expectations**<*feature*>.

{Without change management process new requirements cannot be efficiently incorporated resulting in unsatisfactory product}

If **Programming Guidelines**<*artifact*> is not developed then **Module**<*artifact*> loses **Readability**<*feature*> and then **Testing**<*activity*> takes more time than expected.

{Lack of programming guidelines results in unreadable code that requires extra testing effort}

If **Project**<*activity*> loses **Model visually (UML)**<*practice*> then **Requirements**<*artifact*> loses **Clarity**<*feature*>.

{Requirements not specified visually are less clear to the development team}

If **Management**<*role*> loses **Presentation, communication, and negotiation skills**<*capability*> then **Customer**<*role*> loses **Commitment**<*capability*>.

{Poor interpersonal skills of management discourage the customer}

4.3. Experiment summary

We have gathered some metrics in the particular phases of the experiment to achieve better awareness about the effort of the analytical work. Table 3 lists measured durations of experiment phases.

Table 3. Duration of experiment phases

Phase	Duration [hours]
Steps 1 and 2	10
Step 3	2
Step 4	1
Step 5	2
Entire experiment	15

In the experiment all of the work was performed without any tool support (dedicated for the method). Complete analysis of McConnell's process model as done in step 3 of the experiment will certainly require such support. A suitable supporting tool will be developed within our further research.

Finally, we can mention a list of 17 recommended practices for the "Project" activity extracted from Steve McConnell's checklist as an extra result of the experiment. The practices include, for example:

- Avoid excessive schedule pressure,
- Acquire support from end users,
- Maintain good relationships between developers and management,
- Resolve conflicts between team members,
- Adhere to software policies and standards,
- Avoid bureaucratic adherence to software policies and standards.

5. Conclusions

In the paper we presented a new method of systematic risk identification based on risk identification patterns. The basic idea of this approach is that we explicitly model the context within which we seek for risks and this way we can control the scope and provide for the completeness of the analyses. As the risk context model we take the model of a business process. Our meta-model is general enough to cover virtually any business process model although presently our primary interest is in software development processes. By referring to RUP we ensure that we do not overlook any important areas that are relevant in software development.

We then defined a complete set of risk identification patterns that reflect the departures from the recommendations included in the description of RUP. Those risk patterns can then be applied to a concrete software development model that is expressed in the RUP terms to identify potential risks that are inherent in this model. Depending on the size of the model, the number of resulting risks can be quite large, but the pattern application effort is spent only once for the defined model. The actual effort to identify the risk scenarios by applying patterns to a model of a real software process is difficult to assess at the current stage of our research. It will require further experiments to be performed as well as the extensive use of supporting tools.

To validate our approach we applied it to the well known checklist published in [5] and the experiment provided the following results:

1. We were able to identify a number of missing risks that were not included in the checklist although they do not seem to be less important than those already present in the checklist,
2. We were able to complement the process model implied by the checklist with some important elements (that were not referred to by the original checklist) and then to identify even more risks missing from the checklist.

The distinguishing feature of our approach is that it creates a framework for systematic risk identification and it can be used on different process abstraction levels. Differently from the questionnaires that often contain mixed questions of diverse level of detail, the use of risk patterns is driven by the well-defined levels of process model. This gives the risk analyst a possibility to adjust it to the required level of detail to control the resources spent on risk identification while still controlling the scope and completeness of the analyses. Moreover, the scenarios identified through risk patterns recall more precisely the details of the analyzed project than the universal questions of the questionnaires thanks to their strict reference to the process model.

We believe that our approach can be helpful in both methods of risk identification: using the predefined risk checklists or brainstorming on risks during the controlled risk identification sessions.

In our further research plans we include:

- risk identification experiments carried out together with our industrial partners,
- studying the construction of more complex risk identification patterns including event and temporal logic
- investigating the ways our risk pattern based approach can support the risk analysis phase,
- integration of this approach into our risk management support tool [2].

We also plan to apply this approach to other business processes, e.g. hospital management or e-business processes.

References

[1] Capers Jones, *Assessment and Control of Software Risks*, Yourdon Press, 1994.
[2] RiskGuide Project site, http://mkzlway.eti.pg.gda.pl/RiskGuide.
[3] Rational Unified Process, http://www.rational.com/rup.

[4] Kontio J., *Software Engineering Risk Management: A Method, Improvement Framework, and Empirical Evaluation*, Doctoral Dissertation, Helsinki University of Technology, Finland, 2001.
[5] McConnell S., *Rapid Development*, Microsoft Press, 1996.
[6] Miler J., Górski J., *Supporting team risk management in software procurement and development projects*, proc. 4th National Conference on Software Engineering, Poznań - Tarnowo Podgórne, October 2002, NAKOM, Poznań, 2002, 17-32.
[7] PMBOK Guide, 2000 Edition, Project Management Institute, 2000.
[8] Sisti F. J., Joseph S., *Software Risk Evaluation Method*, Tech. Rep. CMU/SEI-94-TR-19, Carnegie Mellon University, Pittsburgh PA, December 1994.
[9] Software Engineering Institute, http://www.sei.cmu.edu.