MODELING STRATEGIC ACTOR RELATIONSHIPS TO SUPPORT RISK ANALYSIS AND CONTROL IN SOFTWARE PROJECTS

Subhas C. Misra, Vinod Kumar, Uma Kumar
Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, Canada

Keywords: Conceptual Modeling, Risk Analysis, Software Project Management.

Abstract: In this paper, we present an approach that project managers could use to model and control risks in software projects. There are no similar approaches on modeling software project risks in the existing pieces of literature. The approach is, thus, novel to the area of software risk management. The approach is helpful to project managers for performing means-end analysis, thereby uncovering the structural origin of risks in a project, and how the root-causes of such risks can be controlled from the early stages of the projects. We have illustrated this approach with a simple example typical of software development projects. Though some attempt has been made to model risk management in enterprise information systems using conventional modeling techniques, like data flow diagrams, and UML, the previous works have analyzed and modeled the same just by addressing "what" a process is like, however, they don't address "why" the process is the way it is. The approach addresses this limitation of the existing software project risk management models by exploring the strategic dependencies between the actors of a project, and analyzing the motivations, intents, and rationales behind the different entities and activities in a project.

1 INTRODUCTION

Analyzing risks from the early stages of software projects is essential for ensuring their success. Early project risk assessment helps managers to make speculative decisions, predict the causative agents of project failure, and thereby undertake remedial actions to control different project parameters (viz., resources, and external interfaces) from the early phases of the project. The technique described in this paper can be used by project managers to investigate the structural origin, and the root causes of the risks of a project, and thereby control risks from the early stages of the projects.

There have been a few remarkable studies on model-based software risk management. Two important ones are the *Riskit* Method (Freimut *et al*., 2001, Kontio, 1997), and Boehm's *Win-Win* approach (Boehm and Bose, 1994). In addition to the above, there are other pieces of literature available on the topic (e.g., Charette, 1989, Fairley, 1994, Gemmer and Koch, 1994), but we have listed some of those important ones that set our problem in the proper context.

In contrast to the previously proposed modeling techniques, in this paper, we show how we can use the concepts of modeling intentional dependencies between actors to explore the structural origins of risks in a software project. Although the concept of actor-dependency is not new, the way we use these concepts to analyze software project risks is *novel* to our work, and requires ingenuity of the modelers.

Since the problem that we embark on in this study has broad scope, before we proceed to Section 2, let us acknowledge some of the "in-scope" and "out-of-scope" items of this paper. In this paper, our goal is not to propose a fully functional risk management framework as that of Riskit (Kontio, 1997). Rather, we propose a novel approach that can be used in software project risk management by leveraging the modeling of dependencies between strategic actors, and thereby trying to explore the structural origins of project risks. In this paper, we focus on only the two *important* phases (*risk analysis* and *risk control*) of typical risk management lifecycle frameworks (e.g. Riskit, or Win-Win). But, we believe, our approach can be extended for exploring similar "means-end" relationships in other risk management phases. In

McLeod D. (2005).
SEMANTICS-BASED SIMILARITY DECISIONS FOR ONTOLOGIES.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 288-293
DOI: 10.5220/0002554402880293

this paper, we restrict ourselves to conceptual modeling only, by proposing the model, and illustrating the approach with a small "toy example", by keeping with the trend followed by most of the papers published on conceptual modeling in other areas (see the references to other conceptual modeling papers mentioned in Section 1). Finally, the approach discussed in the paper has been designed keeping software project environments in mind. However, we believe that the approach can be used, with minimal or no changes, for studying risk management in other application domains.

2 MODELING DEPENDENCIES BETWEEN STRATEGIC ACTORS

In this paper we discuss the actor dependency concept using i* (Yu, 1997). i* explores "why" processes are performed in the existing way. Moreover, it is much easier to obtain real and understandable requirements using i* modeling. Expected behavior of the software and its rationale could also be modeled using i*. Furthermore, i* does not take directly into account precision, completeness, and consistency as UML does. In contrast, i* principally takes into account actors' interests, goals, rationale, tasks and concerns.

In this work, we have used i* to model both requirements, and risk management elements which help managers to identify, monitor, analyze and control risks, all from the point of view of project goals. i* provides a qualitative analysis of project viability under several scenarios. In our context, this analysis will allow for verifying if all required actions to control risks have been taken into account (i.e., if project goals can be satisfied in all the studied scenarios). The strongest relation between requirements and risks are project goals. For any project, requirements can be modeled as goals and softgoals to be reached during project development. On the other hand, risks can be conceived as a set of "risky goals" and "risky tasks" performed by some actors in a particular role. Goals and tasks undertaken in a project often have some degrees of risks associated with them. The risks associated with these goals can be of varying degrees. The terminologies "risky goals" and "risky tasks" are used to signify those goals and risks that have associated *high risk factors*. The risky tasks may be perceived as undesirable tasks that may lead to a risky output (a "sub-optimal" goal). The reader should note that it is very common in software projects to knowingly undertake *high-risk goals and tasks*. What is important for a project manager is to analyze the structural origin of the risks, and vulnerabilities associated with those goals, and tasks. This is what we advocate in this paper. For example, a development team with poor java knowledge will have "risky goals" like developing the product without considering the quality, and "risky tasks" like the introduction of bugs due to poor development skills of the team. Because of other factors (such as, limited project budget, and lack of human resources), it might still be required for a project manager to proceed with a project, even after knowing the risks associated with having an inexperienced project team. What is rather important for the project manager is to model and analyze the root causes of the risks, their structural origin, and how the known risks can be controlled from the early stages of the project.

Those "risky goals", and "risky tasks" are actually risks that have to be reduced by other tasks performed by actors like a quality manager. "Risky goals", and "risky tasks" might pose substantial risks for achieving the project goals. In consequence, new tasks have to be defined to reduce those risks. Once the model is concluded, a simple algorithm is performed to mark project goals as either *satisfied* or *denied*, taking into account effects of tasks mentioned above. Finally, denied high-level goals indicate affected requirements. This is considered as risky, which means there are some missed *defensive* tasks in order to guarantee project success.

In order to model, and solve this problem, two actor-dependency diagrams are used: the Strategic Dependency Model (SD), and the Strategic Rationale Model (SR). In the interest of brevity, only a brief introduction of SD and SR is provided. Interested readers are referred to the literatures mentioned in Section 1 for learning further details. SD diagrams are used to model dependencies between actors, while SR diagrams are used to model internally why each actor has those dependencies. In other words, SD describes dependencies at a higher level of abstraction than SR, since SR shows an internal description of an actor and supports those dependencies.

All dependencies comprise of a "depender", a "dependee", and a "dependum". "Depender" depends on a "dependee" to get "dependum". The most important elements in SD diagrams are:

Goal dependency: It is used to model when one actor depends on another to make a tangible condition come true. Dependee has freedom to choose how to achieve this goal.

Task dependency: It is used to model when one actor depends on another to perform an activity. In this case, there is an implicit (usually not shown)

depender's goal, which explains why this task must be performed.

Resource dependency: It is used to model when one actor depends on another for the availability of an entity. Depender assumes that obtaining this resource will be straightforward.

Softgoal dependency: It is used to model when one actor depends on another to realize a fuzzy condition. In this case, *fuzzy* means there is no clear criteria for such a condition to be true. In this case dependee collaborates, but depender will decide how to achieve the softgoal.

Actors can be modeled as a generalized relationship among agents, position and role (Dubois *et al*., 1998). In general, agents represent physical manifestation of actors. Agents occupy a position in SD diagrams. In fact, a position is a generalization of an agent. Furthermore, positioned agents can have or cover several roles. For example, IT department is an agent, which occupies development team position. Also, development team can cover the role of development team with poor java knowledge.

On the other hand, SR diagrams focus inside actors. In fact, SR diagrams show both external and internal information. External information is modeled using the same elements of SD diagrams (e.g., goals, softgoals, resources and tasks). Internal information is represented basically using the same elements but arranged hierarchically in either a *means-end* or a *task-decomposition* relationship.

Internal elements of SR respond to external dependency relationships among actors. In general, external goals, tasks, softgoals, and resources are attached to internal tasks. Internal tasks might be decomposed into subtasks, subgoals, and subsoftgoals (task-decomposition relationships). Moreover, internal goals might depend on other subtasks (means-end relationships). Finally, internal softgoals might obtain either negative or positive contribution from tasks, and other subsoftgoals.

In the context of performing risk analysis, risky tasks performed by a particular role are linked to external goal and softgoals. This link will have a negative contribution to those external goals, and softgoals, which can be qualified as "some-", "hurt", "break", and so on, depending on the magnitude of contribution. On the other hand, tasks intended to help minimize the risk in identified risky tasks will have a negative contribution over links mentioned above. Again this contribution will be qualified as "some-", "hurt", "break", and so on. The stronger is the negative impact to the contribution to those links, the weaker is the effect of risky tasks on project goals. Finally, goals viability can be estimated by executing an algorithm to propagate contributions of tasks to project goals. Goals will be qualified as *satisfied*, *weakly satisfied*, *weakly denied*, and *denied*, depending on total effects on such a goal from the above mentioned tasks.

3 MODELING RISK ANALYSIS AND CONTROL IN SOFTWARE PROJECTS: THE PROPOSED APPROACH

In this section, we provide a very simple example to demonstrate our approach of using actor-dependencies to analyze and control risks.
Suppose a development team is composed of several members. John Doe is a member of the development team, and his *position* is a documenter. Also, he could *cover* several roles such as users manual documenter, and requirements documenter.

If it is needed to perform risk management focusing on documenters, new roles can be added which will represent documenter role introducing such risks. For example, a documenter could be one with poor writing skills. Suppose there is a client occupying the position of user, and covering the role of users manual reader. This user definitely depends on documenter to achieve goals like having a good users manual. A model describing the above concepts is shown in Figure 1.

Now, risks of this simple example can be analyzed using i*. When the documenter acts as one with poor writing skills, he will have some goals that can turn out to be "risky goals" – such goals are included into the SR diagram. Moreover, those goals might be the result of some "risky tasks". The risk of having a documenter with poor writing skills must be managed by another development team member, who will act as a quality assurance engineer with the role of *guaranteeing the user manual quality*. In this role, this member will have some goals and perform some tasks to minimize the risk effect. Figure 2 shows SR diagram, which analyzes the risks mentioned above.

Documenter covers the role of DOCUMENTER WITH POOR WRITING SKILLS, in order to model the risk associated with this situation. The effect of this risk is modeled as the risky goal USER MANUAL BY A POOR DOCUMENTER. This impact could be achieved by two risky tasks: ALLOW GRAMMAR ERRORS IN THE MANUAL, and ALLOW MANUAL DIFFICULT TO READ. Those bad tasks definitely will have impact on the goal HAVE GOOD USER MANUAL. The impact is modeled as a BREAK contribution, which implies a severe impact on such goal. Indirectly, impacting this goal will affect the

user covering the role of USER MANUAL READER. Of course, this situation affects project success.

In order to avoid this effect on the project success, a quality mechanism is implemented. In fact, the position QUALITY ASSURANCE ENGINEER covering the role USER MANUAL QA ENGINEER will have the goal of ASSURE USER MANUAL QUALITY. This engineer will achieve its goal by performing two tasks: REVIEW AND IMPROVE USER MANUAL, and USE AUTOMATIC GRAMMAR CORRECTOR.

The task USE AUTOMATIC GRAMMAR CORRECTOR has an effect against the effect of the risky task INCLUDE GRAMMAR ERRORS. Since correctors usually find, and correct all grammar errors, using automatic correctors will BREAK the effect of the risky task. In other words, correctors definitely will eliminate risky task effects.

On the other hand, REVIEW AND IMPROVE USER MANUAL will have an effect against the effect of the risky task ALLOW MANUALS DIFFICULT TO READ. However, in this case, corrective task will only HURT the risky task effect. In other words, there is a contribution to avoid the effect of ALLOW MANUALS DIFFICULT TO READ but this risk is not completely eliminated.

Once risk dependencies and corrective actions are modeled, the next step is to verify if analyzed goal named HAVE GOOD USER MANUAL can be satisfied. One can model several satisfaction levels for diagram elements, i.e. satisfied (√), weakly satisfied (√•), weakly denied (X•), undecided (?•) and denied (X).

The first step is to mark *leaves* in our graph with a satisfaction level, depending on its viability. In general, such leaves are tasks, because tasks do not have incoming links. In order to mark goal with its satisfaction level and verify goal viability, the effect of associated tasks is propagated between links taking into account the contribution of such links. Figure 3 shows SR diagram including satisfaction level, which represents viability of each element.

Tasks USE AUTOMATIC GRAMMAR CORRECTOR, ALLOW GRAMMAR ERRORS IN THE MANUALS, REVIEW AND IMPROVE USER MANUAL, and ALLOW MANUALS DIFFICULT TO READ are marked as satisfied because all of them can easily be performed. In other words, they are independently viable. The effect of a satisfied risky task that can BREAK a goal is to deny it. On the other hand, the effect of a corrective task that can BREAK the effect previously mentioned is to satisfy associated goal. However, the effect of a corrective task that HURT the effect previously mentioned is to weakly satisfy associated goal. In this case, the effect of ALLOW GRAMMAR ERRORS IN THE MANUALS is completely eliminated. Nevertheless, the effect of ALLOW MANUALS DIFFICULT TO READ is reduced but not eliminated. In consequence, associated goal named HAVE GOOD USER MANUAL is weakly satisfied.

After this analysis, project managers could decide to just include this USER MANUAL QA ENGINEER if they consider that the weakly satisfied goal HAVE GOOD USER MANUAL is acceptable. On the other hand, if managers consider that the above-mentioned goal has to be satisfied, they could include other risk control elements represented as other roles or simply as other tasks assigned to USER MANUAL QA ENGINEER. Also, another option is to eliminate the possibility of having a documenter with poor writing skills, which can be done by simply replacing him or training him and improving his writing skills. Unfortunately, this option cannot be modeled directly by i*, which is a limitation. i* assumes that roles that insert risks on the project cannot be easily eliminated.

4 CONCLUSIONS

We have outlined a new technique for modeling the analysis and control of risks in software projects using the concept of actor-dependency, and extending its scope to the domain of risk management. The approach presented in this paper can be used to model intentional relationships among the strategic actors. The technique can reason about the opportunities, vulnerabilities, changes, and risks that are associated with software projects, and can incorporate prominently the issues related to risk in the process of system analysis and design.

REFERENCES

Boehm, B.W. and Bose, P., 1994. A Collaborative Spiral Software Process Model Based on Theory W. In *Proceedings of the 3rd International Conference on Software Process*, IEEE Computer Society, Washington DC, 1994.

Charette, R.N., 1989. *Software Engineering Risk Analysis and Management*, McGraw Hill, New York.

Fairley, R., 1994. *Risk Management of Software Projects*, IEEE Software, Vol. 11, pp. 57-67.

Freimut, B., Hartkopf, S., Kontio, J. and Kobitzsch, W., 2001. An Industrial Case Study of Implementing Software Risk Management, In *Proceedings of the 9th ACM SIGSOFT International Symposium on*

Foundations of Software Engineering, Vienna, pp. 277-287.

Gemmer A. and Koch, P., 2003. Rockwell Case Studies in Risk Management, *Proceedings of the 3rd SEI Conference on Software Risk Management*, SEI, Pittsburgh, PA.

Kontio, J., 1997. *The Riskit Method for Software Risk Management*, Version 1.00, Technical Report, CS-TR-3782/UMIACS-TR-97-38, University of Maryland, College Park, MD, USA.

Yu, E, 1997. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering, Proc. of the 3rd Intl. IEEE Symp. Requirements Engineering, January 6-8, 1997, Washington D.C., pp. 226-235.

APPENDIX

The Appendix contains all the figures referred to in the paper.

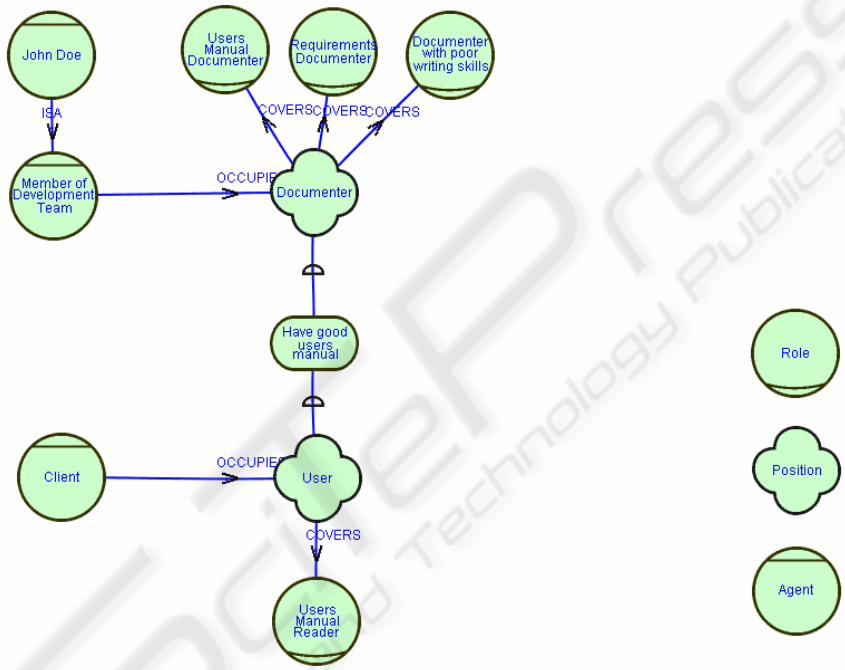

Figure 1: SD diagram with roles, agents and positions for a simple software project.

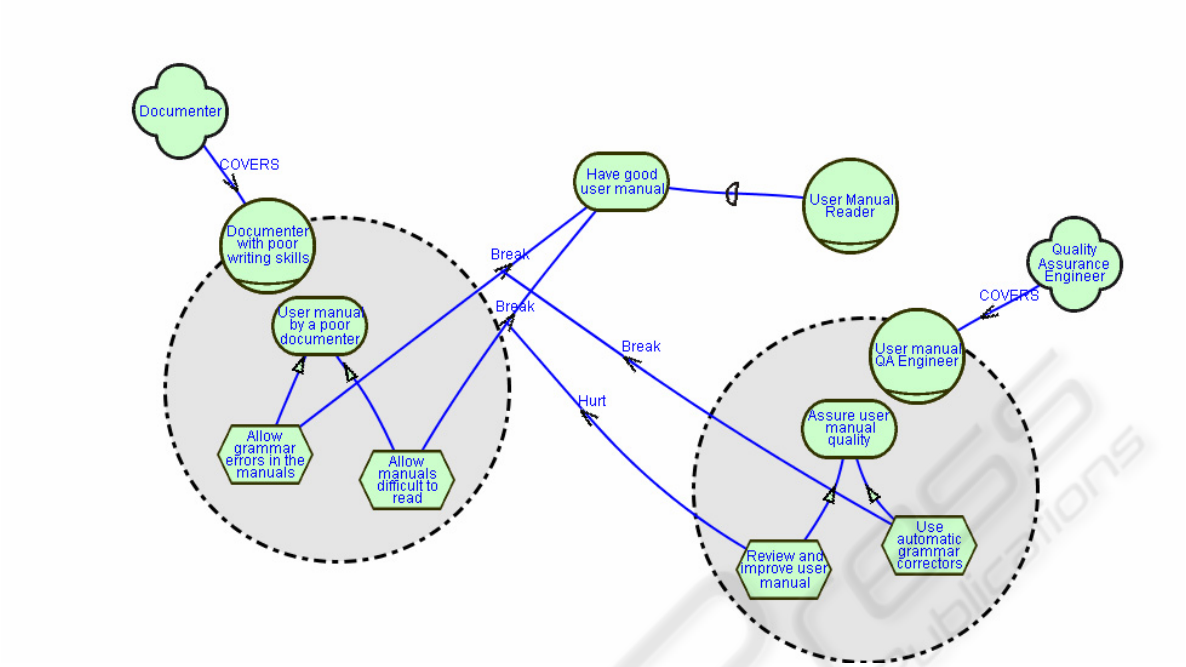

Figure 2: SR diagram showing poor writing skills risk.

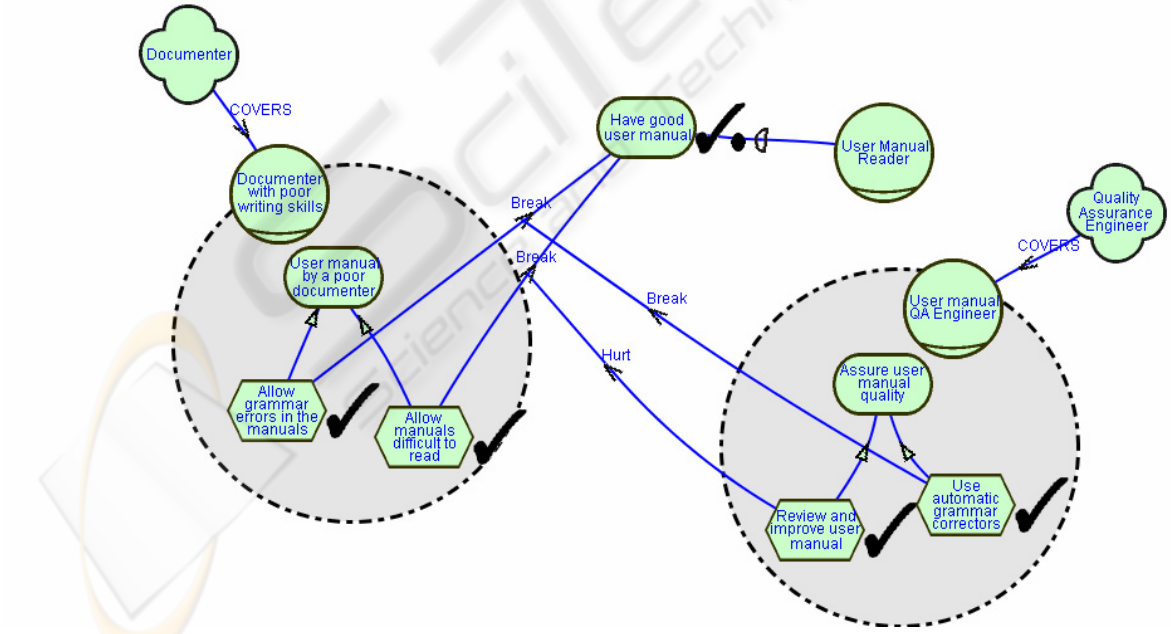

Figure 3: SR diagram showing poor writing skills risk and satisfaction levels.