Software Requirement Risk Assessment Using UML[1]

K.Appukkutty, Hany H. Ammar, Katerina Goseva Popstajanova
Lane Department of Computer Science, West Virginia University
Morgantown WV 26506
{avani, ammar, katerina}@csee.wvu.edu

Abstract

Risk assessment is an integral part of software risk management. There are several methods for risk assessment during various phases of software development and at different levels of abstraction. However, there are very few techniques available for assessing risk at the requirements level and those that are available are highly subjective and are not based on any formal design models. Such techniques are human-intensive and highly error prone. This paper presents a methodology that assesses software risk at the requirements level using Unified Modeling Language (UML) specifications of the software at the early development stages. Each requirement is mapped to a specific operational scenario in UML. We determine the possible failure modes of the scenario and find out the complexity of the scenario in each failure mode. The risk factor of a scenario in a failure mode is obtained by combining the complexity of the failure mode in that scenario and the severity of the failure. The result of applying the methodology on a Cardiac Pacemaker case study is presented.

1. Introduction

Risk is defined as a combination of two factors: probability of malfunctioning (failure) and the consequence of malfunctioning (severity) [1]. There is much ongoing research in the field of reliability, security, and safety-based risk assessment. Most of the current risk assessment methodologies can only be used to estimate risk in the later stages of the software life cycle, typically from design models or code. As a result these methodologies can identify risks but have limited capability in preventing these risks from occurring.

Software requirement risk addresses the possibility of suffering a loss of any functional or non-functional requirement of the software system. It is a well known fact that it is more feasible to make changes to the software system under development in the early stages of the software development cycle. A more efficient risk analysis would be to follow a proactive approach, where the potential risks are identified early and preventive measures are taken to avoid the risks from occurring. Identifying requirement risk at this stage usually is subjective and needs a panel of domain experts to analyze the requirements. Currently available techniques for assessing requirement-based risk are ad hoc in nature and are not based on any popular modeling techniques. They are usually performed in an informal and subjective way by a "risk team". But in reality, not all the software projects have the luxury of having a group of domain experts to carry out this analysis and such an assessment is completely dependant on the opinion of the risk assessment team. To overcome the above mentioned problems our research focused on a method that is less dependent on domain experts and more systematic. This paper introduces a formal methodology that we have developed for requirement based risk assessment at the early design phase, using UML [4] models .Our method relies less heavily on domain experts than the currently used methodologies [6] [7]. The output of our method consists of the risk factors associated with each requirement.

This paper is organized as follows: In Section 2 we present the related work and our contributions. In Section 3 we introduce the methodology and describe in detail, each step involved. In Section 4, we discuss the results obtained by applying the methodology on an example and conclude with discussion of our results and future work.

2. Related Work and Our Contributions

Software risk assessment provides a means for identifying risky components and scenarios of a software product [2]. Risk assessment is done at different levels of software architecture and different stages of software development.

[1] This project is funded by grants to West Virginia University Research Corp. from the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) managed through the NASA Independent Verification and Validation (IV&V) Facility, Fairmont, West Virginia.

0-7803-8735-X/05/$20.00©2005 IEEE

Several formal methodologies have been introduced in the recent years to assess and deal with different kinds of risks associated with software. Gilliam et al introduced a method to reduce software security risk by identifying the vulnerabilities of the software using Vmatrix (vulnerability matrix) and by performing Property-Based Testing (PBT) [10].

A methodology to estimate risk at the architectural level using UML models was introduced in [1]. Risk factors of components and connectors were estimated by combining dynamic complexity and severity ranking of a failure. Then, a Component Dependency Graph (CDG) is developed to assess the risk of the systems. In [2] a similar approach was followed to obtain component and connector risk factors and a Markov model was used to estimate the scenario level risk factors. This approach also identified critical components and connectors in the system. Both these techniques are based on UML models and provide a formal method to assess risk. These approaches require the details of components and connectors involved which are not available until the later stages of the design phase. Moreover, the approaches do not relate the risks to the requirements that will be affected, and because of this it would be difficult to communicate the risk assessment results to the client.

Risk mitigation based on the impact of failures on requirements has been previously addressed by Feather et al [6] [7] in the form of defect detection and prevention. The detection of defects involves identifying failure modes for each requirement and estimating the impact of these failures on the corresponding requirements. This is done at the requirement analysis stage and is not based on any formal design models. The estimation of impact or risk is completely subjective and is based on the opinions of the members of the domain-expert team, who in most cases are not involved in the design of the software.

The above discussed limitations are addressed by our methodology, which is motivated by the need to:

- Assess requirement-based risk of software during the early design phases. It is important to identify the requirements that are at high risk earlier in the software development since preventive measures can be taken prior to the development of the software in order to reduce the occurrence of the failure modes that contribute to requirement-risks.
- Develop a formal methodology for requirement-based risk assessment based on UML. Earlier requirement-based risk assessment techniques were based on analyzing the requirement documents. UML provides a semi-formal method for modeling object oriented software. UML also provides several extensibility features in the form of UML profiles for aid in building models in certain domains, as well as addressing certain modeling problems.
- Make the requirement-risk assessment methodology less dependent on domain experts since, the methods that are solely based on input from domain experts are highly subjective and error prone.

The contributions of this paper are briefly described as follows:

- We present a systematic methodology to estimate requirement-based risk using UML design specifications. The UML specifications are obtained in the early phases of the software development. Our methodology relies less heavily on domain experts and therefore is less subjective and less error-prone.
- We introduce a technique to estimate risk at the requirement level based on previously established formal metrics for estimating error proneness by Fenton et al [5] and severity analysis techniques based on Function Failure Analysis (FFA), which has previously been used in techniques for risk assessment at the architectural level [2].

3. Methodology for estimating requirements risk factors

A UML use case model is defined in the UML specifications [4] as a model that describes a system's functional requirements in terms of use-cases. The use cases are realized by scenarios. Each scenario is modeled by a sequence diagram, which shows the interaction of various components in that scenario. According to pervious risk related work [1], Risk is a "Combination of two factors: probability of malfunctioning (failure) and the consequence of malfunctioning (severity)." The probability of malfunctioning of a system is proportional to its complexity. The more complex a system is, the more prone it is, to faults and failures [5]. Therefore, the complexity of a system is measured in order to estimate the probability of its failure. Since the low level details of the components and connectors of the system are not available during the early design phase, the complexity is calculated at the scenario level, using UML sequence diagrams.

Requirements are mapped to UML use cases. Each requirement contains one or more scenarios, each of which is represented as a sequence diagram. A sequence diagram is defined as a diagram that shows the object interactions arranged in a time sequence [4].

The steps of our requirement-risk assessment methodology are presented in Figure 1. In STEP 1, we identify the use cases and the corresponding scenario and sequence diagrams. In STEP 2, we construct a control flow graph from the sequence diagram. This is fairly straightforward since the sequence diagrams themselves can be viewed as a control flow diagram where the control flows from one component to another in the form of messages and triggers an action or event in the destination component. The control flow graph of the scenario captures all this information in the form of nodes

and arcs. It shows the flow of control with the nodes representing different states of components and the arcs representing the flow of control.

```
STEP 1: Map the requirements to UML
sequence diagrams
For each Sequence diagram
    STEP 2: Construct the control flow
    graph of the scenario from the
    sequence diagram
    STEP 3: Identify the different failure
    modes on the sequence diagram and the
    control flow graph
    For each failure mode
        STEP 4: Assess the severity class
        of the failure mode using Function
        Failure Analysis (FFA)
        STEP 5: Determine the cyclomatic
        complexity of the sub-control flow
        graph
        STEP 6: Measure the number of
        messages exchanged between the
        components in the sequence diagram
        STEP 7: Calculate the complexity of
        the scenario for the failure mode
        as Cyclomatic complexity * Number
        of messages
        STEP 8: Calculate the risk of the
        scenario for the failure mode as
        Complexity * Severity
    End For each failure mode
End for each sequence diagram
```

Figure 1. Requirement risk assessment methodology

A failure mode is based on how the scenario fails during execution. Failure Mode Effects and Criticality Analysis (FMECA) [12] defines a failure mode as the way in which a product or process could fail to perform its desired function. A scenario could fail at several places but we only consider the failure modes based on the scenario inputs or outputs and not on the failure of individual components or connectors involved. In STEP 3, the failure modes of a scenario are captured and marked on the sequence diagram and the control flow graph. The severity (consequences) of each failure mode is assessed in STEP 4 using Functional Failure Analysis, where domain experts come up with the effects of that failure mode. The failure mode is then categorized into one of the four classes, based on its severity: Minor, Marginal, Critical and Catastrophic. Severity indices assigned for each category of failure are 0.25, 0.50, 0.75 and 0.95 respectively. Our previous work in reliability-risk assessment also uses this type of severity classification [2].

In STEP 5, we measure McCabe's cyclomatic complexity (CC) of the sub control flow graph that gives the flow of the scenario to the point where the relevant failure mode occurs.

CC is calculated as:

$$CC = E - N + 2$$

where E is the number of edges and N is the number of nodes in the graph considered. In STEP 6, we count the number of messages (Msg) exchanged between the components from the start to the point where the considered failure mode occurs. Earlier work in metrics by Fenton et al [5] showed that the product of the two metrics, McCabe's cyclomatic complexity and *sigFF,* is a good predictor of fault proneness. *sigFF* is the count of new and modified signals exchanged between the various modules. The metric *sigFF* is analogous to *Msg*, the number of messages that are exchanged between various components in the system. The complexity of the scenario for a specific failure mode is calculated as the product of *CC* and *Msg* for that specific failure mode. By combining these two metrics, we combine the complexity of the control flow of the scenario and also, the number of interactions that take place between the different components. Note that, while calculating the complexity, we do not go into the details of each component but we take into account only the interactions between the various components involved.

In STEP 8, the risk value of a scenario in a specific failure mode is calculated as a product of the normalized complexity of the scenario in that failure mode and the corresponding severity value. The process is repeated for each failure mode and for each scenario. The results are tabulated in the requirement-risk matrix.

4. Results and Conclusion

We have applied the requirement risk assessment methodology on the Pacemaker case study used in [1], [2]. The details of the application of the methodology on the case study are presented in [3]. In this paper only the results are presented. Table 1 shows the requirement risk matrix of the pacemaker case study for 3 different requirements.

Table 1. Requirement risk matrix for the Pacemaker

Requirements	Scenarios	Failure Modes				
		FM1	*FM2*	*FM3*	*FM4*	*FM5*
R1	*AVI*	0.11	0.37	0.475		
R2	*AAI*	0.151	0.283	0.566		
R3	*Prg*				0.017	0.233

The first row in the matrix shows requirement R1 and the corresponding risk values for the various failure modes *FM1*, *FM2* and *FM3* associated with it. The failure modes *FM4* and *FM5* are not relevant for the requirements *R1* and *R2*. Therefore the corresponding cells are left blank.

The requirement risk matrix gives the risk factor of each requirement in specific failure modes. It is evident from the risk values that the requirements *R1* and *R2* are at higher risk when compared to the requirement *R3*, for the identified failure modes. Using this information, the components that contribute in satisfying the requirements *R1* and *R2* should be given more attention during development and tested more rigorously.

The results are shown using a 3-D bar chart in Figure 2. Note that the bars are colored according to their severity category. The results are shown for the scenarios corresponding to the requirements *R1*, *R2* and *R3*.

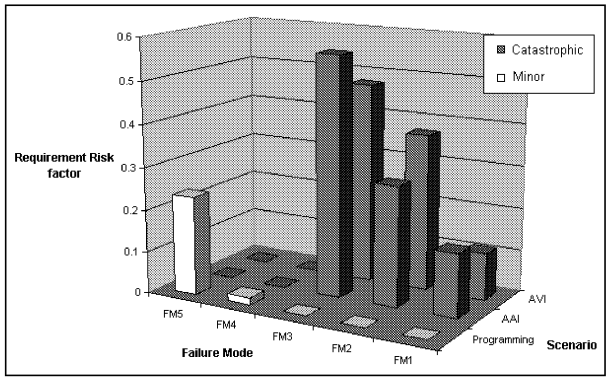

Figure 2. Requirement risk values for the Pacemaker

We have presented a methodology for requirements based risk assessment using the requirements models in UML. This method provides a systematic way to deal with requirements based risk assessment with less input from the domain experts. It fills the gap between completely domain-expert dependant methods that are applied at the requirement analysis stage and formal analytical methods that do not assess risk at the requirements level and require low level implementation details of the components involved. The methodology can be applied during the early phase of software development based on high level design specifications. We have illustrated our methodology using UML but the method can be easily extended to other design paradigms with minor modifications. Future work will focus on developing tool support to assist the analyst with the steps of this methodology. We also intend to apply the methodology and develop supporting tools on large scale applications.

5. References

[1] S. M. Yacoub, H. H. Ammar, "A Methodology for Architecture-Level Reliability Risk Analysis", *IEEE Transactions on Software Engineering*, Vol. 28, No. 6, June 2002, pp. 529-547.

[2] K.Goseva-Popstojanova, A.Hassan, A.Guedem, W.Abdelmoez, D.Nassar, H.Ammar & A.Milli, "Architectural-level Risk Analysis Using UML", *IEEE Transactions on Software Engineering*, Vol. 29, No. 10, Oct 2003, pp. 946-960.

[3] K.Appukkutty, H. H. Ammar, K. Goseva-Popstajanova, "Early Risk Assessment of Software Systems", Submitted to *The 3rd International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications*, Dec 2004.

[4] OMG Unified Modeling Language Specification, Mar 2003, Version 1.5

[5] Norman E. Fenton, Niclas Ohlsson "Quantitative analysis of faults and failures in a complex software system" *IEEE Transactions on Software Engineering,* Vol. 26, No. 8, Aug 2000, pp. 797-814.

[6] M.S. Feather, S.L. Cornford, J. Dunphy & K. Hicks, "A Quantitative Risk Model for Early Lifecycle Decision Making", in *Proceedings of the Conference on Integrated Design and Process Technology*, Pasadena, California, June 2002.

[7] S.L.Cornford, M.S. Feather & K.A. Hicks, "DDP - A tool for life-cycle risk management", in *Proceedings of the IEEE Aerospace Conference*, Big Sky, Montana, Mar 2001, pp. 441-451

[8] A. Hassan, W. Abdelmoez , A. Guedem, K. Apputkutty, K. Goseva-Popstojanova, H. Ammar, "Severity Analysis at Architectural Level Based on UML Diagrams', *The 21st International conference System Safety conference*, Ottawa, Canada, August 2003,pp. 571-580.

[9] A. Hassan, K. Goseva-Popstojanova, H. Ammar, "Methodology for Architecture Level Hazard Analysis, A Survey", *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'03)*, Tunis, Tunisia, July 2003, pp. 68-70.

[10] D.P. Gilliam, J.D. Powell, J.C. Kelly, M. Bishop, "Reducing Software Security Risk through an Integrated Approach", *The 26th Annual NASA Goddard Software Engineering Workshop*, November 2001, pp. 36-42.

[11] UML Profile for Schedulability, Performance, and Time Specification, Version 1.0, September 2003.

[12] Failure Mode, Effects and Criticality Analysis (FMECA) web site, http://www.fmeca.com