Software risk assessment based on UML models

Kalaivani Appukkutty
West Virginia University

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Recommended Citation
https://researchrepository.wvu.edu/etd/2059

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.
Software Risk Assessment based on UML Models

Kalaivani Appukkutty

Thesis submitted to the
College of Mineral and Engineering Sciences
at West Virginia University
in partial fulfillment of the requirements
for the degree of
Master of Science
in
Computer Science

Hany H. Ammar, Ph.D., Chair
Katerina Goseva Popstanjanova, Ph.D.
Mark Shereshevsky, Ph.D.

Lane Department of Computer Science and Electrical Engineering
Morgantown, West Virginia
2004

Keywords: Risk, Software Metrics, Severity, Requirements Risk, Performance Risk, Software Architectures, UML

© 2004 Kalaivani Appukkutty
ABSTRACT

Software Risk Assessment based on UML models

Kalaivani Appukkutty

Risk is the possibility of suffering loss. Risks identified during the early stages of software development are easier and cheaper to handle by making changes to the software architecture. This thesis presents methodologies to assess software risk using Unified Modeling Language (UML) specifications of the software from the early design stages. We present methodologies to assess two types of software risk: Requirements-based risk and Performance-based risk. In assessing requirements-based risk, each requirement is mapped to a specific operational scenario in UML. The risk factor of a scenario in a failure mode is obtained by combining the probability of failure of the scenario and the severity of the failure. For the performance-based risk analysis, we use UML diagrams with performance related annotations, build a software execution model for each scenario and then map it to a system execution model using the deployment information. For estimating the performance-based failures of each scenario we use an asymptotic bounding analysis. The methodologies are applied on various case studies.
ACKNOWLEDGEMENTS

I would like to thank Dr. Ammar for his constant motivation and support for the past two years. He helped me stay focused and to realize my potential for developing innovative solutions to complex research problems. I would also like to thank Dr. Katerina for teaching me the importance of an organized approach to solving a problem and to strive for perfection in any given task.

I am indebted to my parents for their love and affection without which none of this would have been possible. Thanks for believing in me and cheering me on through my most difficult challenges.

Finally my friends and colleagues for making my days in WVU a memorable experience: Rajesh, Naveen, Amar, Yue, Greg, Shashi, Ahmed and Nathan.
# TABLE OF CONTENTS

1. INTRODUCTION ................................................................................................................. 1

1.1 PROBLEM STATEMENT .................................................................................................. 1

1.2 BACKGROUND AND PREVIOUS WORK ....................................................................... 2

   1.2.1 The Unified Modeling Language ........................................................................... 2

   1.2.2 Requirements based risk related work ................................................................. 3

   1.2.3 Performance based risk related work ................................................................... 5

1.3 RESEARCH OBJECTIVES .............................................................................................. 6

1.4 THESIS OUTLINE ......................................................................................................... 8

2. REQUIREMENTS-BASED RISK ASSESSMENT ................................................................. 9

   2.1 METHODOLOGY FOR ESTIMATING REQUIREMENTS-BASED RISK FACTORS ......... 9

   2.2 CASE STUDY: THE CARDIAC PACEMAKER ............................................................ 13

   2.3 RESULTS .................................................................................................................... 19

3. PERFORMANCE-BASED RISK ASSESSMENT ..................................................................... 22

   3.1 METHODOLOGY FOR ASSESSMENT OF PERFORMANCE BASED RISK ............... 23

   3.1.1 STEP 1: Assign demand vectors and build a Software Execution Model .......... 24

   3.1.2 STEP 2: Add hardware platform characteristics and conduct stand-alone analysis ....................................................................................................................... 26

   3.1.3 STEP 3: Estimate probability of failure ............................................................... 28

   3.1.4 STEP 4: Conduct severity analysis ..................................................................... 30

   3.1.5 STEP 5: Estimate the performance risk and identify high-risk components ....... 31

   3.2 THE E-COMMERCE EXAMPLE .................................................................................. 32

   3.3 APPLYING THE RISK ASSESSMENT METHODOLOGY ON THE PLACE REQUISITION
      SCENARIO ...................................................................................................................... 33

   3.4 APPLYING THE RISK ASSESSMENT METHODOLOGY ON THE OTHER SCENARIOS ... 43

4. CONCLUSION AND FUTURE WORK ............................................................................... 47

REFERENCES .......................................................................................................................... 49

APPENDIX ............................................................................................................................. 53
LIST OF FIGURES

Figure 1. Requirement risk assessment methodology .......................................................... 10
Figure 2. Use case view of the Pacemaker case study ............................................................ 14
Figure 3. Sequence diagram for AVI scenario with failure modes and Msg values ............ 16
Figure 4. Control flow diagram for AVI scenario with failure modes ............................... 17
Figure 5. Requirement risk values for Pacemaker case study ........................................... 20
Figure 6. Methodology for assessment of performance-based risk ..................................... 24
Figure 7. Sequence diagram annotations ......................................................................... 25
Figure 8. Annotated deployment diagram ......................................................................... 27
Figure 9. Asymptotic bounds and failure probability estimate ......................................... 29
Figure 10. The use case diagram of the e-commerce example ........................................... 33
Figure 11. Sequence diagram for the Place Requisition scenario ....................................... 34
Figure 12. Execution graph for the Place Requisition scenario ......................................... 36
Figure 13. Deployment diagram for the e-commerce system ............................................. 37
Figure 14. The asymptotic bounds of the Place Requisition scenario ................................ 40
Figure 15. System level sequence diagram for Place Requisition scenario ........................ 41
Figure 16. Risk factors for the scenarios in the e-commerce case study ............................. 44
Figure 17. Residence time of the components in various scenarios ..................................... 46
Figure 18. The use case diagram of the Thermal Ctrl System ............................................. 53
Figure 19. Sequence diagram for Retry_Both_Pumps scenario ......................................... 55
Figure 20. Control flow graph of Retry_Both_Pumps scenario ......................................... 56
Figure 21. Requirement risk graph for Thermal Control System ........................................ 58
Figure 22. Use case view of EOS-commanding system ....................................................... 60
Figure 23. Sequence diagram for Preplanned emergency command transmission ............. 61
Figure 24. Sequence diagram for Handle transmission failure scenario ............................ 62
Figure 25. Execution graph for Transmit preplanned command and Retransmit ................ 63
Figure 26. Deployment diagram for EOS commanding system .......................................... 64
Figure 27. Asymptotic bounds and failure probability estimate ........................................ 66
Figure 28. Identification of high-risk components .............................................................. 67
LIST OF TABLES

Table 1. Requirement risk matrix .......................................................................................... 13
Table 2. Complexity of AVI scenario in the various failure modes ....................................... 18
Table 3. FFA table for failure mode FM1 ............................................................................... 19
Table 4. Requirement risk matrix for Pacemaker case study .................................................. 19
Table 5. Data types and sizes ................................................................................................ 34
Table 6. Annotations for Place Requisition sequence diagram .............................................. 35
Table 7. Mapping of components to nodes in the deployment diagram .................................... 37
Table 8. Service times of the hardware platform devices ........................................................ 38
Table 9. The demand vectors of the Place Requisition scenario .............................................. 39
Table 10. FFA table for Place Requisition scenario ................................................................. 42
Table 11. Performance risk table for various scenarios of the e-commerce case study .......... 44
Table 12. FFA table for Retry_Both_Pumps scenario ............................................................... 57
Table 13. Complexity for Retry_Both_Pumps scenario ......................................................... 57
Table 14. Requirement risk matrix for Thermal Control System ............................................. 57
Table 15. Software units for EOS commanding system .......................................................... 64
Table 16. Hardware units for EOS commanding system ........................................................ 64
Table 17. Resource requirements for Preplanned emergency scenario .................................... 65
Table 18. Resource requirements for Transmission failure scenario ....................................... 65
Chapter 1

Introduction

Risk is defined as a combination of two factors: probability of malfunctioning (failure) and the consequence of malfunctioning (severity) [22]. Various measures are taken to assess risk during software development in order to estimate the possible risks that might occur and eliminate them by making relevant changes. Software risk is assessed at various stages of software development and at various levels of software architecture.

This thesis focuses on the assessment of two kinds of Risk: Requirements based risk and Performance based risk.

1.1 Problem Statement

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 development 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. It is a 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. Risk assessment should therefore be performed during the early stages of software development in order to avoid the high costs of
modifying software at the later stages. Risks that are assessed closer to the release of 
software are mostly infeasible to prevent, since making changes to software architecture, 
design or code at that stage is very expensive. There is a need for a more efficient risk 
analysis that follows a proactive approach, where the potential risks are identified during 
the early phases of software development life cycle and preventive measures are taken to 
avoid the risks from occurring.

1.2 Background and Previous work

Software risk assessment is a means to identify risky components and scenarios of 
software during its development [11]. Risk assessment is done at different levels of 
software and different stages of software development. Several formal methodologies 
were introduced in the recent years to assess and deal with different kinds of risks 
associated with software. Our risk assessment methodologies are based on UML 
 specifications.

1.2.1 The Unified Modeling Language

The Unified Modeling Language (UML) is a language for specifying, 
constructing, visualizing, and documenting the artifacts of a software-intensive system 
[3]. It was developed by OMG as a step towards integrating the best practices in the 
industry to view the levels of abstraction, domains, architectures, life cycle stages and 
implementation technologies. UML provides a formal method for modeling object 
oriented software. It is language independent and it can be tailored to a specific domain. 
UML provides several extensibility features to aid in building models in certain domains 
as well as addressing certain modeling problems. A collection of such extensions that
address a particular modeling problem or a problem domain can be grouped together to form a UML profile. OMG has also introduced UML profile for performance, schedulability and time specifications [11]. Most software projects today follow UML modeling. UML modeling is used especially during the design stages of software, therefore, risk assessment using UML models will aid in early identification of software risk and hence modifications to software to prevent those risks can be performed early.

1.2.2 Requirements based risk related work

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) [3]. The CORAS project [8] introduces model-based risk assessment methodology for security-based risks using UML models.

A methodology to estimate risk at the architectural level using UML models is introduced in [22]. Risk factors of components and connectors are estimated by combining dynamic complexity and severity ranking of a failure. Further Component Dependency Graphs (CDG) are developed to assess the risk of subsystems in a hierarchical architecture. In [11] a similar approach is followed to obtain component and connector risk factors and a Markov model is used to come up with scenario-level risk and system-level risk factors. This approach also identifies 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 need details of the 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.

Identifying requirements-based risk at an early stage is usually 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.

Risk analysis based on the impact on requirements has been previously addressed by Feather et al [14] [21] in the form of defect detection and prevention. This approach considers requirements and the various failure modes that occur as an obstruction in satisfying the requirements. The impact of each failure mode on the requirements is estimated by a group of domain experts and is represented using an impact matrix. A cost matrix is then developed with a list of measures to be taken to prevent the failures and their corresponding cost. The two matrices are then combined to find a balanced solution for the cost of prevention measures and the risks involved. This is a completely subjective approach and is performed at the requirement analysis stage. The analysis is not based on any formal design models. The risk assessment 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.
1.2.3 Performance based risk related work

Performance-based analysis measures risk in terms of the non-functional performance attributes of the system and is estimated using the architectural level design information from UML specifications. Several approaches have been presented in the last few years that focus on embedding performance (and other non-functional properties) information in UML software models using the UML extensions [26] [19].

A framework that allows UML diagrams to be used for building performance models is presented in [12]. Performance modeling is carried out based on a precise textual notation, called Performance Modeling Language, which represents the UML characteristics relevant to performance models. These UML based performance models are then transformed into stochastic queuing networks with simultaneous resource possession. Queues are derived from Class Diagram, workload from Collaboration Diagram, and service demands are partially derived from triggering properties of Class Diagram. Merseguer et al [15] [16] presented an extension of the UML notation to performance annotations (pa-UML), to embed performance related information in UML.

A framework that allows UML diagrams to be used for building performance models is presented by Kahkipuro [12]. Dimitrov et al [5] presented various kinds of UML extensions that are used to annotate and embed performance aspects of various UML diagrams. A UML-driven framework is presented and several interesting and useful approaches of direct and expanded extensions to UML are presented, such as, load-and-time weighted use case diagrams, sequence and activity diagrams with time information, state diagrams with transition probability and most importantly deployment diagrams (which map the various software components to the system hardware platform). Smith [23] [24] presented the conversion of various parameterizations of the execution graph
with demand vectors, to convert it into an Execution Graph (EG). This is called the Software Execution Model [23]. A more extensive approach has been introduced in [4], where asynchronous communication patterns and concurrent action executions have also been considered. The performance-based risk analysis presented in [27] defines the performance failure on a scenario level and quantifies risk as a combination of the probability failure and the severity of the failure. An asymptotic bounding analysis to come up with the asymptotic bounds on throughput and response time of a scenario (modeled by a software execution graph) bounding analysis is presented in [13].

1.3 Research objectives

The risk assessment methodologies presented in this thesis are motivated by the need to:

- Assess requirement-based risk of software during the early design phases of development. 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 so as to reduce the occurrence of the failure modes that contribute to requirement-risks.

- Develop a formal methodology based on UML for requirement-based risk assessment. Earlier requirement-based risk assessment techniques are based on analyzing the requirement documents. UML provides a formal method for modeling object oriented software. UML also provides several extensibility features in the form of UML profiles to aid in building models in certain domains as well as addressing certain modeling problems.
- Develop a requirement-risk assessment methodology that relies less heavily on domain experts since, methods solely based on input from domain experts are highly subjective and error prone.

- Develop a methodology for performance based risk assessment that takes into account the severity of performance failures and uses the stereotype and annotation tags introduced in the UML profile for performance, schedulability and time.

The contributions of this thesis are briefly described as follows:

- We present a formal 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 lesser on domain experts and therefore is less subjective and less error-prone. The technique to estimate risk at the requirement level is based on previously established formal metrics for estimating error proneness by Fenton et al [17] and severity analysis techniques based on Function Failure Analysis (FFA).

- We introduce a methodology for performance based risk assessment that jointly takes into account the performance and the risk attributes on UML software models. The risk factors that result from applying our methodology to UML models incorporate the probability of violating a performance requirement and the severity of the consequences. Therefore, on one hand, they can be used to induce critical component refinements, and on the other hand, to devise critical scenarios that may remain hidden in a pure performance analysis. The main contribution of this thesis towards this
methodology is to assign demand vectors, add hardware platform characteristics and perform stand alone analysis (step 1 and step 2 of the methodology).

1.4 Thesis Outline

This Thesis is organized as follows:

Chapter 2 deals with the steps in the requirements-based risk assessment methodology and the illustration of the methodology on the cardiac pace maker case study.
Chapter 3 explains the steps in the performance-based risk assessment methodology and illustrates the methodology with the E-commerce case study.
Finally, Chapter 4 concludes with discussion about future work.
The Appendices show other case studies, on which the methodologies were applied.
Chapter 2

Requirements-based Risk Assessment

Software requirement risk is the possibility of suffering a loss of any functional or non-functional requirement of the software system. This chapter presents our methodology for requirements-based risk assessment at the early design phase, using UML models. Our method relies less heavily on domain experts than the currently used methodologies [14] [21]. As a result, we obtain the risk factors associated with each requirement.

2.1 Methodology for estimating requirements-based risk factors

A UML use case model is defined in the UML specifications [18] as a model that describes a system’s functional requirements in terms of use-cases. We map requirements to UML use cases and 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 previous risk-related work [22], 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 it is prone to faults and failures. Therefore, the complexity of a system is measured in order to come up with 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 [18].

The steps of our requirement-risk assessment methodology are presented in Figure 1.

**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:** For the failure mode, measure the number of messages exchanged between the components in the sequence diagram.

**STEP 7:** Complexity of the scenario for the failure mode is calculated as

\[ \text{Cyclomatic complexity} \times \text{Number of messages} \]

**STEP 8:** Risk of the scenario for the failure mode is calculated as \( \text{Complexity} \times \text{Severity} \)

End For each failure mode

End for each sequence diagram

Figure 1. Requirement risk assessment methodology

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 diagram can be viewed as a control flow diagram where the control flows from one component to another in the form of messages
that trigger 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 control flow arcs. It gives a sequential flow of control with the nodes representing different states of components and the arcs representing the flow of control. This methodology has been published as [9] and has been applied on an illustrative case study in [10].

A failure mode is based on how the scenario fails during execution. Failure Mode Effects and Criticality Analysis (FMECA) [6] 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 identified and are marked on the sequence diagram and the control flow graph. The severity (consequences) of each one of the failure modes is assessed in STEP 4 using Functional Failure Analysis, where each failure mode is analyzed by domain experts based on 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 a similar classification [11].

In STEP 5, we measure McCabe’s cyclomatic complexity (CC) of the sub control flow graph that shows 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 software metrics by Fenton et al [17] shows 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 combined metrics appear to be better in terms of measuring complexity than both sigFF and Cyclomatic complexity on their own and also better than the size metric” [17]. The metric sigFF is analogous to Msg or 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 as shown in Table 1. The first Column contains the requirements of the system (R1 through Rm). Each requirement is realized using one or more scenarios. The second column shows the scenarios that fulfill the requirements. Requirement R1 is realized by two scenarios S1...
and $S_2$. Requirement $R_2$ is realized by one scenario $S_3$. The failure modes are represented from column 3.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Scenarios</th>
<th>Failure Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>$S_1$</td>
<td>$R_f$</td>
</tr>
<tr>
<td></td>
<td>$S_2$</td>
<td></td>
</tr>
<tr>
<td>$R_2$</td>
<td>$S_3$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_m$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Requirement risk matrix

$FM_1$ through $FM_n$ are the failure modes. The value $R_f$ is the risk factor of scenario $S_1$ (that partially fulfills requirement $R_1$) in failure mode $FM_2$. We fill in values for the cells only if the failure mode is relevant to a particular scenario. For example, scenario $S_1$ may fail only in failure mode $FM_2$. In that case, we only fill the risk factor for the cell corresponding to row $S_1$ and column $FM_2$ and leave the rest of the row of scenario $S_1$ blank. We only consider the mapping of one requirement to multiple scenarios and not vice versa.

2.2 Case study: The Cardiac Pacemaker

The methodology is illustrated using the pacemaker example. This example was used in the previous risk analysis methodologies presented in [22] and [11]. The pacemaker is an implanted medical device for regulating the heart beat. The purpose of a cardiac pacemaker is to stimulate the heart when there is a block in the heart or when the heart rate is slow. The device listens to the heart’s native electrical rhythm, and if it does not sense any electrical activity within a certain time period, it stimulates the heart with a
preset amount of energy, in the form of an electrical pulse. The listening phase is called sensing and the stimulation is called pacing.

The pacemaker is a critical real-time application. An error in the software operation of the device can cause loss of the patient's life. The basic requirements of this device are to sense and pace the heart at the proper time and for a proper duration. Failure to achieve these requirements can cause severe heart conditions. Therefore, this is an apt case study for requirement risk analysis, using which we can assess the risk of each requirement and modify the design and development process to reduce high risks.

The use case model of the Pacemaker, shown in Figure 2 shows the different operating modes the Pacemaker works in and the external actors involved. The pacemaker runs in either a programming mode or in one of the operational modes. During programming, the programmer specifies the type of the operation mode in which the device will work.

Figure 2. Use case view of the Pacemaker case study
The operation mode depends on whether the Atrium (A), Ventricle (V), or both are being monitored or paced. The programmer also specifies whether the pacing is inhibited (I), triggered (T), or dual (D). The various components of the pacemaker are as follows: [22]

Reed_Switch (RS): A magnetically activated switch that must be closed before programming the device. The switch is used to avoid accidental programming by electric noise.

Coil_Driver (CD). Receives/sends pulses from/to the device programmer. These pulses are counted and then interpreted as a bit of value zero or one. These bits are then grouped into bytes and sent to the communication gnome.

Communication_Gnome (CG). Receives bytes from the coil driver, verifies these bytes as commands, and sends the commands to the Ventricular and Atrial models. It sends positive and negative acknowledgments to the coil driver to verify command processing.

Ventricular_Model (VT) and Atrial_Model (AR). These two components are similar in operation. They both could pace the heart and/or sense heartbeats. Once the pacemaker is programmed, the magnet is removed from the Reed_Switch. The Atrial_Model and Ventricular_Model communicate together without further intervention. Only battery decay or some medical maintenance reasons force reprogramming.

For illustration purposes, we have shown the methodology applied on the AVI scenario. In this scenario, the Ventricular (VT) component senses the heart beat and when the heart misses a pulse, it sends a signal to the Atrial (AR) component and the AR component paces the heart. The sequence diagram for the AVI scenario is shown in Figure 3.
Figure 3. Sequence diagram for AVI scenario with failure modes and Msg values
Figure 4. Control flow diagram for AVI scenario with failure modes
The possible failure modes (at the requirements level) for this scenario are: Heart beat not sensed by the Ventricular component (FM1), Heart not paced by the Atrial component (FM2), Heart is kept pacing i.e., pacing does not end (FM3). The Msg value is counted for each of the failure mode. The failure modes and the corresponding Msg values are also marked at the places where they occur in the sequence diagram in Figure 3.

The control flow diagram for the AVI scenario is shown in Figure 4. The places where the three failure modes FM1, FM2 and FM3 occur are marked on the control flow diagram. The cyclomatic complexity is measured from the control flow graph, for each failure mode.

The complexity of the scenario in each failure mode is calculated. The complexity of the AVI scenario is calculated as the product of CC and Msg values. The complexity is normalized for use in the further calculation of risk. The calculation is shown in Table 2.

<table>
<thead>
<tr>
<th>Failure Modes</th>
<th>Cyclomatic Complexity (CC)</th>
<th>Number of Messages (Msg)</th>
<th>Complexity (CC * Msg)</th>
<th>Normalized Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM1</td>
<td>3</td>
<td>4</td>
<td>12</td>
<td>0.111</td>
</tr>
<tr>
<td>FM2</td>
<td>6</td>
<td>7</td>
<td>42</td>
<td>0.389</td>
</tr>
<tr>
<td>FM3</td>
<td>6</td>
<td>9</td>
<td>54</td>
<td>0.500</td>
</tr>
</tbody>
</table>

Table 2. Complexity of AVI scenario in the various failure modes

The severity of each failure mode is determined using Functional Failure Analysis [2] [1], where each failure mode is analyzed by domain experts in coming up with the effects of that failure mode. The result of FFA analysis is a table that shows the event that
is associated with the failure, the kind of failure, the effects of the failure and its severity.

For example, the FFA table for the failure mode \( FM1 \) is shown in Table 3.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Failure Mode</th>
<th>Effects</th>
<th>Severity</th>
</tr>
</thead>
</table>
| AVI      | Heart not sensed \((FM1)\) | ▪ If a pulse is missed, the heart will not be paced  
▪ Might cost the life of the patient | Catastrophic |

Table 3. FFA table for failure mode FM1

The numerical index for the severity class ‘Catastrophic’ is 0.95 [11]. Since all the failure modes in the AVI scenario fall into the catastrophic severity category, their severity value is 0.95. The risk factor for each failure mode in AVI scenario is calculated as the product of the normalized complexity and its severity value.

2.3 Results

The result is in the form of the requirement risk matrix. The requirement risk matrix shows the requirement risk associated with each failure mode for a given requirement. For example 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. This is shown in Table 4.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Scenarios</th>
<th>Failure Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( FM1 )</td>
</tr>
<tr>
<td>( R1 )</td>
<td>AVI</td>
<td>0.11</td>
</tr>
<tr>
<td>( R2 )</td>
<td>AAI</td>
<td>0.151</td>
</tr>
<tr>
<td>( R3 )</td>
<td>Programming</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Requirement risk matrix for Pacemaker case study
The failure modes \textit{FM1}, \textit{FM2} and \textit{FM3} are common to the requirements \textit{R1} and \textit{R2}. Requirement \textit{R3} corresponds to the programming scenario, where the pacemaker is programmed. The failure modes \textit{FM4} and \textit{FM5} are relevant to this requirement alone and the severities of these failure modes are marginal. None of the failure modes involved fall into the critical or minor categories. The failure modes \textit{FM4} and \textit{FM5} are not relevant for the requirements \textit{R1} and \textit{R2}. Therefore the corresponding cells are left blank. The blank cells are “don’t cares”.

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 \textit{R1} and \textit{R2} are at higher risk when compared to the requirement \textit{R3}, as the risk values of \textit{R1} and \textit{R2} are higher than that of \textit{R3} for the identified failure modes. Using this information, the components that contribute in satisfying the requirements \textit{R1} and \textit{R2} can be given more attention during development and can be rigorously tested.

![Risk Factor Chart]

\textbf{Figure 5. Requirement risk values for Pacemaker case study}
The results are shown using a 3-D bar chart in Figure 5. 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 \( R3 \).

Note that the failure mode \( FM3 \) corresponding to requirements \( R1 \) and \( R2 \), is more risky than \( FM1 \) and \( FM2 \). \( FM3 \) is the case that indicates that component AR keeps pacing the heart. Practically, it certainly involves more risk when compared to missing a pace \( (FM2) \) or failing to sense the heart beat \( (FM1) \).

The methodology was applied on a larger, more complex case study. The details of the case study and the results are available in the Appendix.
Chapter 3

Performance-based Risk Assessment

This chapter presents performance-based risk analysis. Performance is a nonfunctional software attribute that plays an important role in application domains ranging from safety-critical systems to e-commerce applications. We introduce the concept of performance-based risk as risk resulting from the failure of scenarios to meet the specified performance requirements. In particular we are interested in performance attributes such as response time and throughput. Performance-based risk is defined as a combination of the probability of violating a performance objective/requirement and the severity of that violation.

We define performance failure as an unexpected performance result originated from the violation of a non-functional requirement (or objective). Since performance requirements are usually expressed in terms of time, a requirement violation is said to occur when a certain operation takes too long to be completed. This type of failure follows faults that concern system performance rather than system functionalities. For example, the extra time taken for an operation to complete may have been spent in a device that has been saturated due to the heavy system workload. Thus, even though the software system is functionally correct, it may suffer from performance failures. In some cases, these failures may have worse consequences than the functional failures. The performance-based risk analysis is based on annotations of the UML diagrams that
support such analysis. The methodology derives the software and hardware parameters from the annotated UML sequence diagrams and deployment diagrams and provides the values for performance-risk for a specified workload.

3.1 **Methodology for Assessment of Performance Based Risk**

Our methodology to estimate the risk factor depends on performance failures of a software system modeled with UML diagrams. The risk model that we introduce combines the probability and severity of performance failures, where such a failure basically comes from a violation of a performance requirement. There are several distinguished types of performance requirements. We focus only on time-related requirements, that is, we only consider performance failures due to an excessively long completion time of certain scenarios.

We assume that system scenarios (i.e., the sequences of actions that a software system performs in order to react to an external trigger) are modeled using UML Sequence Diagrams. Therefore, our observation point to estimate the probability of a specific performance failure is a single Sequence Diagram. In order to estimate the probability of such a performance failure (i.e., the completion time of the operation of the scenario overcomes a required threshold), we build a model which takes into account the time contribution of all the actions performed to complete the specific scenario. Obviously, the estimate of the completion time of a scenario will not be given only from a combination of time contributions of all the actions; the resource contention originated from the system workload will be considered as well. The steps of our methodology for assessment of performance-based risk are shown in Figure 6. Next, we present details of the steps involved in the methodology.
The contributions of this thesis to the performance-risk analysis methodology are steps 1 and 2. All the other steps have been described shortly for the sake of completeness. The details of this methodology is also available in [27].

### 3.1.1 **STEP 1: Assign demand vectors and build a Software Execution Model**

The Sequence Diagram presented in Figure 7 is annotated with information related to the resources that each action/interaction needs in order to be completed. There are two parameters defined for each action/step of a component: 

- **CPU\_{work\_units}** which contains work units as a relative measure of the CPU required to perform this action and 
- **DISK\_{data}** which contains the number of bytes that are read or written to disk to perform this action.

| INPUT: Performance objective. UML diagrams: Use case Diagram, Sequence Diagram (SD), and Deployment Diagram. |
| For each Use Case |
| For each scenario |
| **STEP 1** – Assign demand vector to each action/interaction in Sequence Diagram; build a Software Execution Model |
| **STEP 2** – Add hardware platform characteristics on the Deployment Diagram; conduct stand-alone analysis |
| **STEP 3** – Devise the workload parameters; build a System Execution Model; conduct contention-based analysis and estimate probability of failure as a violation of a performance objective |
| **STEP 4** – Conduct severity analysis and estimate severity of performance failure for the scenario |
| **STEP 5** – Estimate the performance risk of the scenario; Identify high-risk components |

**OUTPUT:**

- Probability of performance-based risk of the scenarios.
- Identification of performance-critical components.

Figure 6. Methodology for assessment of performance-based risk
The interaction/step of a connector is identified by the *Interaction* parameter, which contains the size of data that is being transferred across that connector.

Extensions of UML to represent performance-related concepts have been introduced and accepted by OMG as an UML adopted specification [25]. A step represents an execution of some action.

**Component action (example: A1)**

\[ A1 = [CPU_{\text{work units}}, DISK_{\text{data}}] \]

- **CPU_{\text{work units}}**: CPU work units needed to accomplish the action task
- **DISK_{\text{data}}**:  
  - \( = 0 \) – no access to disk
  - \( \neq 0 \) – size of data to be accessed

**Interaction (example: 1:lbl1)**

\[ lbl1 = [MSG] \]

- **MSG**: size of data exchanged in the interaction

---

Figure 7. Sequence diagram annotations

Each value in the demand vector represents the amount of a resource needed in a unit that depends on the type of resource. Therefore, CPU demand is expressed in terms of CPU work units, disk demand in terms of size of data to be accessed, and network connection demand in terms of size of data to be exchanged. In the following methodological steps we combine hardware device characteristics and demand vectors in order to obtain uniform time-based demands.
The second part of the step 1 is to translate the Sequence Diagram (SD) dynamics into a flow graph. This graph, when parameterized with demand vectors, becomes an Execution Graph (EG), which is a Software Execution Model [23]. Ideas on how to translate sequence diagram patterns into execution graph patterns were first given in [25] [29], even though all the potential patterns were not considered. A more extensive approach was introduced in [25], including asynchronous communication patterns and concurrent action executions. In case of the concurrent executions, only the longest branch is considered for calculating the total demand or the total time taken for the execution of the scenario. The longest branch is the branch of execution that has the highest demand, that is, the branch that takes the longest time to execute. We consider only this branch because the other concurrent branches finish executing before the longest branch does.

3.1.2 STEP 2: Add hardware platform characteristics and conduct stand-alone analysis

In order to translate a demand vector in the elapsed time, we need to know the characteristics of the hardware platform where the software application will be executed. For example, the same number of CPU work units may take a very different time depending on the CPU speed.

In this step we get the hardware platform information from an annotated Deployment Diagram. Thus, each deployment site in the Deployment Diagram is annotated with the number and type of resources (i.e., devices) that it hosts. The annotated Deployment Diagram in Figure 8 shows two servers, A and B, connected via LAN, and the clients, A, B and C, connected to the servers via WAN. The site stereotype gives the set of devices allocated on it. Server-A uses a processor of type CPU-1, Server-B uses a processor of
type CPU-1, and in addition uses a disk of type Disk-1. Similarly, Client A uses a processor of type CPU-2, and client B and C use a processor of type CPU-3.

The service demands for each hardware device in time units $D_i$ are calculated as follows. First, based on the software execution graph of the scenario, we calculate the sum of demands for each hardware device in: work units for the CPUs, KBs for the disks and networks. Then, we multiply these demands by the service times of the corresponding hardware. Since our goal for performance-based risk analysis is to provide conservative estimates, while calculating the completion time of a scenario we consider those branches/paths/alternations which have higher service demands and ignore the others.

The output of the Step 2 is a stand-alone analysis which evaluates the completion time of the whole sequence diagram as it would be executed on a dedicated hardware platform with a single user workload. This stand-alone analysis does not consider delays due to contention for resources. Therefore, if the time value from the stand-alone analysis
violates the performance objective, then the failure probability is equal to 1 without any further investigation, that is, the software system has no feasible implementation under the given set of requirements. Otherwise, it is worth investigating the system behavior under the realistic workload in order to estimate the failure probability in the presence of contention for resources.

### 3.1.3 STEP 3: Estimate probability of failure

In order to build the System Execution Model and conduct contention-based analysis we first need to define the workload intensity in one of the following terms: the arrival rate $\lambda$ (for transaction workload), or the population $N$ (for batch workload), and think time $Z$ (for terminal workload). We consider batch workload; hence, the workload is parameterized by number of customers (population $N$) and the think time is assumed to be zero\(^1\).

A complete system contention-based analysis is based on the parameterization of a System Execution Model (typically a Queuing Network representing the hardware platform topology) with values coming from the synthesis of the Software Execution Model. The parameterized model can then be solved to obtain performance indices. However, we are not interested in actually solving the performance model, but rather in estimating lower and upper bounds on system throughput and response time for a given scenario in order to estimate the performance failure probability. For example, equations defining the asymptotic bounds on the system throughput $X(N)$ and response time $R(N)$ with a batch workload of $N$ customers are given by [13]:

\(^1\) Note that similar results can be derived for other types of workloads.
\[
\frac{1}{D} \leq X(N) \leq \min \left( \frac{N}{D}, \frac{1}{D_{\text{max}}} \right)
\]

\[
\max (D, N \cdot D_{\text{max}}) \leq R(N) \leq N \cdot D
\]

where \( N \) is the number of customers, \( D = \sum D_i \) is the sum of all demands in the scenario, and \( D_{\text{max}} \) is the maximum demand in that scenario.

Figure 9 shows a diagram of the asymptotic bounds on response time \( R(N) \) versus the workload \( N \) (customers). The upper bound \( N \cdot D \) is shown as the line marked with (-o-). The lower bound is estimated as the maximum value of \( D \), shown as line marked with (-+-), and \( N \cdot D_{\text{max}} \), shown as a line marked with (-*-). The values of the actual response time must lie between these three lines. In Figure 9 3.4., we have also shown a 6
seconds response time objective (parallel to x-axis) and a workload of 7 customers (parallel to y-axis).

In order to estimate the probability of performance failure, we partition the workload domain into three zones. In zone $Z_1$ both upper and lower bounds on the response time are below the performance objective, so the probability of failure is zero. The failure probability in zone $Z_3$ is 1 since both bounds fall over the performance objective. In zone $Z_2$ we estimate the failure probability as the ratio between the distance of the upper bound from the performance objective (failure range) and the distance between the bounds (whole range). This estimation approach can be summarized as follows:

$$\text{Failure probability (Z1)} = 0$$
$$\text{Failure probability (Z2)} = \frac{\text{upper bound} - \text{performance objective}}{\text{upper bound} - \text{lower bound}}$$
$$\text{Failure probability (Z3)} = 1$$

3.1.4 STEP 4: Conduct severity analysis

Severity analysis is performed using Functional Failure Analysis (FFA), based on UML use case Diagrams. FFA is performed on system-level sequence diagrams. The system-level sequence diagrams show the system states, the actors involved, and the input and output events. Note that the system-level sequence diagram of a scenario is different from the component/connector-interaction sequence diagram. The internal components and their interactions are not shown in the system-level sequence diagram.

We perform FFA using a subset of refined software HAZOP guidewords [20]. We use the guidewords to analyze each event, to identify its possible failure modes, and the way it contributes to hazards and accidents. The analysis considers each event and decides whether or not hypothetical failure modes are credible and, if they are, what the
consequences might be. For the consequences we use severity classification recommended by MIL_STD_1629A: catastrophic, critical, marginal, and minor. The output results of FFA analysis are recorded in a tabular form which, for each event in the system-level sequence diagram, contains the guideword, failure mode, its effects, and its severity. The details of estimating severity can be found in [2].

3.1.5 **STEP 5: Estimate the performance risk and identify high-risk components**

The performance risk of a scenario is defined as the product of two factors:

- The probability that the system will fail to meet the required performance objective (e.g., desired response time), estimated in STEP 3, and
- The severity associated with this performance failure of the system in the scenario, estimated in STEP 4.

In addition to estimating performance risk of a scenario (i.e., identifying high-risk scenarios), our methodology helps in identifying a set of high-risk components that should undergo more rigorous development, implementation, and testing. For this purpose, we first estimate the overall residence time of each component in a given scenario. In a case of a performance failure in a scenario, the component with the highest residence time is the bottleneck component. Next, we normalize the component’s residence time with the response time of the corresponding scenario. For a component $C_i$ in a scenario $S_j$, the normalized residence time is given by

$$R_{C_i \text{ in } S_j} = \frac{\text{Overall residence time of } C_i \text{ in } S_j}{\text{Response time of } S_j}.$$  

The normalization enables us to compare the component’s residence times across scenarios, that is, to identify high-risk components in a set of scenarios under consideration.
3.2 The E-commerce example

In this section we illustrate our performance-based risk assessment methodology on an e-commerce application which allows customers and suppliers to interact with each other over the Internet. In this type of applications long response times may easily lead customer to change the supplier, with consequent damages such as loss of money and market. Of course, the severity of performance failures (i.e., violations of performance requirements) depends on the type of failure and usually is different for different types of failures.

Shortly, the e-commerce system allows a customer to browse through the various catalogs provided by the suppliers, select the item to be purchased, and place the order. The order is validated by checking that the customer has a contract with the supplier and one or more bank accounts through which payments can be made. The supplier checks for the availability of the product, and if available, ships the product. After receiving the product, the customer sends back an acknowledgement. Finally, the invoice is processed by electronically transferring funds from the customer’s bank account to the supplier’s bank account [7]. The Use case model of the e-commerce application is shown in Figure 10. In the remaining of this section we first apply stepwise the methodology introduced on a given scenario, and then present the results for the other scenarios, including the identification of the high-risk components.
3.3 Applying the risk assessment methodology on the Place Requisition scenario

Step 1: Assign demand vectors and build a Software Execution Model

The Sequence Diagram for the Place Requisition scenario is shown in Figure 11. Each action/step of components is identified by a local state (e.g., CI1, CA3, RA1, CTS1, etc.). A demand vector is assigned to each action/interaction in Figure 11 (CPU work units and the size of data to be written from/to a disk for each action of a component, i.e., the size of data to be sent across network for the interaction parameter of a connector).

The data that are read/written from/to the disk or sent across the network are categorized on the basis of their sizes. The various data types involved and their sizes are shown in Table 5. Scalar represents an acknowledgement or status message, Queries/Requests are categorized into Simple, Average and Complex and the size of the databases involved is given in terms of number of records.
Table 5. Data types and sizes

<table>
<thead>
<tr>
<th>Units</th>
<th>Simple Qy/Req</th>
<th>Average Qy/Req</th>
<th>Complex Qy/Req</th>
<th>Record</th>
<th>Scalar</th>
<th>No. of Inventory</th>
<th>No. of Orders</th>
<th>No. of Customers</th>
<th>No. ofcatalogs</th>
<th>No. of Item/Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>0.08</td>
<td>0.16</td>
<td>0.24</td>
<td>1</td>
<td>0.01</td>
<td>10000</td>
<td>50</td>
<td>500</td>
<td>1000</td>
<td>20</td>
</tr>
</tbody>
</table>

The demand vectors assigned to each action/interaction in Place Requisition sequence diagram presented in Figure 11 are given in Table 6. The interactions are associated with the component actions that produce them. Since the actions RS1 and OS1 do not generate any interactions, the corresponding cells are left blank.
Table 6. Annotations for Place Requisition sequence diagram

<table>
<thead>
<tr>
<th>Component action</th>
<th>$\text{CPU}_{\text{work units}}$ (Work Units)</th>
<th>$\text{DISK}_{\text{data}}$ (KB)</th>
<th>Interaction</th>
<th>MSG (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI1</td>
<td>1</td>
<td>0</td>
<td>Customer Request</td>
<td>0.08</td>
</tr>
<tr>
<td>CA3</td>
<td>3</td>
<td>0</td>
<td>Requisition Request</td>
<td>0.16</td>
</tr>
<tr>
<td>RA1</td>
<td>6</td>
<td>0</td>
<td>Contract Query</td>
<td>0.24</td>
</tr>
<tr>
<td>CTS1</td>
<td>2.699</td>
<td>1</td>
<td>Available Contracts</td>
<td>1</td>
</tr>
<tr>
<td>RA2</td>
<td>4</td>
<td>0</td>
<td>Reserve Funds</td>
<td>0.16</td>
</tr>
<tr>
<td>OFS1</td>
<td>5.699</td>
<td>1</td>
<td>Funds Reserved</td>
<td>0.01</td>
</tr>
<tr>
<td>RA3</td>
<td>6</td>
<td>0</td>
<td>Requisition Data</td>
<td>1</td>
</tr>
<tr>
<td>RS1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA4</td>
<td>2</td>
<td>0</td>
<td>Requisition Status</td>
<td>0.01</td>
</tr>
<tr>
<td>CA4</td>
<td>5</td>
<td>0</td>
<td>Purchase Request</td>
<td>0.24</td>
</tr>
<tr>
<td>DOA1</td>
<td>3</td>
<td>0</td>
<td>Order Data</td>
<td>1</td>
</tr>
<tr>
<td>OS1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA5</td>
<td>2</td>
<td>0</td>
<td>Requisition Status</td>
<td>0.01</td>
</tr>
<tr>
<td>CI3</td>
<td>3</td>
<td>0</td>
<td>Requisition Output</td>
<td>1</td>
</tr>
</tbody>
</table>

The Sequence Diagram for Place Requisition scenario shown in Figure 11 is transformed (by applying the algorithm based on the translation principles illustrated in Appendix) into the Execution Graph shown in Figure 12. Each rectangular node represents a component action, including any interaction produced by it. The rationale behind the translation is that each interaction in a Sequence Diagram is originated by a certain amount of computation in the sending component. Therefore each interaction can be translated into a basic block of an Execution Graph whose demand vector defines the computational and memory load of the action (i.e. CPU and disk demands) as well as the communication load of the interaction (i.e. network demand).

The first node in the Execution Graph denotes an expanded node PLACE-REQ which represents the sequence of the following steps: CI1, CA3, RA1, CTS1, RA2, FS1, and RA3. The triangular nodes represent the splitting branches (i.e., concurrent process sequences that are executed at the same time). Note that the time taken for the execution of the whole scenario does not include the time taken by each individual branch; rather,
we consider only the longest path for calculating the total demand for that scenario [23]. In Figure 12, the path with the highest demand (i.e., the longest path) is shown in bold and it forms the process sequence that takes the longest time and shows the highest demand. We discard two branches (i.e., RS1 and the CA5, CI3 sequence) since, due to their demands, they terminate earlier than the branch including RA4, CA4, DOA1, OS1. Hence, for this particular scenario, we consider steps included in the expanded node PLACE-REQ along with the steps RA4, CA4, DOA1, and OS1.

![Figure 12. Execution graph for the Place Requisition scenario](image)

**Step 2: Add hardware platform characteristics on the Deployment Diagram and conduct stand-alone analysis**

The hardware platform is divided into several subsystems, which differ in processing speed and disk accessing speed. These subsystems interact with each other through different kinds of networks. The Deployment Diagram showing the different subsystems and their communication links is given in Figure 13. Each subsystem
contains a group of related components that reside in a single node, as described in Table 7.

Table 7. Mapping of components to nodes in the deployment diagram

<table>
<thead>
<tr>
<th>Subsystem node</th>
<th>Customer</th>
<th>Supplier</th>
<th>Requisition</th>
<th>Orders</th>
<th>Invoice</th>
<th>Bank</th>
<th>Shared DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of hosted components</td>
<td>Customer Agent (CA)</td>
<td>Supplier Agent (SA)</td>
<td>Requisition Agent (RA)</td>
<td>Delivery Order Agent (DOA)</td>
<td>Invoice Agent (IA)</td>
<td>Bank Interface (BI)</td>
<td>Contracts Server (CS)</td>
</tr>
<tr>
<td></td>
<td>Customer Interface (CI)</td>
<td>Supplier Interface (SI)</td>
<td>Requisition Server (RS)</td>
<td>Orders Server (OS)</td>
<td>Invoice Server (IS)</td>
<td>Bank (external actor)</td>
<td>Operation Funds Server (OPS)</td>
</tr>
<tr>
<td></td>
<td>Customer (external actor)</td>
<td>Supplier (external actor)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Deployment Diagram in Figure 13 shows the name and stereotype of each node. The additional tags of the stereotype are represented between << >> symbols. This
gives the type of CPU and disk type used for a particular node. The networks through which these subsystems communicate are shown as devices with the stereotype of a specific type of network. The communication between the components in the same subsystem is termed as ‘Local’ interaction. The service times of the various CPUs, disks, and networks are given in Table 8.

Table 8. Service times of the hardware platform devices

<table>
<thead>
<tr>
<th>Units</th>
<th>CPU1</th>
<th>CPU3</th>
<th>CPU4</th>
<th>Disk3</th>
<th>Disk4</th>
<th>LAN</th>
<th>External Network</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service time</td>
<td>µsec/WU</td>
<td>µsec/WU</td>
<td>µsec/WU</td>
<td>µsec/KB</td>
<td>µsec/KB</td>
<td>µsec/KB</td>
<td>µsec/KB</td>
<td>µsec/KB</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>50</td>
<td>25</td>
<td>15</td>
<td>10</td>
<td>100</td>
<td>100000</td>
<td>1</td>
</tr>
</tbody>
</table>

Using the demand vectors given in Table 9, information presented in the Deployment Diagram (Figure 13.), and the service times of hardware devices given in Table 8, we estimate the completion time of the Place Requisition scenario. The total demand for each device (in work units for CPUs and KB for disks and network devices) is calculated by combining the demands of the steps in the scenario, excluding the steps that appear in the branches that have been discarded from the Execution Graph of Figure 12 (discarded steps are shown as shaded rows in Table 9). These demands are then multiplied by the corresponding service times given in Table 8 to obtain demands in time units for each hardware device, shown in the last row in Table 9. The completion time of the scenario is estimated as the sum of the demands in time units for all devices.

It follows that the completion time of the Place Requisition scenario is equal to 0.1326 seconds. If we assume as a realistic performance objective on this scenario a response time of 1.5 seconds, then for a stand-alone analysis of the Place Requisition scenario (considering the workload of a single customer), the performance objective is
satisfied. Hence, we proceed with building the system execution model that takes into account the contention-based analysis and provides an estimate of the probability of performance failure in a presence of a realistic workload.

Table 9. The demand vectors of the Place Requisition scenario

<table>
<thead>
<tr>
<th>Processing Step</th>
<th>CPU1</th>
<th>CPU3</th>
<th>CPU4</th>
<th>Disk3</th>
<th>Disk4</th>
<th>LAN</th>
<th>External Network</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units</td>
<td>WU</td>
<td>WU</td>
<td>WU</td>
<td>KB</td>
<td>KB</td>
<td>KB</td>
<td>KB</td>
<td>KB</td>
</tr>
<tr>
<td>CI1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.16</td>
</tr>
<tr>
<td>RA1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.24</td>
</tr>
<tr>
<td>CTS1</td>
<td>2.699</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0.16</td>
<td></td>
</tr>
<tr>
<td>OFS1</td>
<td>5.699</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>RA3</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>RS1</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.01</td>
</tr>
<tr>
<td>CA4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.24</td>
<td></td>
</tr>
<tr>
<td>DOA1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>OS1</td>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>CI3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Total demand (WU or KB)</td>
<td>9</td>
<td>26.398</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1.41</td>
<td>0.41</td>
<td>2.08</td>
</tr>
<tr>
<td>Total demand ($D_i$ in µsec)</td>
<td>90000</td>
<td>1319.897</td>
<td>125</td>
<td>30</td>
<td>10</td>
<td>141</td>
<td>41000</td>
<td>2.08</td>
</tr>
</tbody>
</table>

Step 3: Devise the workload parameters; build a System Execution Model; conduct contention-based analysis and estimate probability of failure as a violation of a performance objective

Since the Place Requisition scenario passed the stand-alone analysis with respect to the assumed performance objective, we move on to build a System Execution Model. We estimate the probability of performance failure using the asymptotic bounds on the response time derived from the queuing network that represents the system execution
model.

Figure 14. The asymptotic bounds of the Place Requisition scenario

Assuming that the performance objective of 1.5 seconds response time must be satisfied under a workload of 15 customers, we plot the graph for the asymptotic bounds in Figure 14. For the considered workload the values of the upper and lower bounds are 2.0295 and 1.35, respectively. Since this workload falls in the zone Z2, as shown in the Figure 14, the performance failure probability is calculated as follows:

\[
\text{Failure probability (Z2)} = \frac{(\text{upper bound} - \text{performance objective})}{(\text{upper bound} - \text{lower bound})}
\]

\[
= \frac{(2.0295 - 1.5)}{(2.0295 - 1.35)} = 0.7792.
\]
Thus, the failure probability for the Place Requisition scenario (under the workload of 15 customers and 1.5 seconds performance objective for the response time) is equal to 0.7792.

**Step 4: Conduct severity analysis and estimate the severity of a performance failure for the scenario**

We consider only the events that occur between the external factors and the system. In this level of abstraction, the system is considered as a black box and the events that occur among the software components of the system are not considered.

The system level sequence diagram of the Place Requisition scenario in fig 11 shows the external events, CustomerInput and RequisitionOutput. These events are then analyzed using FFA by applying the proposed guidewords. Since we are dealing with performance-based risk, we apply only the guideword “LATE”.

![System level sequence diagram for Place Requisition scenario](image)
Results are shown in Table 10. We assign the worst failure mode severity from the FFA table to the severity of a performance failure of the Place Requisition scenario.

Table 10. FFA table for Place Requisition scenario

<table>
<thead>
<tr>
<th>SCENARIO</th>
<th>EVENT</th>
<th>GUIDEWORDS</th>
<th>FAILURE</th>
<th>EFFECTS</th>
<th>SEVERITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Place Requisition</td>
<td>Customer input</td>
<td>Late</td>
<td>n/a</td>
<td>The confirmation message for the order placed takes a long time to be displayed to the customer</td>
<td>Catastrophic</td>
</tr>
<tr>
<td>Request output</td>
<td>Late</td>
<td></td>
<td></td>
<td>• Customer’s time is wasted.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• The customer gets impatient.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• The customer might cancel the order.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• The customer might not order again.</td>
<td></td>
</tr>
</tbody>
</table>

Step 5: Estimate the performance risk of the scenario; Identify high-risk components

The performance risk of a scenario is estimated as a product of the probability that the system fails to meet the required performance objective (i.e., desired response time) and the severity associated with this performance failure of the system in this scenario. We adopt a linear scale for the severity ranking, that is, we assign values 0.95, 0.75, 0.5, and 0.25 to the catastrophic, critical, marginal, and minor severity classes, respectively [22].

Let us consider the Place Requisition scenario. The probability of performance failure estimated in step 3 is equal to 0.7792 and the severity associated with the performance failure of this scenario is catastrophic, and therefore rated as 0.95. Hence, the performance risk associated with Place Requisition scenario is equal to 0.74024.
We also identify the critical components in each scenario by estimating the normalized residence time of the components in that scenario. The components with high residence time in a scenario are identified as the bottleneck components or the high risk components. For the Place Requisition scenario, the component CA (Customer Agent) has the highest normalized residence time. This includes the time taken by the component for the process steps CA3 and CA4 (note that we do not consider CA5 since it has been discarded). The normalized residence time of a component $C_i$ in scenario $S_j$ is calculated as:

$$R_{C_i} \text{ in } S_j = \frac{\text{Overall residence time of } C_i \text{ in } S_j}{\text{Response time of } S_i}$$

The overall residence time of component CA in the place requisition scenario is 0.12 seconds and the response time of Place Requisition scenario is 0.1326. The normalized residence time of CA is $0.12/0.1326 = 0.905$. This implies that 90.5% of total time taken by the scenario is spent in the component CA.

### 3.4 Applying the risk assessment methodology on the other scenarios

Results given in Table 11 have been obtained upon applying the methodology to the other scenarios of the e-commerce case study. The table shows the response time objective in seconds, the workload in number of customers, the probability of failure, the severity of the failure, and the calculated risk factor for each scenario.

Figure 16 shows the bar chart of the performance-based risk factors of all the scenarios in the e-commerce case study. The color of each bar represents the severity associated with the scenario.
Table 11. Performance risk table for various scenarios of the e-commerce case study

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Response time objective</th>
<th>Workload</th>
<th>Probability of failure</th>
<th>Severity value</th>
<th>Risk factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browse Catalog</td>
<td>1.5</td>
<td>11</td>
<td>0.6583</td>
<td>0.25</td>
<td>0.1646</td>
</tr>
<tr>
<td>Place Requisition</td>
<td>1.5</td>
<td>15</td>
<td>0.7792</td>
<td>0.95</td>
<td>0.74024</td>
</tr>
<tr>
<td>Process Delivery order</td>
<td>2</td>
<td>17</td>
<td>0.9145</td>
<td>0.5</td>
<td>0.4573</td>
</tr>
<tr>
<td>Confirm Shipment</td>
<td>3</td>
<td>29</td>
<td>0.3103</td>
<td>0.95</td>
<td>0.295</td>
</tr>
<tr>
<td>Confirm Delivery</td>
<td>1</td>
<td>24</td>
<td>0.4253</td>
<td>0.75</td>
<td>0.319</td>
</tr>
<tr>
<td>Send Invoice</td>
<td>0.08</td>
<td>35</td>
<td>0.2899</td>
<td>0.75</td>
<td>0.21743</td>
</tr>
</tbody>
</table>

Figure 16. Risk factors for the scenarios in the e-commerce case study

The identification of high-risk components is based on the estimated normalized residence time. The graph in Figure 17 shows the components on the x-axis, scenarios on
the y-axis, and the associated normalized residence times (1:100 scale) of the components in the z-axis. The bars are colored according to the severity of the scenario. In each scenario, the component with the highest residence time is a bottleneck.

To illustrate this concept, let us consider the Place Requisition scenario whose sequence diagram is shown in Figure 11. The main source of performance failure in this scenario is the Customer Agent (CA) component, which has the maximum residence time in the scenario, as shown in Figure 17. This is due to the extensive usage of external network by the component CA in that scenario, clearly seen from the demand vectors of process steps CA3 and CA4 given in Table 9. Hence, the most cost effective way to decrease the performance risk of the Place Requisition scenario is to reduce the residence time of the CA component. Similarly, the component Supplier Agent (SA) has the highest normalized residence time in the Confirm Shipment scenario which also has catastrophic severity.

From, we can identify high-risk components for a particular scenario and the high risk components across multiple scenarios. Of course, this information is valuable for managing the performance-based risk. Thus, as shown in Figure 17, the component CA (Customer Agent) has very high residence time in two scenarios (Place Requisition and Confirm Shipment) and a moderately high residence time in one scenario (Browse Catalog). Similarly, the component SA (Supplier Agent) has very high residence time in the Confirm Shipment scenario and a moderately high residence time in the Process Delivery scenario. By identifying components that are critical in more than one scenario, risk management efforts could be properly distributed by prioritizing the components that need to be improved.
Figure 17. Residence time of the components in various scenarios

Similarly, the component SA (Supplier Agent) has a very high residence time in the Confirm Shipment scenario and a moderately high residence time in the Process Delivery scenario. By identifying components that are critical in more than one scenario, risk management efforts could be properly distributed by prioritizing the components that need to be improved.

The methodology was applied on a larger, more complex case study. The details of the case study and the results are available in the Appendix.
Chapter 4

Conclusion and Future Work

We have presented methodologies to estimate requirements-based risk and performance-based risk using UML specifications. The requirements-based risk assessment technique presents a systematic process to assess requirements-based risk with lesser 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 also 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.

The rationale behind the performance-based risk assessment methodology is that performance analysis may not be sufficient in some application domains, like safety-critical systems. A performance failure may or may not cause heavy consequences depending on the severity of the failure. Our methodology annotates UML diagrams with risk related attributes, and translates these diagrams into models that are ready to be evaluated. The methodology highlights the key scenarios in the whole software/hardware system and the bottleneck components in the scenarios which have the higher service demands. This is an important feedback for software designers, that (based on this information) may devise more effort to the design and the implementation (or to the acquisition, in case of COTS) of the most critical components. Since the methodology
does not rely on solving performance models, rather on the evaluation of formulas representing the lower and upper bounds, as well as failure probabilities, its scalability is not an issue.

This work is part of a longer project on which we are working to define a general framework for risk analysis. In the near future we also plan to integrate requirements-based and performance-based risk analysis approaches, and to build a XML-based tool that allows annotating UML diagrams and automatically producing and evaluating risk models. XML technologies have enhanced the potential for interoperability, thus we are confident that in a medium-term view a wider integration of tools for non-functional and functional analysis of UML models can be achieved based on XML.
References


APPENDIX

Requirements-based Risk Assessment: Thermal Control System

The Requirements-based risk assessment methodology was applied to a The Thermal Control System (TCS), which is a part of a complex, mission critical real time system.

Figure 18 gives the use case model of the TCS. The use cases “Setting_1” to “Setting_5” are the different modes the system could be set to work in. These are grouped into a single use case “Mode Setting”. The use cases “Pump_1_Retry”, “Pump_2_Retry” and “Retry_Both_Pumps” are used during an attempt to recover from failure. The “Monitoring” use case involves monitoring if the system works well in any mode that it is set to work in and in case of any failures, recover from it.

Figure 18. The use case diagram of the Thermal Control System
The pumps and valves involved in the system are shown as actors since they are considered external to the software that controls and monitors them. The Sequence diagram and control flow graph of Retry Both Pumps scenario are shown in Figure 19 and Figure 20 respectively. This scenario fails in three different failure modes FM1, FM2 and FM3. The failure modes are marked on the sequence diagram as well as the control flow graph. The Msg and CC values for the corresponding failure modes are also noted in the figures.
Figure 19. Sequence diagram for Retry_Both_Pumps scenario
Figure 20. Control flow graph of Retry_Both_Pumps scenario
The Function Failure Analysis (FFA) table in Table 12 shows the effects and severity of failure mode FM1 in Retry_both_Pumps scenario.

Table 12. FFA table for Retry_Both_Pumps scenario

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Failure Mode</th>
<th>Effects</th>
<th>Severity</th>
</tr>
</thead>
</table>
| Retry_both_Pumps | Pump Failure is not reported (FM1) | • Failure-recovery measures are not taken  
• Temperature not controlled. (Mission Loss) | Catastrophic |

The complexity corresponding to each failure mode in this scenario is calculated based on the Msg and CC values. This is shown in Table 13.

Table 13. Complexity for Retry_Both_Pumps scenario

<table>
<thead>
<tr>
<th>Failure Modes</th>
<th>Cyclomatic Complexity (CC)</th>
<th>Number of Messages (Msg)</th>
<th>Complexity (CC * Msg)</th>
<th>Normalized Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.015</td>
</tr>
<tr>
<td>FM2</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>0.176</td>
</tr>
<tr>
<td>FM3</td>
<td>5</td>
<td>11</td>
<td>55</td>
<td>0.808</td>
</tr>
</tbody>
</table>

Table 14. Requirement risk matrix for Thermal Control System

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Scenarios</th>
<th>Failure Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FM1</td>
<td>FM2</td>
</tr>
<tr>
<td><strong>R1</strong></td>
<td>Retry_Both_Pumps</td>
<td>0.015</td>
</tr>
<tr>
<td><strong>R2</strong></td>
<td>Pump_1 Retry</td>
<td></td>
</tr>
<tr>
<td><strong>R3</strong></td>
<td>Setting_1</td>
<td></td>
</tr>
</tbody>
</table>
Table 14 gives the Requirement risk matrix for the TCS case study based on the key scenarios Retry_Both_Pumps, Pump_1_Retry and Setting_1. No two scenarios in this case fail in the same failure mode. Therefore, the other cells in the matrix are left blank.

![Requirement risk graph for Thermal Control System](image)

Figure 21. Requirement risk graph for Thermal Control System

Figure 21 is a graphical representation of the requirement risk matrix. The risk bars are colored according to the severity of the failure modes.

**Performance-based Risk Assessment: Earth Observing System**

NASA’s Earth Observing System (EOS) is the first observing system to offer integrated measurements of the Earth's processes. It is composed of a series of satellites,
a science component, and a data system supporting a coordinated series of polar-orbiting and low inclination satellites for long-term global observations of the land surface, biosphere, solid earth, atmosphere, and oceans. It is a large scale, geographically distributed data intensive system designed to handle terabytes/day of data.

The Flight Operations Segment (FOS) is a significant subsystem of the EOS. The FOS is responsible for the planning, scheduling, commanding, and monitoring of the spacecraft and the instruments on board. We applied our methodology on the commanding subsystem, which by itself is a large real time system. The commanding system is responsible for transmission of commands from the ground station to the satellite. It also manages the queuing of multiple commands or command groups, proper execution and maintenance of logs.

Figure 22 gives the use case model of the EOS-commanding system. The ground operator and the spacecraft are shown as external actors. Our methodology was applied on a combination of two scenarios: “Preplanned emergency command transmission” scenario and “Handle transmission failure” scenario. Any exceptions occurring in the spacecraft is to be handled by executing certain sets of commands or command groups. The commands to handle some expected or preplanned exceptions are stored in a database. In the Preplanned emergency command transmission scenario, such exceptions are analyzed and the commands to handle these exceptions are retrieved from the database and are sent back to the spacecraft. If the transmission fails, the Handle transmission failure scenario is executed. The retransmission can occur at most twice. We have considered a combination of the two scenarios in assessing the performance-based risk of the commanding subsystem.
Figure 22. Use case view of EOS-commanding system
Figure 23. Sequence diagram for Preplanned emergency command transmission scenario

Figure 23 and Figure 24 show the sequence diagram for the Preplanned emergency command transmission scenario and the Handle transmission failure scenario with the actions and interactions annotated.
Figure 24. Sequence diagram for Handle transmission failure scenario

The execution graph derived from the sequence diagrams is shown in Figure 25.
Figure 25. Execution graph for Transmit preplanned command and Retransmit scenarios

Figure 26 shows the deployment diagram of hardware resources involved in the EOS Commanding subsystem. The figure shows the networks, processors and the databases involved in the system with their corresponding speeds.
Figure 26. Deployment diagram for EOS commanding system

Table 15. Software units for EOS commanding system

<table>
<thead>
<tr>
<th></th>
<th>Scalar</th>
<th>Average Qty</th>
<th>Complex Qty</th>
<th>Simple Record</th>
<th>Average Record</th>
<th>Complex Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units</td>
<td>KB</td>
<td>KB</td>
<td>KB</td>
<td>KB</td>
<td>KB</td>
<td>KB</td>
</tr>
<tr>
<td>Size</td>
<td>0.005</td>
<td>0.3</td>
<td>0.5</td>
<td>100</td>
<td>1000</td>
<td>10000</td>
</tr>
</tbody>
</table>

Table 16. Hardware units for EOS commanding system

<table>
<thead>
<tr>
<th></th>
<th>EOC</th>
<th>ICC</th>
<th>Comm Subsystem</th>
<th>IDB</th>
<th>Ground n/w</th>
<th>Emergency n/w</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units</td>
<td>microsec/KB</td>
<td>microsec/KB</td>
<td>microsec/KB</td>
<td>microsec/KB</td>
<td>microsec/KB</td>
<td>microsec/KB</td>
</tr>
<tr>
<td>Service time</td>
<td>0.0025</td>
<td>0.0025</td>
<td>0.02</td>
<td>60</td>
<td>80</td>
<td>25000</td>
</tr>
</tbody>
</table>

Table 15 shows the software units classified according to their type and size.

Table 16 gives the hardware units from the annotations in the deployment diagram.
Table 17. Resource requirements for Preplanned emergency scenario

<table>
<thead>
<tr>
<th>Processing Step</th>
<th>EOC</th>
<th>ICC</th>
<th>Comm Subsystem</th>
<th>IDB</th>
<th>Ground n/w</th>
<th>Emergency n/w</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units</td>
<td>job</td>
<td>job</td>
<td>Job</td>
<td>Job</td>
<td>job</td>
<td>byte</td>
</tr>
<tr>
<td>EOC1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.005</td>
</tr>
<tr>
<td>ICC1</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>IDB1</td>
<td></td>
<td>7</td>
<td>10000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICC2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10000</td>
</tr>
<tr>
<td>EOC2</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
</tr>
<tr>
<td>ICC3</td>
<td>8</td>
<td></td>
<td>10000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOC3</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>T1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>T2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>0.005</td>
<td></td>
</tr>
<tr>
<td>EOC4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 18. Resource requirements for Transmission failure scenario

<table>
<thead>
<tr>
<th>Processing Step</th>
<th>EOC</th>
<th>ICC</th>
<th>Comm Subsystem</th>
<th>IDB</th>
<th>Ground n/w</th>
<th>Emergency n/w</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units</td>
<td>job</td>
<td>job</td>
<td>Job</td>
<td>Job</td>
<td>job</td>
<td>byte</td>
</tr>
<tr>
<td>R1</td>
<td></td>
<td>3</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOC5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.005</td>
</tr>
<tr>
<td>SE1</td>
<td></td>
<td>4</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOC6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.005</td>
</tr>
<tr>
<td>TC1</td>
<td></td>
<td>2</td>
<td>0.005</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOC7</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.005</td>
</tr>
<tr>
<td>T1</td>
<td></td>
<td>6</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 17 and Table 18 give the resource requirements or demand vectors for the process steps involved in the two scenarios considered.
Figure 27. Asymptotic bounds and failure probability estimate

Figure 27 shows the asymptotic bounds on response time for the above discussed scenarios. The failure probability is calculated as 0.7958 based on a workload of 2 requests per second and a performance objective of 15 seconds. The severity of this failure is assessed as catastrophic. The risk factor of this scenario is 0.756.
Figure 28. Identification of high-risk components

Figure 28 shows the high risk or the critical components involved in the preplanned emergency scenario and the handle transmission failure scenario. Ground (GN) and the Space (ECOM) networks are the most critical components. The service times of the other components are significantly smaller than the service times of GN and ECOM network components and hence are not visible on the graph.