Architectural-Level Risk Analysis using UML

K. Goseva-Popstojanova†, A. Hassan, A. Guedem, W. Abdelmoez, D. Nassar, H. Ammar

Lane Department of Computer Science and Electrical Engineering, West Virginia University
Morgantown, WV26506-6109
{katerina, hassan, guedem, rabie, dmnassar, ammar}@csee.wvu.edu

A. Mili

College of Computing Science
New Jersey Institute of Technology
Newark, NJ 07102
{mili}@cis.njit.edu

Abstract

Risk assessment is an essential part in managing software development. Performing risk assessment during the early development phases enhances resource allocation decisions. In order to improve the software development process and the quality of software products, we need to be able to build risk analysis models based on data that can be collected early in the development process. These models will help identifying the high risk components/connectors, so that remedial actions may be taken in order to control and optimize the development process and improve the quality of the product. In this paper we present a risk assessment methodology which can be used in the early phases of the software life cycle.

† Corresponding author
We use the Unified Modeling Language (UML) and commercial modeling environment Rational Rose Real Time (RoseRT) to obtain UML model statistics. First, for each component and connector in software architecture a dynamic heuristic risk factor is obtained and severity is assessed based on hazard analysis. Then, a Markov model is constructed to obtain scenarios risk factors. The risk factors of use cases and the overall system risk factor are estimated using the scenarios risk factors. Within our methodology we also identify critical components and connectors that would require careful analysis, design, implementation, and more testing effort. The risk assessment methodology is applied on a pacemaker case study.

**Index terms:** Risk assessment, UML specification, software architecture, dynamic complexity, dynamic coupling, Markov model.

1. **Introduction**

   Risk assessment provides useful means for identifying potentially troublesome software components that require careful development and allocation of more testing effort. According to NASA-STD-8719.13A standard [38] risk is a function of the anticipated frequency of occurrence of an undesired event, the potential severity of resulting consequences, and the uncertainties associated with the frequency and severity. This standard defines several types of risk, such as for example availability risk, acceptance risk, performance risk, cost risk, schedule risk, etc. In this study, we are concerned with reliability-based risk, which takes into account the probability that the software product will fail in the operational environment and the adversity of that failure.

   We define risk as a combination of two factors [32]: probability of malfunctioning (failure) and the consequence of malfunctioning (severity). Probability of failure depends on the probability of occurrence of a fault combined with the likelihood of exercising that fault. During the early phases of software life cycle it is difficult to find exact estimates for the probability of failure of individual components and connectors in the system. Therefore, in this study we use quantitative factors, such as
complexity and coupling that are proven to have major impact on the fault proneness [10]. Moreover, to account for the probability of a fault manifesting itself into a failure, we use dynamic metrics. Dynamic metrics are used to measure the dynamic behavior of a system based on the premise that active components are sources of failures [43]. To determine the consequence of a failure (i.e., severity), we apply the MIL_STD 1629A Failure Mode and Effect Analysis as discussed later.

Risk assessment can be performed at various phases throughout the development process. Architecture models, abstract design and implementation details describe systems using compositions of components and connectors. A component can be as simple as an object, a class, or a procedure, and as elaborate as a package of classes or procedures. Connectors can be as simple as procedure calls; they can also be as elaborate as client-server protocols, links between distributed databases, or middleware. Of course, risk assessment at the architectural level is more beneficial than assessment at later development phases for several reasons. Thus, the architecture of a software product is critical to all development phases. Also, early detection and correction of problems significantly less costly than detection at the code level.

In this paper we develop a risk assessment methodology at the architectural level. Our methodology uses dynamic complexity and dynamic coupling metrics that we obtain from the UML specifications. Severity analysis is performed using the Failure Mode and Effect Analysis (FMEA) technique. We combine severity and complexity (coupling) metrics to obtain heuristic risk factors for the components (connectors). Then, we develop Markov model to estimate scenarios risk factors from the risk factors of components and connectors. Further, use cases and overall system risk factors are estimated using the scenarios risk factors.
1.1. Motivation and Objectives

The work presented in this paper is primarily motivated by the need to develop risk assessment methodology based on quantitative metrics that can be systematically evaluated with little or no involvement of subjective measures from domain experts. Quantitative risk assessment metrics are integrated into risk assessment models, risk management plans, and mitigation strategies. This work comes as a continuation of our previous work presented in [1], [42].

This work is also motivated by the need to compute risk factors based on UML specifications such as use cases and scenarios during the early phases of the software life cycle. The approach we pursue in this paper will enable software analysts and developers to:

- Compute the scenarios risk factors,
- Compute the use cases risk factors,
- Compute the overall system risk factor based on use cases and scenarios risk factors,
- Determine the distribution of the scenarios/use cases/system risk factors over different severity classes,
- Generate a list of components/connectors ranked by their relative risk factor, and
- Generate a list of use cases and a list of scenarios (in each use case) ranked by their risk factors.

1.2. Contributions

The contributions of this paper are summarized as follows:

i) We present a lightweight methodology to perform analytical risk assessment at the architecture level based on the analysis of behavioral UML specifications, mainly use cases and sequence
ii) We introduce the notions of scenario/use case risk factors than enable an analyst to focus on high-risk scenarios and/or use cases. This is particularly important for the high-risk scenarios and/or use cases which are executed rarely. Although these scenarios and/or use cases will not contribute significantly to the overall system risk factor as computed in [42], their risk analysis is extremely important due to the fact that they usually provide exception handling of rare but critical conditions.

iii) We develop a Markov model to determine scenarios risk factors using components and connector risk factors. This model provides exact close form solution for the scenarios risk factors, while the algorithm for traversal of the component dependency graphs used in [42] provides approximate solution. Further advantage of the derived closed form solution for the scenarios risk factors is more effective way for conducting sensitivity analysis. Thus, we simply plug different values of the parameters in the closed form solution, while in [42] algorithmic solution is reapplied for each set of different parameters. Using scenarios risk factors we also derive the risk factor of each use case and the overall system risk factor.

iv) The Markov model used for estimating the scenarios risk factors generalizes the existing architecture – based software reliability models [6], [13] in two ways. Thus, while software reliability model presented in [6] considers only component failures, in the scenarios risk models we account for both components and connectors failures, that is, consider both components and connectors risk factors. Further, instead of a single failure state, we consider multiple failure states that represent failure modes with different severities. This approach allows us to derive the distribution of scenarios/use cases/system risk factors over different severity classes which
provide additional insights important for risk analysis. Thus, scenarios and use cases that have risk factors distributed among more severe classes will be more critical and deserve more attention than other scenarios and use cases.

v) Since the approach proposed in this paper is entirely analytical, development of a tool for automatic risk assessment is straightforward. Using Rational Rose Real Time [35] as a front end, we have already developed a prototype of a tool for risk assessment based on the methodology presented in this paper.

The paper is organized as follows. Section 2 describes the well-known cardiac pacemaker system and presents its UML specification based on the use case diagrams and sequence diagrams. Section 3 presents the proposed methodology and its application to the pacemaker example. Section 4 summarizes the related work. Finally, in Section 5, we conclude the paper and discuss directions for future research.

2. A Motivating Example

We have selected a case study of a cardiac pacemaker device [9] to illustrate how our proposed methodology works. A cardiac pacemaker is an implanted device that assists cardiac functions when the underlying pathologies make the intrinsic heartbeats low. An error in the software operation of the device can cause loss of a patient’s life. This is an example of a critical real-time application. We use the UML real-time notion to model the pacemaker. Figure 1 shows the components and connectors of the pacemaker in the capsule diagram. The figure also shows the input/output port to the Heart as an external component, as well as the two input ports to the Reed Switch and the Coil Driver components. A pacemaker can be programmed to operate in one of the five operational modes depending on which part of the heart is to be sensed and which part is to be paced. Next, we briefly describe the components of the pacemaker system.
Figure 1. The architecture of the pacemaker example

- **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 programmer. These pulses are counted and then interpreted as a bit of value zero or one. The bits are then grouped into bytes and sent to the Communication Gnome. Positive and negative acknowledgments, as well as programming bits, are sent back to the programmer to confirm whether the device has been correctly programmed and the commands are validated.

- **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 the 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 the heartbeats. Once the pacemaker is programmed the
magnet is removed from the RS. The AR and VT communicate together without further intervention. Only battery decay or some medical maintenance reasons may force reprogramming.

2.1. The Use Case Model

The pacemaker runs in either a programming mode or in one of five operational modes. During programming, the programmer specifies the operation mode in which the device will work. The operation mode depends on whether the atrial, ventricular, or both are being monitored or paced. The programmer also specifies whether the pacing is inhibited, triggered, or dual. For example, in the AVI operation mode, the atrial portion of the heart is paced (shocked), the ventricular portion of the heart is sensed (monitored), and the atrial is only paced when a ventricular sense does not occur (inhibited mode).

The use case diagram of the pacemaker application is given in Figure 2. It presents the six use cases and the two actors: doctor programmer and patient’s heart. Each use case in Figure 2 is realized by at least one sequence diagram (i.e., scenario).

![Use case diagram of the pacemaker](image-url)

Figure 2. Use case diagram of the pacemaker

8
Domain experts determine probabilities of occurrence of use cases and the scenarios within each use case. This can be done in a similar fashion as the estimation of the operational profile in the field of software reliability [29]. For the pacemaker example, according to [9] the inhibit modes are more frequently used than the triggered mode. Also, the programming mode is executed significantly less frequently than the regular usage of the pacemaker in any of its operational modes. Hence, we assume the probabilities for programming use case and five operational use cases (AVI, AAI, AAT, VVI, and VVT) as given in Table 1.

Table 1. Probabilities of the use cases executions

<table>
<thead>
<tr>
<th>Use case</th>
<th>Programming</th>
<th>AVI</th>
<th>AAI</th>
<th>VVI</th>
<th>AAT</th>
<th>VVT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability</td>
<td>0.01</td>
<td>0.29</td>
<td>0.20</td>
<td>0.20</td>
<td>0.15</td>
<td>0.15</td>
</tr>
</tbody>
</table>

Figure 3 shows the sequence diagram of a scenario in the programming use case. In this use case the programmer interacts with the RS and CD components to input a set of 8 bits specifying an operation mode for the pacemaker. This byte is received by the CG component which in turn sets the operation mode of the AR and VT components to one of five modes (or use cases): AVI, AAI, AAT, VVI, and VVT. Figure 4 shows a scenario from the AVI use case in which the VT senses the heart and the AR paces the heart when a heart beat is not sensed. As in all scenarios, a refractory period is then in effect after every pace.

For the pacemaker example described here only one scenario was available for each use case. However, the methodology presented in the next section is more general and supports multiple scenarios defined for each use case.
3. Risk Analysis Methodology

In this section we introduce our risk assessment methodology. We start by describing the proposed risk analysis process. Then, we describe the techniques for determining the risk factors of components and connectors in a given scenario and present a Markov model for determining scenario risk factor. Next, we present the methods used to estimate use cases and overall system risk factors and conduct sensitivity analysis.

3.1. The Proposed Risk Analysis Process

The use cases and scenarios of a UML specification drive the risk analysis process that we propose in this section. The proposed risk analysis process consists of the steps shown in Figure 5. We assume that the UML logical architectural model consists of a use case diagram defining several independent use cases as shown in Figure 2, and that each use case is realized with one or more independent scenarios modeled using sequence diagrams as shown in Figure 3 and Figure 4.

The proposed risk analysis process iterates on the use cases and the scenarios that realize each use case and determines the component/connector risk factors for each scenario, as well as the scenarios and use cases risk factors. For each scenario, the component (connector) risk factors are estimated as a product of the dynamic complexity (coupling) of the component (connector) behavioral specification measured from the UML sequence diagrams and the severity level assigned by the domain expert using hazard analysis and Failure Mode and Effect Analysis (see section 3.2). Then, a Markov model is constructed for each scenario based on the sequence diagram and a scenario risk factor is determined as described in section 3.3. Further, the use cases and overall system risk factors are estimated (section 3.4). The outcome of the above process is a list of critical scenarios in each use case, a list of critical use cases, and a list of critical components/ connectors for each scenario and each use case.
Figure 3. Sequence diagram of the programming scenario
Figure 4. Sequence diagram of the AVI scenario
For each use case
   For each scenario
      For each component
         Measure dynamic complexity
         Assign severity based on FMEA and hazard analysis
         Calculate component’s risk factor
      For each connector
         Measure dynamic coupling
         Assign severity based on FEMA and hazard analysis
         Calculate connector’s risk factor
   Generate critical component/connector list
   Construct Markov model & Calculate transition probabilities
   Calculate scenario’s risk factor
   Rank the scenarios based on risk factors, Determine critical scenarios list
   Calculate use case risk factor
   Rank use cases based on risk factors, Determine critical use case list
   Determine critical component/connector list in the system scope
   Calculate overall system risk factor

Figure 5. The risk analysis process

3.2. Assessment of the Component/Connector Risk Factors

For each scenario $S_x$, we calculate heuristic risk factors for each component and connector participating in the scenario based on the dynamic complexity, dynamic coupling and severity level. Note that in general these values will be different for different scenarios.

The risk factor $rf_i^x$ of a component $i$ in scenario $S_x$ is defined as

$$rf_i^x = DOC_i^x \cdot svt_i^x$$

where $DOC_i^x (0 \leq DOC_i^x \leq 1)$ is the normalized complexity of the $i^{th}$ component in the scenario $S_x$, and $svt_i^x (0 \leq svt_i^x < 1)$ is the severity level for the $i^{th}$ component in the scenario $S_x$.

The risk factor $rf_{ij}^x$ for a connector between components $i$ and $j$ in the scenario $S_x$ is given by
\[ rf_{ij}^x = EOC_{ij}^x \cdot svt_{ij}^x \]  

(2)

where \( EOC_{ij}^x (0 \leq EOC_{ij}^x \leq 1) \) is the normalized coupling for the connector between \( i^{th} \) and \( j^{th} \) components in the scenario \( S_x \), and \( svt_{ij}^x (0 \leq svt_{ij}^x < 1) \) is the severity level for the connector between the \( i^{th} \) and the \( j^{th} \) components in the scenario \( S_x \).

Next we describe the process of estimating the normalized component complexity \( DOC_i^x \), normalized connector coupling \( EOC_{ij}^x \), and severity levels for the components \( svt_i^x \) and connectors \( svt_{ij}^x \).

### 3.2.1 Dynamic Specifications Metrics using UML

To develop risk mitigation strategies and improve software quality, we should be able to estimate the fault proneness of software components and connectors in the early design phase of the software life cycle. It is well known that there is a correlation between the number of faults found in a software component and its complexity [31]. In this study we compute the dynamic complexity of state charts as a dynamic metric for components [16]. Coupling between components provides important information for identifying possible sources of exporting errors, identifying tightly coupled components, and testing interactions between components. Therefore, we compute dynamic coupling between components as a dynamic metric related to the fault proneness for connectors [16].

**i) Normalized dynamic complexity of a component**

In 1976 McCabe introduced cyclomatic complexity as a measure of program complexity [30]. It is obtained from the control flow graph and defined as \( CC = e - n + 2 \), where \( e \) is number of edges and \( n \) is number of nodes. We use a measure of component complexity similar to McCabe’s cyclomatic complexity.
complexity. However, in contrast to McCabe’s cyclometric complexity which is based on the control flow graph of the source code, our metric for component’s dynamic complexity is based on the UML state charts that are available during early stages of the software life cycle. The state chart of each component $i$ has a number of states and transition between these states that describe the dynamic behavior of the component. For each scenario $S$, a subset of all states of component $i$ are visited and a subset of all transitions is traversed. Let denote with $C_i^x$ the subset of states for a component $i$ visited in the scenario $S$, and with $T_i^x$ the subset of transitions traversed in the state chart of component $i$ in the scenario $S$. The subset of states $C_i^x$ and the corresponding transitions $T_i^x$ are mapped into a control flow graph. The number of nodes in this graph is $c_i^x = |C_i^x|$ which is the cardinality of $C_i^x$. Similarly, the number of edges in this graph is $t_i^x = |T_i^x|$ which is the cardinality of $T_i^x$. It follows that the dynamic complexity $doc_i^x$ of component $i$ in scenario $S$ is defined as

$$doc_i^x = t_i^x - c_i^x + 2.$$  \hfill (3)

The normalized dynamic complexity $DOC_i^x$ of a component $i$ in scenario $S$ is obtained by normalizing the dynamic complexity $doc_i^x$ with respect to the sum of complexities for all active components in scenario $S$

$$DOC_i^x = \frac{doc_i^x}{\sum_{k \in S} doc_k^x}. \hfill (4)$$

As an illustration, the control flow graph of the CD component in the programming scenario (see Figure 3) is presented in Figure 6. The dynamic complexity of this graph is evaluated using equation (3) and normalized with respect to the sum of complexities of all active components in this scenario.
(RS, CD, and CG) using equation (4). Table 2 and Table 3 show the normalized dynamic complexity for all components active in the programming scenario and AVI scenario respectively.

![Diagram of states and transitions](image)

Figure 6. The subset of states and transitions of the CD component in the programming scenario

Table 2. Normalized dynamic complexity of all components in the programming scenario

<table>
<thead>
<tr>
<th>Component</th>
<th>$DOC_i^x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>0.5</td>
</tr>
<tr>
<td>RS</td>
<td>0.2</td>
</tr>
<tr>
<td>CG</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Table 3. Normalized dynamic complexity of all components in the AVI scenario

<table>
<thead>
<tr>
<th>Component</th>
<th>$DOC_i^x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG</td>
<td>0.00017</td>
</tr>
<tr>
<td>AR</td>
<td>0.60135</td>
</tr>
<tr>
<td>VT</td>
<td>0.34837</td>
</tr>
</tbody>
</table>
ii) Normalized dynamic coupling of a connector

We use the matrix representation for coupling where rows and columns are indexed by components and the off-diagonal matrix cells represent coupling between the two components of the corresponding row and column [16]. The row index indicates the sending component, while the column index indicates the receiving component. For example, the cell with row=RS and column=CD is the export coupling value from RS to CD. On the other side, the cell with row=CD and column=RS is the export coupling value from CD to RS. Dynamic coupling metrics are calculated for active connectors during execution of a specific scenario. We compute these metrics directly from the UML sequence diagrams by applying the same set of formulas given in [43].

Let denote with $MT^x_{ij}$ the set of messages sent from component $i$ to component $j$ during the execution of scenario $S_x$ and with $MT^x$ the set of all messages exchanged between all components active during the execution of scenario $S_x$. Then, we define the export coupling $EOC^x_{ij}$ from component $i$ to component $j$ in scenario $S_x$ as a ratio of the number of messages sent from $i$ to $j$ and the total number of messages exchanged in the scenario $S_x$

$$EOC^x_{ij} = \frac{|MT^x_{ij}|}{\left|MT^x\right|}, \quad i, j \in S_x, i \neq j.$$  \hspace{1cm} (5)

The values of dynamic coupling of the connectors estimated using equation (5) for the sequence diagrams of the programming scenario and AVI scenario are given in Table 4 and Table 5 respectively.
Table 4. Dynamic coupling of connectors in the programming scenario

<table>
<thead>
<tr>
<th></th>
<th>RS</th>
<th>CD</th>
<th>CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>0</td>
<td>0.125</td>
<td>0.125</td>
</tr>
<tr>
<td>CD</td>
<td>0</td>
<td>0</td>
<td>0.375</td>
</tr>
<tr>
<td>CG</td>
<td>0</td>
<td>0.375</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5. Dynamic coupling of connectors in the AVI scenario

<table>
<thead>
<tr>
<th></th>
<th>CG</th>
<th>AR</th>
<th>VT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG</td>
<td>0</td>
<td>0.00039</td>
<td>0.00039</td>
</tr>
<tr>
<td>AR</td>
<td>0</td>
<td>0</td>
<td>0.097</td>
</tr>
<tr>
<td>VT</td>
<td>0</td>
<td>0.9</td>
<td>0</td>
</tr>
</tbody>
</table>

3.2.2 Severity Analysis

In addition to the estimates of the fault proneness of each component and connector based on the dynamic complexity and dynamic coupling, for the assessment of components and connectors risk factors we need to consider the severity of the consequences of potential failures. For example, a component may have low complexity, but its failure may lead to catastrophic consequences. Therefore, our methodology takes into consideration the severity associated with each component and connector based on how their failures affect the system operation. Domain experts play a major role in ranking the severity levels. Experts estimate the severity of the components and connectors based on their experience with other systems in the same field. Domain experts can rank severity in more than one way and for more than one purpose [3]. According to MIL_STD_1629A, severity considers the worst case consequence of a failure determined by the degree of injury, property damage, system damage, and mission loss that could ultimately occur. Based on hazard analysis [39] we identify the following severity classes:

- **Catastrophic**: A failure may cause death or total system loss.
- **Critical**: A failure may cause severe injury, major property damage, major system damage, or major loss of production.

- **Marginal**: A failure may cause minor injury, minor property damage, minor system damage, or delay or minor loss of production.

- **Minor**: A failure is not serious enough to cause injury, property damage, or system damage, but will result in unscheduled maintenance or repair.

We assign severity indices of 0.25, 0.50, 0.75, and 0.95 to minor, marginal, critical, and catastrophic severity classes respectively. The selection of values for the severity classes on a linear scale is based on the study conducted by Ammar *et al.* [44]. However, other values could be assigned to severity classes, such as for example using the exponential scale. Table 6 and Table 7 present results from assessing the severity of components and connectors for the AVI scenario.

<table>
<thead>
<tr>
<th>Triggered hazard</th>
<th>Cause of hazard</th>
<th>Accident</th>
<th>Criticality</th>
</tr>
</thead>
<tbody>
<tr>
<td>A fault in processing command routine</td>
<td>Component CG misinterpreting a VVT command for VVI</td>
<td>Heart is continuously triggered but device is still monitored by physician, need immediate fix or disable.</td>
<td>Marginal</td>
</tr>
<tr>
<td>Sensor error. Pacing hardware device malfunctioning</td>
<td>Component AR failed to sense heart in AAI mode Failed to pace the heart.</td>
<td>Heart is always paced while patient condition requires only pacing the heart when no pulse is detected. Heart operation is irregular because it receives no pacing.</td>
<td>Catastrophic</td>
</tr>
<tr>
<td>Timer not set correctly</td>
<td>Component VT refract timer does not generate a timeout</td>
<td>VT is in refractor state, no pace is generated for the heart, patient could die</td>
<td>Catastrophic</td>
</tr>
<tr>
<td>Triggered hazard</td>
<td>Cause of hazard</td>
<td>Accident</td>
<td>Criticality</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Incorrect interpretation of program bytes</td>
<td>Connector CG-AR sends incorrect command (e.g. ToOff instead of ToIdle) message received in error.</td>
<td>Incorrect operation mode and incorrect rate of pacing the heart. Device is still monitored by the physician, immediate maintenance or disable is required.</td>
<td>Marginal</td>
</tr>
<tr>
<td>Incorrect interpretation of program bytes</td>
<td>Connector CG-VT sends incorrect command (e.g. ToOff instead of ToIdle) Message received in error.</td>
<td>Incorrect operation mode and incorrect rate of pacing the heart. Device is still monitored by the physician, immediate maintenance or disable is required.</td>
<td>Marginal</td>
</tr>
<tr>
<td>Timing mismatches between AR and VT operation.</td>
<td>Connector AR-VT, AR continue refractoring in AVI mode, messages do not stop.</td>
<td>Failure to pace the heart.</td>
<td>Catastrophic</td>
</tr>
<tr>
<td>Timing mismatches between AR and VT operation.</td>
<td>Connector VT-AR, VT failed to inform AR of finishing refractoring in AVI mode, messages do not receive</td>
<td>Failure to pace the heart.</td>
<td>Catastrophic</td>
</tr>
</tbody>
</table>

### 3.3. Scenarios Risk Factors

We use an analytical modeling approach to derive the risk factor of each scenario. For this purpose we generalize the state-based modeling approach previously used for architecture-based software reliability estimation [13]. Thus, the software reliability model first published in [6] considers only component failures. In the scenario risk model we account for both component and connector failures, that is, consider both component and connector risk factors. In addition, instead of a single failure state for the scenario, we consider multiple failure states that represent failure modes with different severity. This approach allows us to derive not only the overall scenario risk factor, but also its distribution over different severity classes which provides additional insights important for risk analysis. For example,
the two scenarios may have close values of scenarios risk factors with significantly different distributions among severity classes. Then, it can be inferred that the scenario with a risk factor distributed among more severe failure classes (e.g., critical and catastrophic) deserves more attention than the other scenario.

The scenario risk model is developed in two steps. First, a control flow graph that describes software execution behavior with respect to the manner in which different components interact is constructed using the UML sequence diagrams. It is assumed that a control flow graph has a single entry \( S \) and a single exit node \( T \) representing the beginning and the termination of the execution, respectively. Note that this is not a fundamental requirement. The model can easily be extended to cover multi-entry, multi-exit graphs.

The states in the control flow graph represent active components, while the arcs represent the transfer of control between components (i.e. connectors). It is further assumed that the transfer of control between components has a Markov property which means that given the knowledge of the component in control at any given time, the future behavior of the system is conditionally independent of the past behavior. This assumption allows us to model software execution behavior for scenario \( S_x \) with an absorbing discrete time Markov chain (DTMC) with a transition probability matrix \( P^x = [p_{ij}^x] \), where \( p_{ij}^x \) is interpreted as the conditional probability that the program will next execute component \( j \), given that it has just completed the execution of the component \( i \). The transition probability from component \( i \) to component \( j \) in scenario \( S_x \) is estimated as \( p_{ij}^x = \frac{n_{ij}^x}{n_i^x} \), where \( n_{ij}^x \) is the number of times messages are transmitted from component \( i \) to component \( j \) and \( n_i^x = \sum_j n_{ij}^x \) is the total number of
massages from component $i$ to all other components that are active in the sequence diagram of the scenario $S_v$.

Analyzing the sequence diagram of the AVI scenario given in Figure 4, we construct the DTMC that represents the software execution behavior as shown in Figure 7. Transition probability matrix for this DTMC is given by:

$$
\begin{bmatrix}
S & CG & AR & VT & T \\
S & 0 & 1 & 0 & 0 & 0 \\
CG & 0 & 0 & 0.5 & 0.5 & 0 \\
AR & 0 & 0 & 0 & 1 & 0 \\
VT & 0 & 0 & 0.5 & 0 & 0.5 \\
T & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
$$

![Figure 7. DTMC of the software execution behavior for the AVI scenario](image)

The second step of building the scenario risk model is to consider the risk factors of the components and connectors. Failure can happen during the execution period of any component or during the control transfer between two components. It is assumed that the components and connectors
fail independently. Note that this assumption can be relaxed by considering higher order Markov chain [13].

In architecture-based software reliability models [6], [13] a single state $F$ is added representing the occurrence of a failure. Because the severity of failures plays an important role in the risk analysis, in this work we add $m$ failure states that represent failure modes with different severity. In particular, since for the pacemaker case study we consider four severity classes for each failure (see Table 6 and Table 7) we add four failure states to the DTMC: $F_{\text{minor}}$, $F_{\text{marginal}}$, $F_{\text{critical}}$, and $F_{\text{catastrophic}}$. The transformed Markov chain, which represents the risk model of a given scenario has $(n+1)$ transient states ($n$ components and a starting state $S$) and $(m+1)$ absorbing states ($m$ failure states for each severity class and a terminating state $T$).

Next, we modify the transition probability matrix $P^x$ to $\overline{P^x}$ as follows. The original transition probability $p_{ij}^x$ between components $i$ and $j$ is modified into $(1-rf_i^x) \cdot p_{ij}^x \cdot (1-rf_{ij}^x)$ which represents the probability that the component $i$ does not fail, the control is transferred to component $j$, and the connector between component $i$ and $j$ does not fail. The failure of component $i$ is considered by creating an arc to the failure state associated with a given severity with transition probability $rf_i^x$. Similarly, the failure of a connector between the components $i$ and $j$ is considered by creating an arc to failure state associated with a given severity with transition probability $(1-rf_i^x) \cdot p_{ij}^x \cdot r_{ij}^x$. The transition probability matrix of the transformed DTMC, $\overline{P^x}$, is then partitioned so that

$$\overline{P^x} = \begin{bmatrix} Q^x & C^x \\ 0 & I \end{bmatrix}$$  \hspace{1cm} (6)
where $Q^X$ is an $(n+1)$ by $(n+1)$ sub-stochastic matrix (with at least one row sum less than 1) describing the probabilities of transition only among transient states, $I$ is an $(m+1)$ by $(m+1)$ identity matrix and $C^X$ is a rectangular matrix that is $(n+1)$ by $(m+1)$ describing the probabilities of transition from transient to absorbing states. We define the matrix $A^X = [a_{ik}^X]$ so that $a_{ik}^X$ denotes the probability that the DTMC starting with a transient state $i$ eventually gets absorbed in an absorbing state $k$. Then it can be shown that [40]

$$A^X = (I - Q^X)^{-1} C^X.$$  \hfill (7)

Since in our case we assume a single starting state $S$, the first row of matrix $A^X$ gives us the probabilities that DTMC is absorbed in absorbing states $T$, $F_{\text{minor}}$, $F_{\text{marginal}}$, $F_{\text{critical}}$, and $F_{\text{catastrophic}}$. In particular, $a_{11}^X$ is equal to $(1 - r_f^X)$, where $r_f^X$ is the scenario risk, while $a_{12}^X$, $a_{13}^X$, $a_{14}^X$, and $a_{15}^X$ give us the distribution of the scenario risk factor among minor, marginal, critical, and catastrophic severity classes respectively.

Next, we illustrate the construction of the scenario risk model and its solution on the AVI scenario. DTMC of the software execution behavior given in Figure 7 is transformed to the DTMC presented in Figure 8, which represents the risk model of the AVI scenario.
Figure 8. Risk model of the AVI scenario

The transition probability matrix of the transformed DTMC is given by:

\[
P^{AVI} = \begin{bmatrix}
S & CG & AR & VT & T & F_{\text{minor}} & F_{\text{marginal}} & F_{\text{critical}} & F_{\text{catastrophic}} \\
S & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
CG & 0 & 0 & 0.4998 & 0.4998 & 0 & 0 & 0.0004 & 0 \\
AR & 0 & 0 & 0 & 0.3619 & 0 & 0 & 0 & 0.6381 \\
VT & 0 & 0 & 0.0472 & 0 & 0.3258 & 0 & 0 & 0 & 0.6270 \\
T & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
F_{\text{minor}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
F_{\text{marginal}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
F_{\text{critical}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
F_{\text{catastrophic}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

It is clear that: \( Q^{AVI} = \begin{bmatrix}
S & CG & AR & VT \\
S & 0 & 1 & 0 & 0 \\
CG & 0 & 0 & 0.4998 & 0.4998 \\
AR & 0 & 0 & 0 & 0.3619 \\
VT & 0 & 0 & 0.0472 & 0 \\
\end{bmatrix}\) and
The matrix $A^{AVI}$ is computed as:

$$A^{AVI} = (I - Q^{AVI})^{-1} C^{AVI}$$

Thus, the risk factor of the AVI scenario is equal to $1-0.2256=0.7744$. This risk factor is distributed among marginal and catastrophic severity classes (0.0004 and 0.7740 respectively).

We developed scenario risk models for all scenarios of the pacemaker example (programming, AVI, AAI, VVI, AAT, and VVT). Table 8 shows how the risk factor of each scenario is distributed among the severity classes, as well as the overall scenario risk factors. Figure 9 presents graphically the information given in Table 8. The bar’s shade represents the severity class and the z-axis represents the value of the risk factor for a given severity class.

Table 8. Distribution of the scenarios risk factors among severity classes

<table>
<thead>
<tr>
<th>Scenario risk factors</th>
<th>Programming</th>
<th>AVI</th>
<th>AAI</th>
<th>VVI</th>
<th>AAT</th>
<th>VVT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minor</td>
<td>0.3169</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Marginal</td>
<td>0.1782</td>
<td>0.0004</td>
<td>0.5002</td>
<td>0.5002</td>
<td>0.5001</td>
<td>0.5001</td>
</tr>
<tr>
<td>Critical</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Catastrophic</td>
<td>0</td>
<td>0.7740</td>
<td>0.4743</td>
<td>0.4743</td>
<td>0.4747</td>
<td>0.4747</td>
</tr>
<tr>
<td>Scenario risk factors</td>
<td>0.4951</td>
<td>0.7744</td>
<td>0.9745</td>
<td>0.9745</td>
<td>0.9748</td>
<td>0.9748</td>
</tr>
</tbody>
</table>
Several observations are made from Table 8 and Figure 9. First, all scenarios from the operational mode have higher risk factors than the programming scenario which is just used to set the mode of the pacemaker. Next, it is obvious that the knowledge of the distribution of scenarios risk factors among severity classes provides valuable information for the risk analysts in addition to the overall scenario risk factor. Thus, the AVI scenario has the smallest scenario risk factor (0.7744) among the operational scenarios (AVI, AAI, VVI, AAT, and VVT). However, most of the AVI scenario risk factor belongs to the catastrophic severity class (0.7740), that is, AVI scenario has the highest value of the risk factor in the catastrophic severity class. The risk factors of the other operational scenarios are distributed almost equally among the marginal and catastrophic severity classes with the values in catastrophic class significantly smaller than for the AVI scenario. Programming scenario has the smallest overall scenario risk factor (0.4951) distributed only among minor and marginal severity classes, which means that it is the less critical scenario in the pacemaker case study.
3.4. Use Cases and Overall System Risk Factors

The risk factor $rf_k$ of each use case $U_k$ is obtained by averaging the risk factors of all scenarios $S_x$ that are defined for that use case

$$rf_k = \sum_{\forall S_x \in U_k} rf^x \cdot p^x_k$$

(8)

where $rf^x$ is the risk factor of scenario $S_x$ in use case $U_k$ and $p^x_k$ is the probability of occurrence of scenario $S_x$ in the use case $U_k$. Since in the pacemaker example we considered one scenario per use case, the use case risk factors are identical to the scenarios risk factors.

Similarly, the overall system risk factor is obtained by averaging the use case risk factors

$$rf = \sum_{\forall U_k} rf_k \cdot p_k$$

(9)

where $rf_k$ and $p_k$ are the risk factor and probability of occurrence of the use case $U_k$.

It is obvious from equations (8) and (9) that the use cases and overall system risk factors depend on the probabilities of scenarios occurrence $p^x_k$ in the use case $U_k$ and the probability of use case occurrence $p_k$. Hence, scenarios (use cases) with high risk factors but very low probability of occurrence will not contribute significantly to the overall system risk factor.

Using the equations (8) and (9) and the use case probabilities shown in Table 1, we estimate the overall risk factor of the pacemaker $0.9118$. The distribution of the overall system risk factor among severity classes is presented in Table 9 and Figure 10. We see that the system risk factor is mostly distributed among marginal and catastrophic severity class. Even more, the catastrophic severity class is the dominant class for this system.
Table 9. Distribution of the overall risk factor over severity classes

<table>
<thead>
<tr>
<th>Overall system risk factor</th>
<th>Minor</th>
<th>Marginal</th>
<th>Critical</th>
<th>Catastrophic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.0032</td>
<td>0.3520</td>
<td>0</td>
<td>0.5566</td>
</tr>
</tbody>
</table>

3.5. Sensitivity Analysis

In the proposed methodology we use an analytical approach and derive close form solutions. One of the advantages of this approach is that sensitivity analysis can be performed simply by plugging different values of the parameters in the close form solutions, which is faster and more effective than reapplying the algorithmic solution for each set of different parameters as in [42]. Next, we illustrate the sensitivity of the scenarios and overall system risk factors to components/connectors risk factors.

Figure 11 illustrates the variation of the risk factor of the AVI scenario as a function of changes in risk factors of components active in that scenario. The variation of the risk factor of VT component introduces the biggest variation of the AVI scenario risk factor (from 0.65 to 1). This is the case because the VT component is the most active component in this scenario that senses the heart pulse.
On the other side, the variations of the risk factor of the AR and CG components affect less the range of the variation the AVI scenario risk factor. However, the AR component is also critical because it results in the smaller value of the scenario’s risk factor. Figure 12 shows the sensitivity of the risk factor of the programming scenario to the risk factors of the components active in that scenario. In this case the variation of the risk factor of the CG component introduces the biggest variation of the programming scenario risk factor (from 0.175 to 0.979). The variation of the overall system risk factor as a function of components risk factors is presented in Figure 13. It is clear that the risk factors of components CG, VT, and AR are more likely to affect the overall system risk. This is due to the fact that these components are active in scenarios that have high execution probabilities. Further, the variation of the risk factors of components that are active only in the programming scenario (i.e. RS and CD) has almost no influence on the variation of the overall system risk factor because the execution probability of the programming scenario is one order of magnitude lower than the execution probabilities of other scenarios.

Figure 14 and Figure 15 show the variation of the AVI scenario risk factors and overall system risk factor as a function of connectors risk factors. It is obvious that both AVI scenario risk factor and the overall system risk factor are the most sensitive to the risk factor of the CG-VT connector.
Figure 11. Sensitivity of the AVI scenario risk factor to the risk factors of the components

Figure 12. Sensitivity of the programming scenario risk factor to the risk factors of the components

Figure 13. Sensitivity of the overall system risk factor to the risk factors of the components

Figure 14. Sensitivity of the AVI scenario risk factor to the risk factors of the connectors

Figure 15. Sensitivity of the overall system risk factor to the risk factors of the connectors
3.6. Identifying critical components

Identifying the critical components in the system under assessment is very helpful in the development process of that system. The set of most risky components in the system should undergo more rigorous development and should be allocated more testing effort. A beneficial outcome of our risk assessment methodology is the ability to identify a set of most critical components. Figure 16 presents risk factors of all components for different scenarios of the pacemaker case study. In this figure, the different severity levels are presented by different shades. It is obvious that VT and AR are the most critical components in the pacemaker case study because they have high risk factors with catastrophic severity in multiple scenarios. Similar approach can be used to identify the set of most critical connectors.

![Graph showing component risk factors for different scenarios](image)

Figure 16. Identification of the critical components for the pacemaker

4. Related Work

In this paper, we present a methodology for risk assessment that is based on the UML behavior specifications. In the sequel we summarize research work related to our work.
A large number of object-oriented measures have been proposed in the literature ([2], [5], [7], [8], [18], [24], [26], [27], [28], [37]). Particular emphasis has been given to the measurement of design artifacts in order to help quality assessment early in the development process.

Recent evidence suggests that most faults are found in only a few of a system’s components [12], [19]. If these few components can be identified early, then mitigating actions can be taken, such as for example focus the testing on high-risk components by optimally allocating testing resources [15], or redesigning components that are likely to cause failures or be costly to maintain.

Predictive models exist that incorporate a relationship between program errors measures and software complexity metrics [20]. Software complexity measures were also used for developing and executing test suites [17]. Therefore, static complexity is used to assess the quality of a software product. The level of exposure of a module is a function of its execution environment. Hence, dynamic complexity [21] evolved as a measure of complexity of the subset of code that is actually executed. Dynamic complexity used for reliability assessment purposes was discussed in [31].

Early identification of faulty components is commonly achieved through a binary quality model that classifies components into either a faulty or non-faulty category [10], [11], [23], [25]. Also, studies exist that predict a number of faults in individual components [22]. These estimates can be used for ranking the components.

Ammar et.al. extended dynamic complexity definitions to incorporate concurrency complexity [1]. Further, they used Coloured Petri Nets models to measure dynamic complexity of software systems using simulation reports. Yacoub et.al. defined dynamic metrics that include dynamic complexity and dynamic coupling to measure the quality of architectures [43]. Their approach was based on dynamic execution of UML state chart specification of a component and the proposed metrics were based on simulation reports. Yacoub et.al. in [42] combined severity and complexity factors to develop heuristic
risk factors for the components and connectors. Based on scenarios, they developed component dependency graph that represents components, connectors, and probabilities of component interactions. The overall system risk factor as a function of the risk factors of its constituting components and connectors was obtained using the aggregation algorithm.

5. Conclusion and Future Work

In this paper we propose a methodology for risk assessment based on the UML specifications such as use cases and sequence diagrams that can be used in the early phases the software life cycle. Building on the previous research work on risk assessment and architecture – based software reliability, we developed a new and comprehensive methodology that provides (1) accurate and more efficient methods to estimate risk factors on different levels and (2) additional information useful for risk analysis.

Thus, the risk assessment in this paper is entirely based on the analytical methods. First, we estimate components and connectors dynamic risk factors analytically based on the information from UML sequence diagrams. Then, we construct a Markov model for estimation of the each scenario risk factor and derive closed form exact solutions for the scenarios, use cases, and overall system risk factors. The fact that the risk assessment is entirely based on the analytical methods enables more effective risk assessment and sensitivity analysis, as well as a straightforward development of a tool for automatic risk assessment.

Some of the useful insights that can obtain from the proposed methodology include the following. In addition to overall risk factor, we estimate scenarios and use cases risk factors which enable us to focus on the high-risk scenarios and uses cases even though they may be rarely used and therefore not contributing significantly to the overall system risk factor. Next, we estimate the
distribution of the scenarios/use cases/system risk factors over different severity classes which allow us to make a list of critical scenarios in each use case, as well as a list of critical use cases in the system. Finally, we identify a list of critical components and connectors that has high risk values in high severity classes.

Our future work is focused on generalization of the methodology presented in this paper. Thus, we are considering different kinds of dependencies that might be present in the UML use case diagrams and the way to derive their risk factors. Another direction of our future research is the development of performance based risk assessment methodology.

6. Acknowledgements

This work is funded in part by grant 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.
7. References


