Architectural-Level Risk Analysis Using UML

Katerina Goseva-Popstojanova, *Senior Member*, *IEEE*,
Ahmed Hassan, *Student Member*, *IEEE*, Ajith Guedem, Walid Abdelmoez, *Student Member*, *IEEE*,
Diaa Eldin M. Nassar, *Student Member*, *IEEE*, Hany Ammar, *Member*, *IEEE*, and
Ali Mili, *Member*, *IEEE*

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 identify the high-risk components and connectors of the product architecture, 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. 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, severity of failure, 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 the NASA-STD-8719.13A standard [27], 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 [24]: 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 in a scenario in which a failure will be triggered. 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 a major impact on the fault proneness [9]. 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 in a given scenario based on the premise that active components and connectors are sources of failures [32]. 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 is significantly less costly than detection and correction 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)

- *K. Goseva-Popstojanova, A. Hassan, A. Guedem, W. Abdelmoez, D. Nassar, and H. Ammar are with the Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506-6109. E-mail: {katerina, hassan, guedem, rabie, dmnassar, ammar}@csee.wvu.edu.*
- *A. Mili is with the College of Computing Science, New Jersey Institute of Technology, Newark, NJ 07102. E-mail: mili@cis.njit.edu.*

Manuscript received 30 Dec. 2002; revised 26 May 2003; accepted 23 June 2003.
Recommended for acceptance by J.B. Dugan.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 118057.

technique. We combine severity and complexity (coupling) metrics to obtain heuristic risk factors for the components (connectors). Then, we develop a 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 a 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], [31].

This work is also motivated by the need to compute risk factors during the early phases of the software life cycle based on UML specifications such as use cases and scenarios. 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 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:

1. We present a lightweight methodology to perform analytical risk assessment at the architectural level based on the analysis of behavioral UML specifications, mainly use cases and sequence diagrams. This risk assessment approach is entirely analytical, in contrast with our previous work [1], [31], which was based on simulations of execution profiles.
2. We introduce the notions of scenario/use case risk factors that 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 exercised rarely. Although these scenarios and/or use cases will not contribute significantly to the overall system risk factor as computed in [31], their risk analysis is extremely important due to the fact that they usually provide exception handling of rare but critical conditions.
3. We develop a Markov model to determine scenarios risk factors using components and connector risk factors. This model provides exact closed form solutions for the scenarios risk factors, while the algorithm for traversal of the component dependency graphs used in [31] provides an approximate solution. An additional advantage of the derived closed form solutions for the scenarios risk factors is a more effective way for conducting sensitivity analysis. Thus, we simply plug different values of the parameters in the closed form solutions, while, in [31], the 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.
4. The Markov model used for estimating the scenarios risk factors generalizes the existing architecture-based software reliability models in two ways. Thus, while the software reliability model presented in [5] considers only component failures, in the scenarios risk models, we account for both components and connectors failures, that is, we consider both components and connectors risk factors. Further, instead of a single failure state considered in all existing architecture-based software reliability models [12], 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 that are 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.
5. 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 [25] as a front end, we have already developed a prototype of a tool for risk assessment [30] 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 [8] to illustrate how the proposed methodology works. A cardiac pacemaker is an implanted device that assists cardiac functions when the underlying pathologies make the intrinsic heartbeats low. A failure 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.

Fig. 1 shows the components and connectors of the pacemaker in a capsule diagram. It 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.

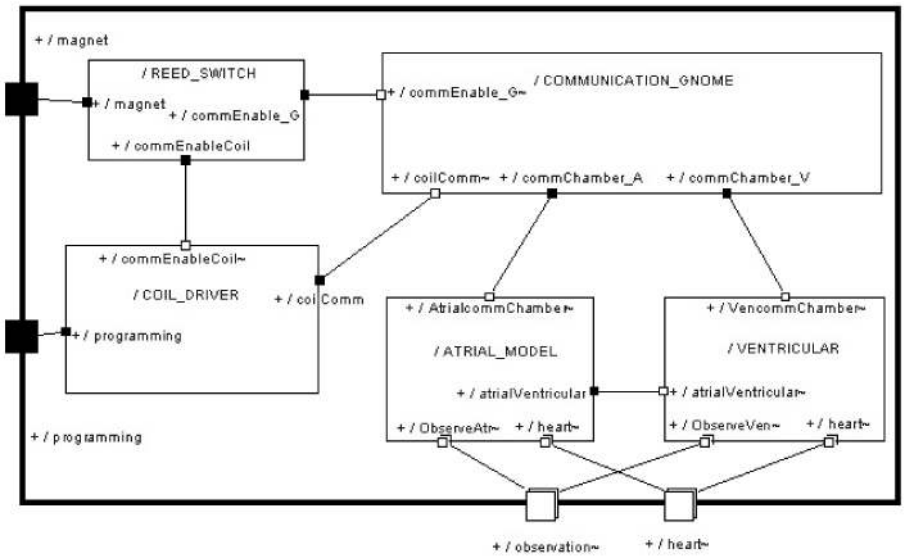

Fig. 1. The architecture of the pacemaker example.

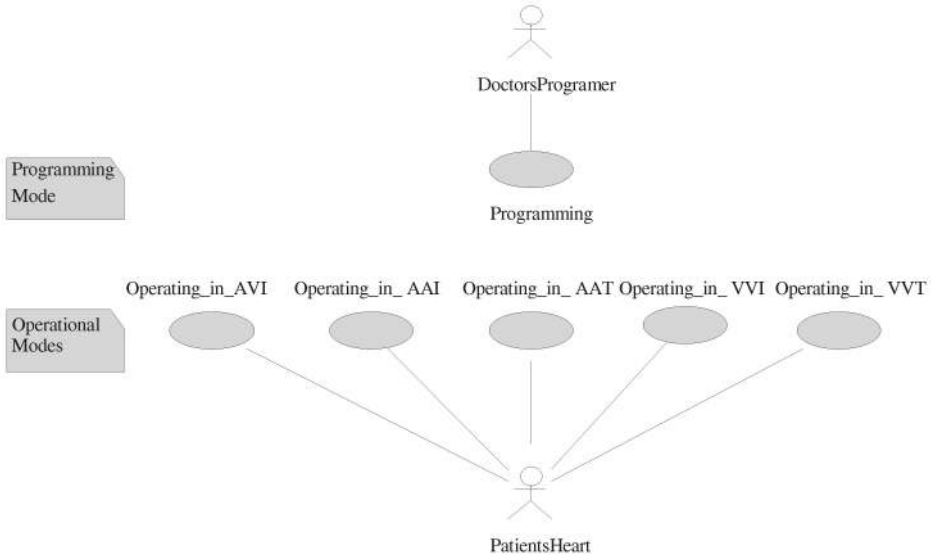

Fig. 2. Use case diagram of the pacemaker.

- *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. The use case diagram of the pacemaker application is given in Fig. 2. It presents the six use cases and the two actors: *doctor programmer* and *patient's*

TABLE 1
Probabilities of the Use Cases' Executions

Use case	*Programming*	*AVI*	*AAI*	*VVI*	*AAT*	*VVT*
Probability	0.01	0.29	0.20	0.20	0.15	0.15

heart. Each use case in Fig. 2 is realized by at least one sequence diagram (i.e., scenario).

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 [21]. For the pacemaker example, according to [8], 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.

Fig. 3 shows a sequence diagram of 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 is 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, as shown in Fig. 4. We assume that the UML logical architectural model consists of a use case diagram defining several independent use cases as shown in Fig. 2 and that each use case is realized with one or more independent scenarios modeled using sequence diagrams as shown in Fig. 3. Sequence diagrams depict how a group of components interact in a use case. Each sequence diagram shows a number of components and messages exchanged between them [26].

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 (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.

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

$$rf_i^x = DOC_i^x \cdot svt_i^x, \tag{1}$$

where DOC_i^x $(0 \leq DIC_i^x \leq 1)$ is the normalized complexity and svt_i^x $(0 \leq svt_i^x < 1)$ is the severity level of the ith component in the scenario S_x.

The risk factor rf_{ij}^x of a connector between components i and j in the scenario S_x is

$$rf_{ij}^x = EOC_{ij}^x \cdot svt_{ij}^x, \tag{2}$$

where EOC_{ij}^x $(0 \leq EOC_{ij}^x \leq 1)$ is the normalized coupling and svt_{ij}^x $(0 \leq svt_{ij}^x < 1)$ is the severity level for the connector between the ith and the jth 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 [23]. In this study, we compute the dynamic complexity of state charts as a dynamic metric for components [14]. 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 [14].

Normalized dynamic complexity of a component. In 1976, McCabe introduced cyclomatic complexity as a measure of program complexity [22]. 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. However, in contrast to McCabe's cyclomatic complexity which is based on the

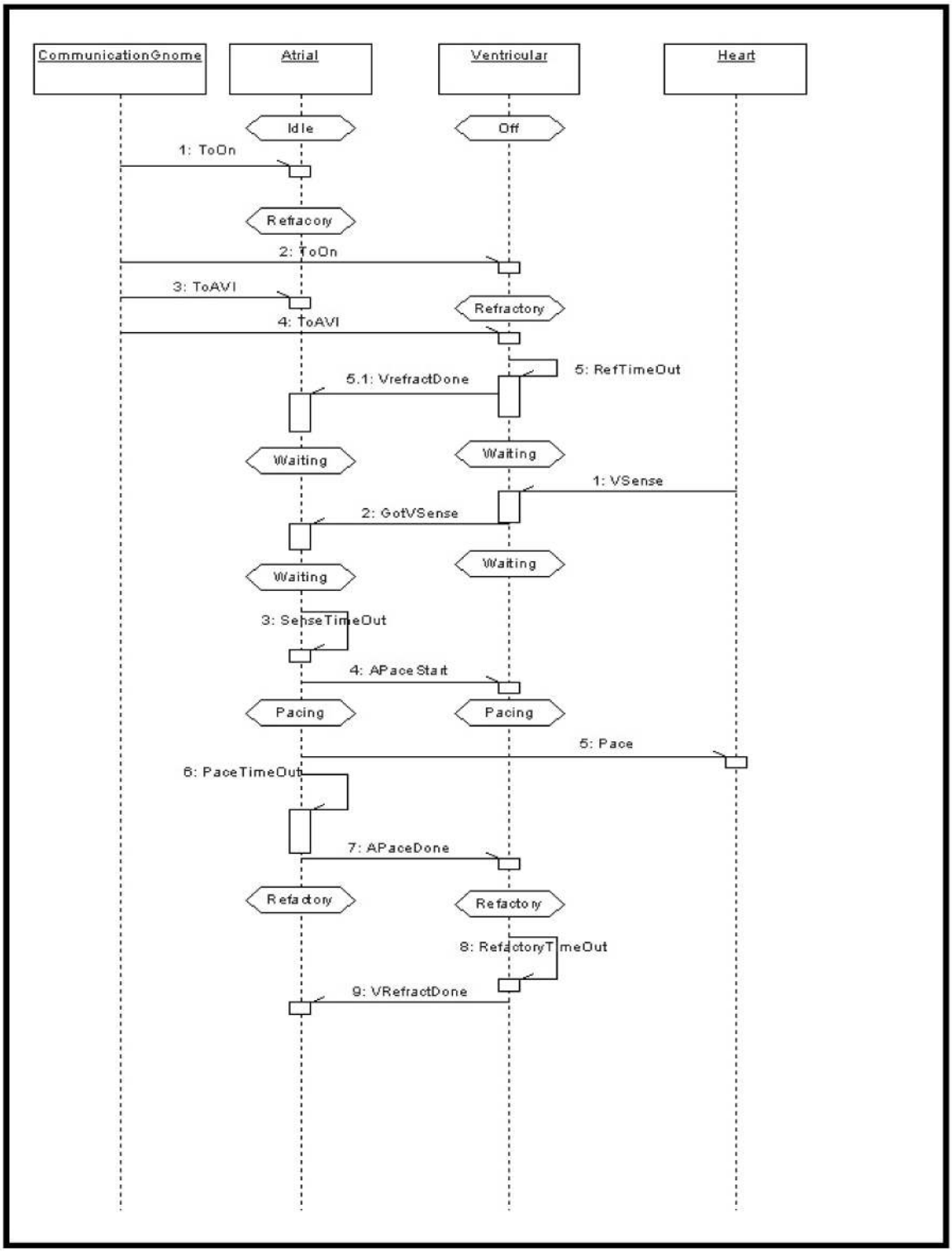

Fig. 3. Sequence diagram of the AVI scenario.

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_x, a subset of all states of component i is visited and a subset of all transitions is traversed. Let C_i^x denote the subset of states for a component i visited in the scenario S_x and T_i^x denote the subset of transitions traversed in the state chart of component i in the scenario S_x. 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. By analogy with McCabe's cyclomatic

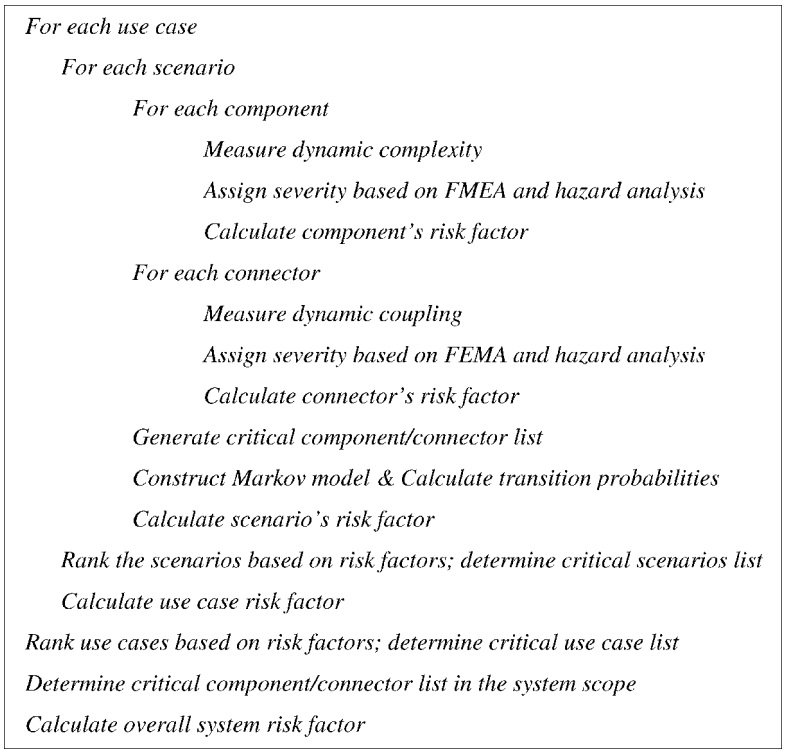

Fig. 4. The risk analysis process.

complexity, we define the dynamic complexity doc_i^x of component i in scenario S_x as

$$doc_i^x = t_i^x - c_i^x + 2. \tag{3}$$

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

$$DOC_i^x = \frac{doc_i^x}{\sum_{k \in S_x} doc_k^x}. \tag{4}$$

As an illustration, the state chart of the CD component in the programming scenario is presented in Fig. 5. The dynamic complexity of this graph is evaluated using (3) and normalized with respect to the sum of complexities of all active components in this scenario (RS, CD, and CG) using (4). Tables 2 and 3 show the normalized dynamic complexity for the components that are active in the programming scenario and AVI scenario, respectively.

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 [14]. 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 [32].

Let MT_{ij}^x denote the set of messages sent from component i to component j during the execution of scenario S_x

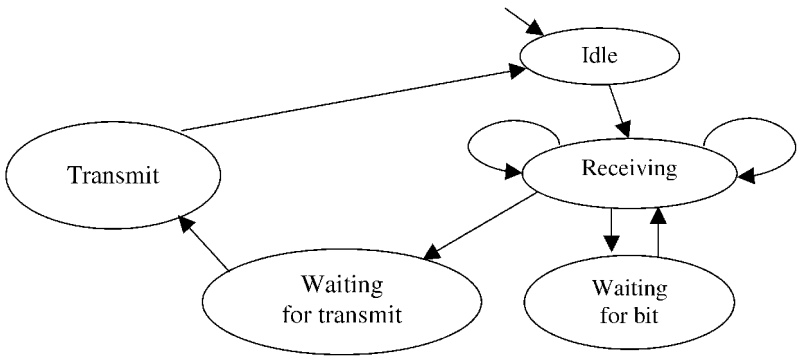

Fig. 5. The state chart of the CD component in the programming scenario.

TABLE 2
Normalized Dynamic Complexity of All Components in the Programming Scenario

Component	DOC_i^x
CD	0.5
RS	0.2
CG	0.3

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

$$EOC_{ij}^x = \frac{\left|MT_{ij}^x\right|_{i,j\in S_x, i\neq j}}{|MT^x|}. \quad (5)$$

The values of dynamic coupling of the connectors estimated using (5) for the sequence diagrams of the programming scenario and AVI scenario are given in Tables 4 and 5, respectively. Note that the DOC and EOC measures linearly depend on the number of states visited by a component, and the number of messages sent over a connector in a given scenario, respectively.

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 the hazard analysis [28], 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, or major system damage.
- *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. [33]. However, other values could be assigned to severity classes, such as for example using the logarithmic scale. Tables 6 and 7 present results from assessing the severity of components and connectors for the AVI scenario.

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 [12]. Thus, the software reliability model first published in [5] considers only component failures. In the scenario risk model, we account for both component and connector failures, that is, we consider both component and connector risk factors. In addition, instead of a single failure state considered in all existing architecture-based software reliability models [12], for each scenario, we consider multiple failure states that represent failure modes with different severity. This approach allows us to derive not only the overall scenario

TABLE 3
Normalized Dynamic Complexity of All Components in the AVI Scenario

Component	DOC_i^x
CG	0.00017
AR	0.60135
VT	0.34837

TABLE 4
Dynamic Coupling of Connectors in the Programming Scenario

	RS	CD	CG
RS	0	0.125	0.125
CD	0	0	0.375
CG	0	0.375	0

TABLE 5
Dynamic Coupling of Connectors in the AVI Scenario

	CG	AR	VT
CG	0	0.00039	0.00039
AR	0	0	0.097
VT	0	0.9	0

TABLE 6
Severity Analysis for Components in the AVI Scenario

Triggered hazard	Cause of hazard	Accident	Criticality
A fault in processing command routine	Component CG misinterpreting the received bytes.	Heart is continuously triggered but device is still monitored by physician, need immediate fix or disable.	Marginal
Sensor error. Pacing hardware device malfunctioning	Component AR stays in the pacing state because it doesn't receive time out message.	Heart is always paced while patient condition requires only pacing the heart when no pulse is detected.	Catastrophic
Timer not set correctly	Component VT refract timer does not generate a timeout.	Component VT is in refractoring state, no pace is generated because it cannot sense the heart. Patient could die.	Catastrophic

risk factor, but also its distribution over different severity classes, which provides additional insights important for risk analysis. Thus, scenarios with risk factors distributed among more severe failure classes (e.g., critical and catastrophic) deserve more attention than the other scenarios.

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 multientry, multiexit 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 $P^x = [p^x_{ij}]$, where p^x_{ij} 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 $p^x_{ij} = n^x_{ij}/n^x_i$, where n^x_{ij} is the number of times messages are transmitted from component i to component j, and $n^x_i = \sum_j n^x_{ij}$ is the total number of messages from component i to all other components that are active in the sequence diagram of the scenario S_x.

TABLE 7
Severity Analysis for Connectors in the AVI Scenario

Triggered hazard	Cause of hazard	Accident	Criticality
Incorrect interpretation of program bytes	Connector CG-AR transmits incorrect command, message received in error by component AR.	Incorrect operation mode and incorrect rate of pacing the heart. Device is still monitored by the physician, immediate maintenance or disabling is required.	Marginal
Incorrect interpretation of program bytes	Connector CG-VT transmits incorrect command. Message received in error by component VT.	Incorrect operation mode. Immediate maintenance or disabling is required.	Marginal
Timing mismatches between AR and VT operation.	Messages transmitted by connector AR-VT do not stop. Component AR stays in refractory state.	Failure to pace the heart.	Catastrophic
Timing mismatches between AR and VT operation.	Connector VT-AR does not transmit the messages. Component VT fails to inform component AR that the heart needs pacing.	Failure to pace the heart.	Catastrophic

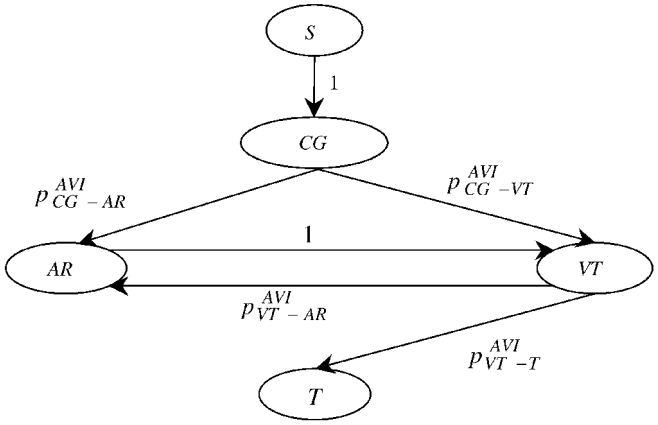

Fig. 6. DTMC of the software execution behavior for the AVI scenario.

Analyzing the sequence diagram of the AVI scenario given in Fig. 3, we construct the DTMC that represents the software execution behavior as shown in Fig. 6. Transition probability matrix for this DTMC is given by:

$$P^x = \begin{array}{c} \\ S \\ CG \\ AR \\ VT \\ T \end{array} \begin{array}{c} \begin{array}{ccccc} S & CG & AR & VT & T \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}.$$

The second step in 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 [12]. In the existing architecture-based software reliability models [5], [12], 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, we add four failure states to the DTMC: F_{minor}, F_{marginal}, F_{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^x_{ij} between components i and j is modified into $(1-rf^x_i)\cdot p^x_{ij}\cdot(1-rf^x_{ij})$, which represents the case when the component i does not fail, the control is transferred to component j, and the connector between components 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^x_i. 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 $(1-rf^x_i)\cdot p^x_{ij}\cdot r^x_{ij}$. The transition probability matrix of the transformed DTMC, $\overline{P^x}$, is then partitioned so that

$$\overline{P^x} = \begin{bmatrix} Q^x & C^x \\ 0 & I \end{bmatrix}, \tag{6}$$

where Q^x is an $(n+1)$ by $(n+1)$ substochastic matrix describing the probabilities of transitions only among transient states, I is an $(m+1)$ by $(m+1)$ identity matrix, and C^x is an $(n+1)$ by $(m+1)$ matrix describing the probabilities of transitions from transient to absorbing states. We define the matrix $A^x=[a^x_{ij}]$ so that a^x_{ij} 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 [29]

$$A^x = (I-Q^x)^{-1}C^x, \tag{7}$$

where I is an $(n+1)$ by $(n+1)$ identity matrix.

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_{minor}, F_{marginal}, F_{critical}, and $F_{\text{catastrophic}}$. In particular, a^x_{11} is equal to $(1-rf^x)$, where rf^x is the scenario risk factor, while a^x_{12}, a^x_{13}, a^x_{14}, and a^x_{15} 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 Fig. 6 is transformed to the DTMC presented in Fig. 7, which represents the risk model of the AVI scenario. The transition probability matrix of the transformed DTMC is given in Fig. 8.

It is clear that:

$$Q^{AVI} = \begin{array}{c} \\ S \\ CG \\ AR \\ VT \end{array} \begin{array}{c} \begin{array}{cccc} S & CG & AR & VT \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0.4998 & 0.4998 \\ 0 & 0 & 0 & 0.3619 \\ 0 & 0 & 0.0472 & 0 \end{pmatrix} \end{array},$$

$$C^{AVI} = \begin{array}{c} \\ S \\ CG \\ AR \\ VT \end{array} \begin{array}{c} \begin{array}{ccccc} T & F_{\text{minor}} & F_{\text{marginal}} & F_{\text{critical}} & F_{\text{catastrophic}} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0004 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.6381 \\ 0.3258 & 0 & 0 & 0 & 0.6270 \end{pmatrix} \end{array}.$$

The matrix A^{AVI} is computed as:

$$A^{AVI} = (I-Q^{AVI})^{-1}C^{AVI} = \begin{array}{c} \\ S \\ CG \\ AR \\ VT \end{array} \begin{array}{c} \begin{array}{ccccc} T & F_{\text{minor}} & F_{\text{marginal}} & F_{\text{critical}} & F_{\text{catastrophic}} \end{array} \\ \begin{pmatrix} 0.2256 & 0 & 0.0004 & 0 & 0.7740 \\ 0.2256 & 0 & 0.0004 & 0 & 0.7740 \\ 0.1200 & 0 & 0 & 0 & 0.8800 \\ 0.3315 & 0 & 0 & 0 & 0.6685 \end{pmatrix} \end{array}.$$

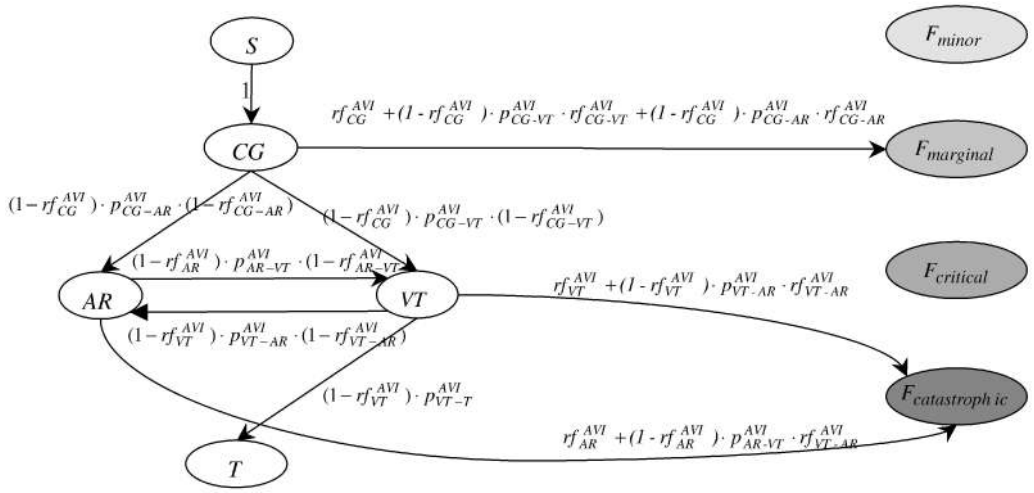

Fig. 7. Risk model of the AVI scenario.

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. Fig. 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.

Several observations are made from Table 8 and Fig. 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). The risk factors of the other operational scenarios are distributed almost equally among the marginal and catastrophic severity classes with the values in the catastrophic class significantly smaller than for the AVI scenario. The programming scenario has the smallest overall scenario risk factor (0.4951) distributed only among minor and marginal severity classes, which means that this 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

$$rf_k = \sum_{\forall S_x \in U_k} rf^x \cdot p_k^x, \tag{8}$$

where rf^x and p_k^x are the risk factor and the probability of occurrence of scenario S_x in the use case U_k, respectively. Since, in the pacemaker example, we considered one scenario per use case, the use case risk factors are identical to the scenario's risk factors.

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

$$rf = \sum_{\forall U_k} rf_k \cdot p_k, \tag{9}$$

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

It is obvious from (8) and (9) that the use cases and overall system risk factors depend on the probabilities of scenarios occurrence p_k^x in the use case U_k and the

$$\overline{P^{AVI}} = \begin{array}{c} \\ S \\ CG \\ AR \\ VT \\ T \\ F_{\text{minor}} \\ F_{\text{marginal}} \\ F_{\text{critical}} \\ F_{\text{catastrophic}} \end{array} \begin{array}{c} \begin{array}{ccccccccc} S & CG & AR & VT & T & F_{\text{minor}} & F_{\text{marginal}} & F_{\text{critical}} & F_{\text{catastrophic}} \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.4998 & 0.4998 & 0 & 0 & 0.0004 & 0 & 0 \\ 0 & 0 & 0 & 0.3619 & 0 & 0 & 0 & 0 & 0.6381 \\ 0 & 0 & 0.0472 & 0 & 0.3258 & 0 & 0 & 0 & 0.6270 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Fig. 8. Transition probability matrix of the transformed DTMC.

TABLE 8
Distribution of the Scenarios Risk Factors among Severity Classes

	Programming	AVI	AAI	VVI	AAT	VVT
Minor	0.3169	0	0	0	0	0
Marginal	0.1782	0.0004	0.5002	0.5002	0.5001	0.5001
Critical	0	0	0	0	0	0
Catastrophic	0	0.7740	0.4743	0.4743	0.4747	0.4747
Scenario risk factor	0.4951	0.7744	0.9745	0.9745	0.9748	0.9748

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 (8) and (9) and the use case probabilities shown in Table 1, we estimate the overall risk factor of the pacemaker to be 0.9118. The distribution of the overall system risk factor among severity classes is presented in Table 9 and Fig. 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.

3.5 Sensitivity Analysis

In the proposed methodology, we use an analytical approach and derive closed 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 closed form solutions, which is faster and more effective than reapplying the algorithmic solutions for each set of different parameters as in [31]. Next, we illustrate the sensitivity of the scenarios and overall system risk factors to components/connectors risk factors.

Fig. 11 illustrates the variation of the risk factor of the AVI scenario as a function of changes in risk factors of the active components in that scenario. The variation of the risk factor of the 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. On the other side, the variations of the risk factor of the AR and CG components have smaller effect on 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. Fig. 12 shows the sensitivity of the risk factor of the programming scenario to the risk factors of the active components 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).

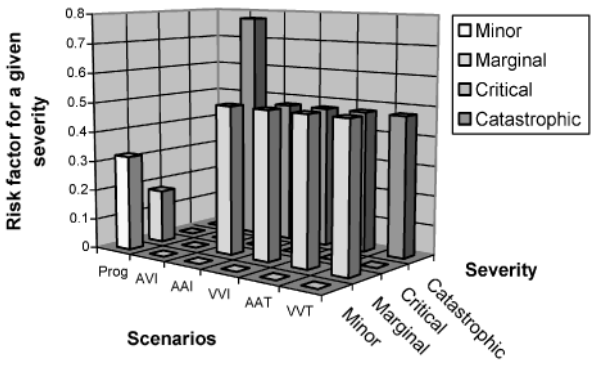

Fig. 9. Distribution of the scenarios risk factors among severity classes.

The variation of the overall system risk factor as a function of components risk factors is presented in Fig. 13. It is clear that the risk factors of components CG, VT, and AR are most 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. In addition, 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 that the execution probabilities of other scenarios.

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

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 the set of the most critical components. Fig. 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. A similar approach can be used to identify the set of most critical connectors.

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], [4], [6], [7], [16]. Particular

TABLE 9
Distribution of the Overall System Risk Factor over Severity Classes

	Minor	Marginal	Critical	Catastrophic
Overall system risk factor	0.0032	0.3520	0	0.5566

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 [11]. If these components can be identified early, then mitigating actions can be taken, such as, for example, focusing the testing on high-risk components by optimally allocating testing resources [13], or redesigning components that are likely to cause failures or to be costly to maintain.

Predictive models exist that incorporate a relationship between program error measures and software complexity metrics [17]. Software complexity measures were also used for developing and executing test suites [15]. Therefore, static complexity is used to assess the quality of a software product. The level of exposure of a component is a function of its execution environment. Hence, dynamic complexity [18] 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 [23]. Early identification of faulty components is commonly achieved through a binary quality model that classifies components into either a faulty or nonfaulty category [9], [10], [20]. Also, studies exist that predict a number of faults in individual components [19]. These estimates can be used for ranking the components.

Ammar et al. extended dynamic complexity definitions to incorporate concurrency complexity [1]. In addition, they used Coloured Petri Nets models to measure dynamic complexity of software systems using simulation reports. Yacoub et al. define dynamic metrics that include dynamic complexity and dynamic coupling to measure the quality of software architectures [32]. 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 and Ammar [31] combine 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 is obtained using the aggregation algorithm.

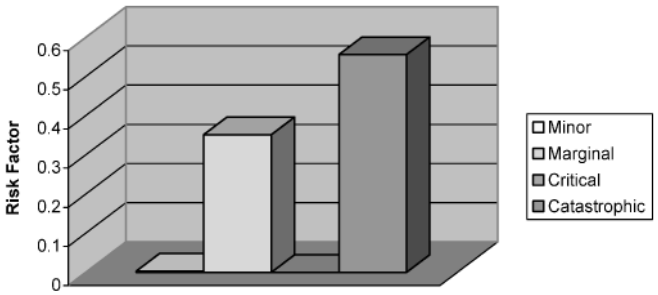

Fig. 10. Distribution of the overall system risk factor over severity classes.

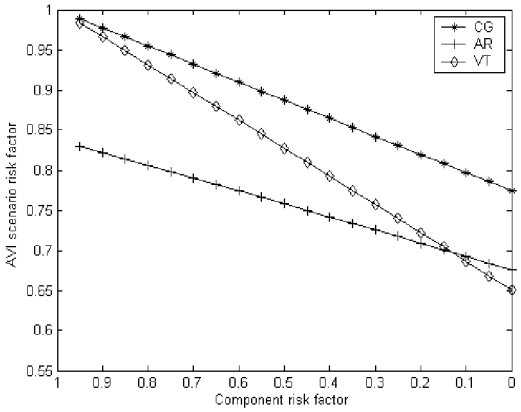

Fig. 11. Sensitivity of the AVI scenario risk factor to the risk factors of the components.

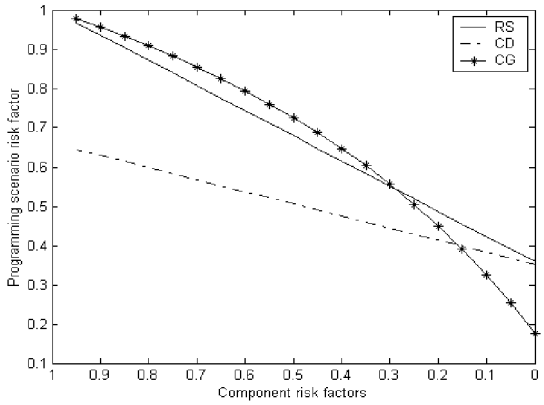

Fig. 12. Sensitivity of the programming scenario risk factor to the risk factors of the components.

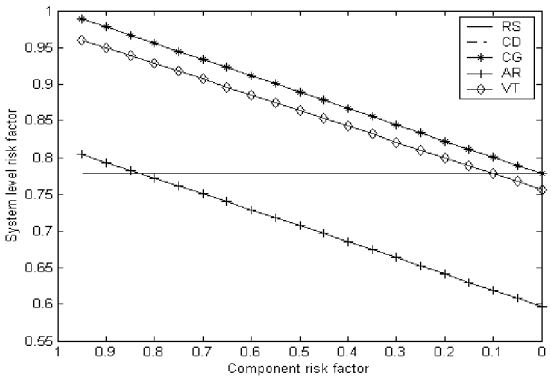

Fig. 13. Sensitivity of the overall system risk factor to the risk factors of the components.

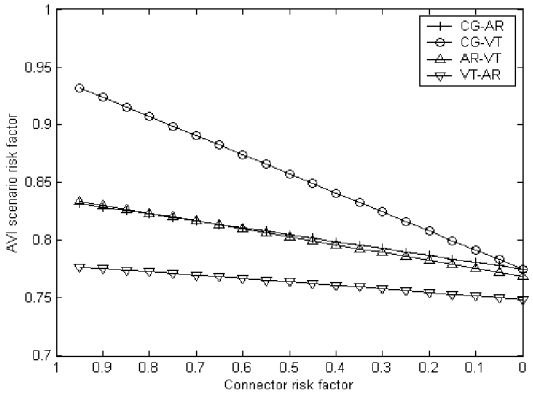

Fig. 14. Sensitivity of the AVI scenario risk factor to the risk factors of the connectors.

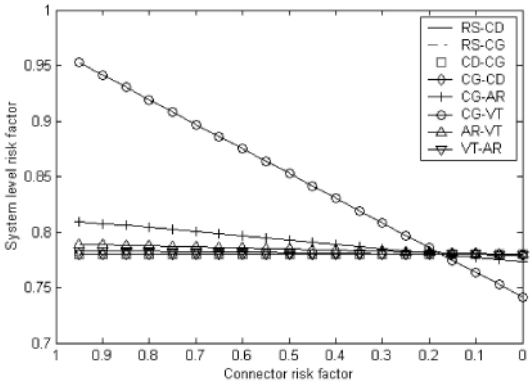

Fig. 15. Sensitivity of the overall system risk factor to the risk factors of the connectors.

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 of 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 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 we 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 the 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.

Acknowledgments

This work is funded in part by a grant from the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP), managed through the

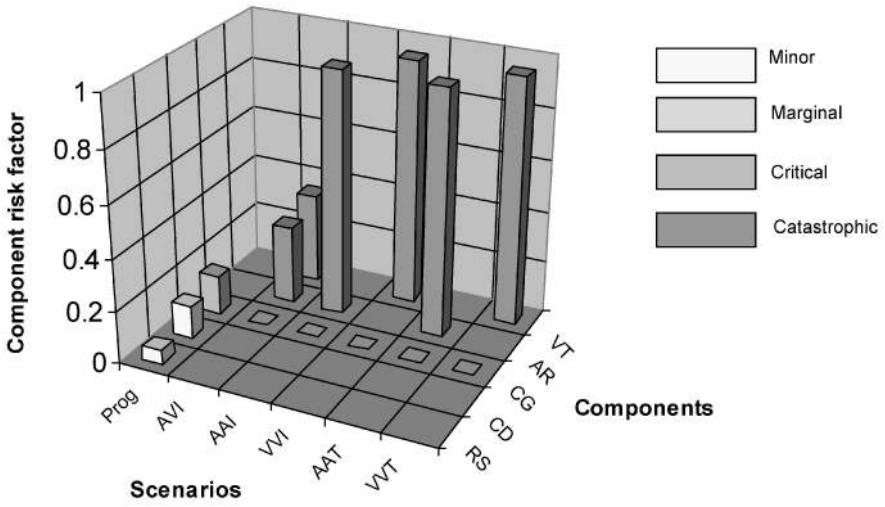

Fig. 16. Identification of the critical components for the pacemaker.

NASA Independent Verification and Validation (IV&V) Facility, Fairmont, West Virgina, and by a grant to the West Virginia University Research Corporation from the US National Science Foundation Information Technology Research (ITR) Program grant number CCR-0082574.

REFERENCES

[1] H. Ammar, T. Nikzadeh, and J. Dugan, "A Methodology for Risk Assessment of Functional Specification of Software Systems Using Coherent Petri Nets," *Proc. Fourth Int'l Software Metrics Symp. (Metrics '97),* pp. 108-117, 1997.

[2] J.M. Bieman and B.K. Kang, "Cohesion and Reuse in an Object-Oriented System," *Proc. ACM Symp. Software Reusability (SSR '94),* pp. 259-262, 1994.

[3] J. Bowles, "The New SEA FMECA Standard," *Proc. Ann. Reliability and Maintainability Symp. (RAMS 1998),* pp. 48-53, 1998.

[4] L. Briand, P. Devanbu, and W. Melo, "An Investigation into Coupling Measures for C++," *Proc. Int'l Conf. Software Eng. (ICSE '97),* pp. 412-421, 1997.

[5] R.C. Cheung, "A User-Oriented Software Reliability Model," *IEEE Trans. Software Eng.,* vol. 6, no. 2, pp. 118-125, 1980.

[6] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Software Eng.,* vol. 20, no. 6, pp. 476-493, 1994.

[7] S.R. Chidamber and C.F. Kemerer, "Towards a Metrics Suite for Object-Oriented Design," *Proc. Conf. Object-Oriented Programming: Systems, Languages and Applications (OOPSLA '91), SIGPLAN Notices,* vol. 26, no. 11, pp. 197-211, 1991.

[8] B. Douglass, *Real-Time UML: Developing Efficient Objects for Embedded Systems.* Addison-Wesley, 1998.

[9] K. El Emam and W. Melo, "The Prediction of Faulty Classes Using Object-Oriented Design Metrics," Technical Report NRC 43609, Nat'l Research Council Canada, Inst. for Information Technology, 1999.

[10] K. El Emam, S. Benlarbi, N. Goel, and S. Rai, "Comparing Case-Based Reasoning Classifiers for Predicting High Risk Software Components," *J. Systems and Software,* vol. 55, pp. 301-320, 2001.

[11] N. Fenton and N. Ohlsson, "Quantitative Analysis of Faults and Failures in a Complex Software System," *IEEE Trans. Software Eng.,* vol. 26, no. 8, pp. 797-814, Aug. 2000.

[12] K. Goseva-Popstojanova and K.S. Trivedi, "Architecture Based Approach to Reliability Assessment of Software Systems," *Performance Evaluation,* vol. 45, nos. 2-3, pp. 179-204, June 2001.

[13] W. Harrison, "Using Software Metrics to Allocate Testing Resources," *J. Management Information Systems,* vol. 4, no. 4, pp. 93-105, 1988.

[14] A. Hassan, W. Abdelmoez, R. Elnaggar, and H. Ammar, "An Approach to Measure the Quality of Software Designs from UML Specifications," *Proc. Fifth World Multi-Conf. Systems, Cybernetics and Informatics,* vol. 4, pp. 559-564, July 2001.

[15] D. Heimann, "Using Complexity Tracking in Software Development," *Proc. Ann. Reliability and Maintainability Symp. (RAMS 1995),* pp. 433-437, 1995.

[16] M. Hitz and B. Montazeri, "Measuring Coupling and Cohesion in Object-Oriented Systems," *Proc. Int'l Symp. Applied Corporate Computing,* pp. 78-84, Oct. 1995.

[17] T. Khoshgoftaar and J. Munson, "Predicting Software Development Errors Using Software Complexity Metrics," *Proc. Software Reliability and Testing,* pp. 20-28, 1995.

[18] T. Khoshgoftaar, J. Munson, and D. Lanning, "Dynamic System Complexity," *Proc. Int'l Software Metrics Symp. (Metrics '93)* pp. 129-140, May 1993.

[19] T. Khoshgoftaar, E. Allen, K. Kalaichelvan, and N. Goel, "The Impact of Software Evolution and Reuse on Software Quality," *Empirical Software Eng.,* vol. 1, pp. 31-44, 1996.

[20] T. Khoshgoftaar, E. Allen, W. Jones, and J. Hudepohl, "Classification Tree Models of Software Quality over Multiple Releases," *Proc. Int'l Symp. Software Reliability Eng. (ISSRE '99),* pp. 116-125, 1999.

[21] J. Musa, G. Fuoco, N. Irving, D. Kropfl, and B. Juhlin, "The Operational Profile," *Handbook of Software Reliability Eng.,* M. Lyu, ed., pp. 167-216, 1996.

[22] T. McCabe, "A Complexity Metrics," *IEEE Trans. Software Eng.,* vol. 2, no. 4, pp. 308-320, Dec. 1976.

[23] J. Munson and T. Khoshgoftaar, "Sotware Metrics for Reliability Assessment," *Handbook of Software Reliability Eng.,* M. Lyu, ed., pp. 493-529, 1996.

[24] *NASA Safety Manual NPG 8715.3,* Jan. 2000.

[25] Rational Rose Real-Time, http://www.rational.com/products/rosert/index.jtmpl, 2003.

[26] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1999.

[27] NASA Technical Std. NASA-STD-8719.13A, *Software Safety,* 1997.

[28] C. Sundararajan, *Guide to Reliability Engineering, Data, Analysis, Applications, Implementation, and Management.* New York: Van Nostrand Reinhold, 1991.

[29] K.S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications,* second ed. John Wiley & Sons, 2002.

[30] T. Wang, A. Hassan, A. Guedem, W. Abdelmoez, K. Goseva-Popstojanova, and H. Ammar, "Architectural Level Risk Assessment Tool Based on UML Specification," *Proc. Int'l Conf. Software Eng. (ICSE 2003),* pp. 808-809, May 2003.

[31] S. Yacoub and H. Ammar, "A Methodology for Architectural-Level Reliability Risk Analysis," *IEEE Trans. Software Eng,* vol. 28, no. 6, pp. 529-547, June 2002.

[32] S. Yacoub, H. Ammar, and T. Robinson, "Dynamic Metrics for Object-Oriented Designs," *Proc. Sixth Int'l Symp. Software Metrics (Metrics '99),* pp 50-61, 1999.

[33] S. Yacoub, T. Robinson, and H. Ammar, "A Matrix-Based Approach to Measure Coupling in Object-Oriented Designs," *J. Object Oriented Programming,* vol. 13, no. 7, pp. 8-19, Nov. 2000.

Katerina Goseva-Popstojanova is an assistant professor in the Lane Department of Computer Science and Electrical Engineering at West Virginia University, Morgantown, West Virginia. Prior to joining West Virginia University, she was a postdoctoral research associate in the Department of Electrical and Computer Engineering at Duke University, Durham, North Carolina. Her research interests include software reliability engineering, dependability, performance and performability assessment, and computer security and survivability. She has published more than 40 journal and conference articles on these topics. She is a senior member of the IEEE and member of the ACM.

Ahmed Hassan received the BSc degree in electrical engineering from Menofia University, Egypt and the MSc degree in artificial intelligence applications in power system, Mansoura University, Egypt. He is a PhD student at West Virginia University, Morgantown, West Virginia. His research interests are software hazard analysis, software metrics, and software risk assessment. He is a student member of the IEEE.

Ajith Guedem received the BTech degree in computer science and information technology from Jawaharlal Nehru Technological University, Hyderabad, India. Currently, he is pursuing the MS degree in computer science at West Virginia University, Morgantown, West Virginia. His research interests are software reliability, risk assessment, distributed systems, and computer security.