A Component-based Approach for Assessing Reliability of Compound Software

Monica Lind Kristiansen[a*], Bent Natvig[b], and Harald Holone[c]
[a]Department of Informatics, Østfold University College, Halden, Norway
[b]Department of Mathematics, University of Oslo, Oslo, Norway
[c]Department of Informatics, Østfold University College, Halden, Norway

Abstract: Predicting the reliability of software systems based on a component approach is inherently difficult, in particular due to failure dependencies between the software components. This paper describes a component-based approach for assessing reliability of compound software, where failure dependencies between software components are explicitly addressed. This is done by finding accepted upper bounds for probabilities that pairs of software components fail simultaneously and then by including these into the reliability models. To find these accepted upper bounds, the approach applies principles of Bayesian hypothesis testing on simultaneous failure probabilities. In addition, the restrictions imposed on the simultaneous reliabilities and failure probabilities by the marginal reliabilities and failure probabilities are taken into account. To illustrate the approach, we use an example based on mobile positioning systems for backtracking. This is for instance used to help people with dementia to find their way home if they get lost.

Keywords: Failure dependencies, Component-based approach, Bayesian hypothesis testing, Mobile positioning systems.

1. INTRODUCTION

Before computerized systems can be used in any kind of critical applications, evidences that these systems are dependable are required [1]. This is desirable for most systems, but essential for systems which affect human safety and welfare, e.g. mobile positioning systems, patient monitoring systems, fly-by-wire systems, railway signal systems etc. Considering that most computerized systems are built as a structure consisting of several software components, of which some might have been pre-developed and used in other contexts, there is a need for methods for assessing reliability of compound software [1].

Although several different approaches to construct component-based software reliability models have been proposed in, among others, Cortellessa and Grassi [2], Gokhale and Trivedi [3], Gokhale [4], Palviainen et al. [5], Goseva-Popstojanova and Trivedi [6], Hamlet [7,8], Krishnamurthy and Mathur [9], Krka et al. [10], Kuball et al. [11], Popic et al. [12], Reussner et al. [13], Singh et al. [14], Trung and Thang [15], Vieira and Richardson [16], and Yacoub et al. [17], most of these approaches tend to ignore failure dependencies between software components [18–20]. This topic is discussed more thoroughly in, among others, Cortellessa and Grassi [2], Dai et al. [21], Gokhale and Trivedi [4], Guo et al. [22], Littlewood et al. [23], Lyu [1], Nicola and Goyal [24], Popic et al. [12], Popov et al. [25], and Tomek et al. [26].

In our research, we have developed a component-based approach for assessing reliability of compound software in which failure dependencies between software components are explicitly addressed [27].

* Corresponding author, monica.kristiansen@hiof.no
[1] Software systems consisting of multiple software components.

The idea is to find accepted upper bounds for probabilities that pairs of software components fail simultaneously and then include these into the reliability models. To find these accepted upper bounds, the approach applies principles of Bayesian hypothesis testing [28] on simultaneous failure probabilities. This includes taking into account the restrictions the marginal reliabilities and failure probabilities put on the simultaneous reliabilities and failure probabilities.

In Section 2, the motivation of our research is given. This is followed up by a description of our component-based approach in Section 3. This section also describes the assumptions of the approach and the direct restrictions on the simultaneous probabilities. In addition, some general rules for selecting the most important component dependencies are given. These rules are based on the concepts of data-parallel and data-serial components, which are defined in Section 3.3. In Section 4, the component-based approach is illustrated with an example from positioning services on mobile phones for geofencing and backtracking. A summary of the results, some conclusions and ideas for further work are given in Section 5.

2. BACKGROUND

Our research started by analyzing two interesting papers written by Cukic et al. [29] and Smidts et al. [30]. These papers present a Bayesian hypothesis testing approach for finding upper bounds for failure probabilities of single software components. The authors' idea is to complement testing with available prior information regarding the software components so that adequate confidence can be obtained with a feasible amount of testing.

In their approach, the null hypothesis (H_0) and the alternative hypothesis (H_1) are specified as: $H_0 : \theta \leq \theta_0$ and $H_1 : \theta > \theta_0$, where θ_0 is a probability in the interval $(0,1)$ representing the upper bound for the failure probability θ of a software component. The upper bound θ_0 is assumed to be context specific and predefined and is typically derived from standards, regulation authorities, customers, etc. In this case, the null hypothesis states that the software component's failure probability is lower than the given predefined upper bound θ_0, whereas the alternative hypothesis states that the software component's failure probability is higher than the given predefined upper bound θ_0.

Furthermore, the authors describe the prior belief in the failure probability ($\pi(\theta)$) of a single software component using two separate uniform probability distributions, one under the null hypothesis and one under the alternative hypothesis. Based on this assumption, the authors show that the number of tests (D) required to obtain an adequate confidence level C_0 so that $P(H_0|D) \geq C_0$, can be significantly reduced compared to the situation where no prior belief regarding the software component is described. By assuming that prior belief in the null hypothesis $P(H_0)$ is 0.01, the predefined upper bound θ_0 is 0.0001, and the confidence level C_0 is 0.99, the authors show that it requires 6831 fault-free tests to reach the confidence level by using Bayesian hypothesis testing compared to 46050 fault-free tests by using classical statistical testing. They also demonstrate that the higher the prior belief in the null hypothesis is, the fewer tests are needed to obtain adequate confidence in the software component.

Although we think that the principles of the Bayesian hypothesis testing approach proposed in Cukic et al. [29] and Smidts et al. [30] are usable, even for compound software, our main concern is related to the use of two separate uniform probability distributions to describe the prior belief in the failure probability of a single software component. This concern is addressed in Kristiansen [31], in which an evaluation of the Bayesian hypothesis testing approach is performed. In this paper, three different prior probability distributions for the failure probability of a software component are evaluated, and their influence on the number of tests required to obtain adequate confidence in a software component is presented. In this evaluation, the first case is based on earlier work done by Cukic et al. [29] and

Smidts et al. [30] and assumes two separate uniform prior probability distributions, one under the null hypothesis and one under the alternative hypothesis. In the second case, the effect of using a flat distribution under the alternative hypothesis is mitigated by allowing an expert to set an upper bound on the failure probability under H_1, i.e. to state a value θ_1 for which the probability of having a failure probability higher than θ_1 is zero. In the third case, the effect of discontinuity in the prior probability distribution is mitigated by using a continuous probability distribution for θ over the entire interval $(0,1)$. A beta distribution is used to accurately reflect prior belief because this distribution is a rich and tractable family that forms a conjugate family to the binomial distribution.

The evaluation in Kristiansen [31] clearly shows that using two separate uniform distributions to describe the failure probability of a software component does not represent a conservative approach at all, even though the use of a uniform probability distribution over the entire interval is usually seen as an ignorance prior. In fact, the number of tests required to obtain adequate confidence in a software component increases significantly when other more realistic distributions for the failure probability of a software component are used. Moreover, it is shown that the total number of tests required by using this approach can both result in fewer and even in more tests compared to classical statistical testing. This means that in the Bayesian hypothesis testing approach, the number of required tests is highly dependent on the choice of prior distribution. It should therefore be emphasized that it is the underlying prior distribution for the failure probability of a software component and underlying assumptions that lead to fewer tests rather than the Bayesian hypothesis testing approach. To choose a prior probability distribution for a software component's failure probability that correctly reflects ones prior belief is therefore of great importance.

3. A COMPONENT-BASED APPROACH FOR ASSESSING RELIABILITY OF COMPOUND SOFTWARE

In Kristiansen [27], we propose a component-based approach for assessing reliability of compound software. In this approach, failure dependencies between software components are addressed explicitly by using Bayesian hypothesis testing [28] on simultaneous failure probabilities. It is assumed that failure probabilities of individual software components are known. The approach consists of five basic steps:

1. Identify the most important component dependencies: based on the structure of the software components in the compound software and information regarding individual software components, identify those dependencies between pairs of software components which are of greatest importance for the calculation of the system reliability [32]. Repeat steps 2-4 for all relevant component dependencies in the system.
2. Define the hypotheses: let $q_{0,ij}$ represent an accepted upper bound for the probability (q_{ij}) that a pair (i,j) of software components fails simultaneously. The upper bound $q_{0,ij}$ is assumed to be context specific and predefined and is typically derived from standards, regulation authorities, customers, etc. Define the following hypotheses:

$$H_0 : a_{ij} \leq q_{ij} \leq q_{0,ij}$$

$$H_1 : q_{0,ij} < q_{ij} \leq b_{ij},$$

   where q_{ij} is defined in the interval $[a_{ij}, b_{ij}]$. The interval limits a_{ij} and b_{ij} represent the lower and upper limit for q_{ij}, respectively, and are decided by the restrictions the components' marginal failure probabilities put on the components' simultaneous failure probabilities [32].
3. Describe prior belief regarding probability q_{ij}: establish a prior probability distribution $g(q_{ij})$ for q_{ij} defined in the interval $[a_{ij}, b_{ij}]$ describing the probability that a pair of software components fails simultaneously [33]. This probability distribution is needed for establishing a prior

probability distribution $\pi(q_{ij})$ for q_{ij} defined in the sub-intervals $[a_{ij}, q_{0,ij}]$ and $[q_{0,ij}, b_{ij}]$ and for calculating $P(H_0)$.

4. Update your belief in hypothesis H_0 through testing: based on the prior belief in the null hypothesis $P(H_0)$ from step 3 and a predefined confidence level $C_{0,ij}$, the number of tests required to obtain an adequate upper bound for the probability of simultaneous failure can be found for different numbers of failures encountered during testing. The more failures that occur during testing, the more tests are required to reach $C_{0,ij}$. For further details on when to stop testing see Cukic et al. [29] or Kristiansen et al. [33].
5. Calculate the complete system's failure probability: information regarding failure probabilities of individual software components (which are assumed to be known) and upper bounds for the most important simultaneous failure probabilities (found in step 1-4) can finally be combined to obtain an upper bound for the failure probability of the entire system. This can be performed by various methods, e.g. by discrete event simulation when direct calculation becomes too complicated.

3.1. Assumptions

The component-based approach for assessing reliability of compound software is based on the following assumptions:

1. The states of the software components are associated random variables [35].
2. All data-flow relations between the software components are known.
3. The reliabilities of the individual software components are known [1, 36, 37].
4. The system and its components have only two possible states (functioning and failure) [38].
5. The system has a monotone structure [39].

Furthermore, the research is restricted to on-demand types of situations where the compound software is given an input and execution is considered to be finished when a corresponding output has been produced. In the following, these assumptions are discussed briefly.

3.2. Restrictions on the components' simultaneous reliabilities and failure probabilities

Assumption 1 and 3 put direct restrictions on the components' simultaneous reliabilities and failure probabilities. To show this, let p_i and p_j denote the reliabilities of components i and j in a simple two component system. Furthermore, let p_{ij} denote the simultaneous reliability of components i and j. If it can be assumed that the components' reliabilities do not change due to changes in operational context, the following is true: $p_{ij} \leq min(p_i, p_j)$. This follows directly from the fact that: $p_{ij} = p_{i|j}p_j = p_{j|i}p_i$, where $p_{i|j}$ and $p_{j|i}$ are conditional reliabilities between components i and j.

Furthermore, since we assume that the states of the software components are associated random variables, it follows that [35]: $p_{ij} \geq p_i p_j$. Reasonable constraints on the simultaneous reliability p_{ij} under the given assumptions can therefore be expressed as follows: $p_i p_j \leq p_{ij} \leq min(p_i, p_j)$. In the same way, under the same assumption, it can be shown that: $q_i q_j \leq q_{ij} \leq min(q_i, q_j)$.

How the reliabilities of individual software components put direct restrictions on the components' conditional reliabilities in general systems consisting of two and three components are elaborated in more detail in Kristiansen et al. [32].

3.3. General rules for selecting the most important component dependencies

To identify possible rules for selecting the most important component dependencies, two new concepts were defined in Kristiansen et al. [32]. These concepts contribute to a deeper understanding of how to

include component dependencies in reliability modeling, and are given in Definitions 1 and 2.

Definition 1 Data-serial components: two components i and j are said to be data-serial components if either i or j receives data (d), directly or indirectly through other components, from the other.

$$i \xrightarrow{d} j \quad or \quad j \xrightarrow{d} i$$

Definition 2 Data-parallel components: two components i and j are said to be data-parallel components if neither i nor j receives data (d), directly or indirectly through other components, from the other.

$$i \xrightarrow{d} j \quad and \quad j \xrightarrow{d} i$$

Based on these definitions, the rules for selecting the most important component dependencies can be summarized as follows [40]:

- Including only partial dependency information may give a substantial improvement in the reliability predictions compared to assuming independence between all software components as long as the most important component dependencies are included.
- It is also clear that dependencies between data-parallel components are far more important than dependencies between data-serial components.

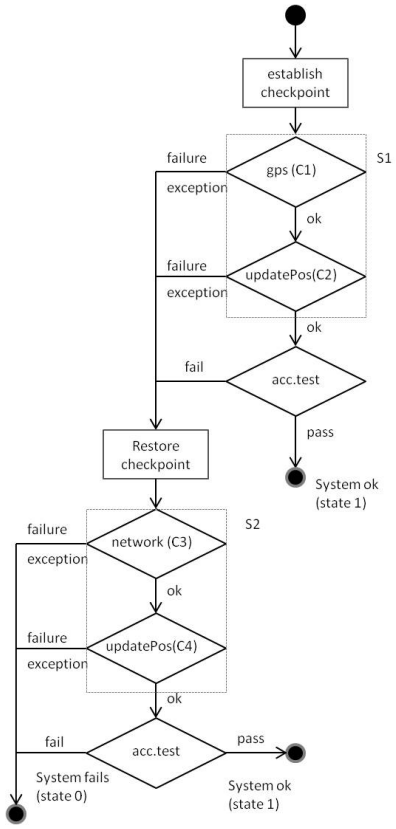
In addition, for a system consisting of both data-parallel and data-serial components, the following applies:

- Including only dependencies between data-serial components may result in a major overestimation of the system's reliability. In some cases, the results are even worse than by assuming independence between all components.
- Including only dependencies between data-parallel components may give predictions close to the system's true reliability as long as the dependency between the most unreliable components is included.
- Including additional dependencies between data-parallel components may further improve the predictions.
- Including additional dependencies between data-serial components may also give better predictions as long as the dependency between the most reliable components is included.

4. CASE

One increasingly popular application of mobile positioning systems is to provide geofencing and backtracking services, for instance for kids or people with dementia. Geofencing is used to alert the person or her family members if the person moves outside of a predefined virtual fence, typically surrounding the neighborhood where the person lives and is familiar. Backtracking systems are typically used for helping the user to find the way back to this familiar area if he/she gets lost. Both these services require reliable positioning of the mobile device. In addition, the backtracker needs a position log, keeping track of the users movements over a period of time. Our example case is a backtracking system.

To ensure effective fault tolerance in the software system, let's assume it is structured as a typical recovery block [41]. Individual system components raise exceptions when they detect errors that their own fault tolerance capabilities are unable to handle. Our example system consists of two independently developed software components capable of establishing and recording the geographical



(a) A control flow diagram describing the mobile positioning system.

x_1	x_2	x_3	x_4	$\phi(\mathbf{x})$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

(b) Possible component and system states.

Figure 1: Example case

position of a mobile device. We refer to these two components as super component 1 (S1) and 2 (S2). In addition, a controller is responsible for defining the interactions between the components and performing acceptance tests on the components' output. The system is illustrated in Figure 1(a).

S1 consists of two sub components, C1 and C2. C1 uses the GPS receiver on the mobile phone for establishing the device's geographical position. C2 takes information from C1, transforms it, and updates the position log. The other super component, S2, consists of sub components C3 and C4. C3 uses mobile network and WiFi information to establish the device's geographical position, and C4 updates the position log after transforming the output from C3 to a suitable form. For simplification, it is assumed that the controller, including the acceptance test, is fault free.

S1 and S2 are not run in parallel like in N-version programming. When logging the current device position is requested, the controller first establishes a checkpoint of the system state (including the

position log) to permit recovery. Then, C1 is invoked to establish the current device position. On success, the controller passes the result from C1 to C2, which transforms it to a suitable form and adds the current position to the position log. The acceptance test is used by the controller as a final check of the newly logged position. If the acceptance test fails, or if C1 or C2 raises an exception, the controller restores the prior system state, and invokes C3 to provide the position. On C3's success, C4 gets position information from C3, transforms it and adds it to the position log. If the resulting update to the position log fails the acceptance test, or if C3 or C4 raises an exception, the system fails.

Let's assume that the software system and its components have binary states. Let x_i $i = 1,2,3,4$ indicate the state of the ith component at a fixed point in time and let $\mathbf{x} = (x_1, x_2, x_3, x_4)$. Let's further assume that the system state, ϕ, at a fixed point in time is uniquely determined by the states of the components, $\mathbf{x}$, i.e. $\phi = \phi(\mathbf{x})$. For $i = 1,2,3,4$, let $x_i = 0$ if component i fails and 1 if component i functions. The possible component and system states are given in Figure 1(b), which includes the following two system states:

$$\begin{aligned} \phi(\mathbf{x}) = 0 &= \text{system fails} \\ \phi(\mathbf{x}) = 1 &= \text{system functions} \end{aligned}$$

Based on the software system's minimal path sets, the reliability of the software system is given by: $P(\phi(\mathbf{x}) = 1) = p_{12} + p_{34} - p_{1234}$. Let's assume that the marginal reliabilities and failure probabilities for components 1, 2, 3 and 4 are known. This knowledge and the assumption that the states of the software components are associated random variables, put direct restrictions on the components' simultaneous reliabilities and failure probabilities (see Section 3.2 for more details). This is shown in Table 1. Based on the restrictions on the components' simultaneous reliabilities in Table 1, the reliability of the complete system must be within the interval $[0.9988011, 1.0000]$.

Table 1: The components marginal reliabilities and failure probabilities and their restrictions on the components simultaneous reliabilities and failure probabilities.

Marginal reliabilities	Marginal failure probabilities
$p_1 = 0.99$	$q_1 = 0.01$
$p_2 = 0.9999$	$q_2 = 0.0001$
$p_3 = 0.999$	$q_3 = 0.001$
$p_4 = 0.9999$	$q_4 = 0.0001$
Simultaneous reliabilities	Simultaneous failure probabilities
$p_{12} \in [0.989901, 0.99]$	$q_{12} \in [10^{-6}, 10^{-4}]$
$p_{13} \in [0.98901, 0.99]$	$q_{13} \in [10^{-5}, 10^{-3}]$
$p_{14} \in [0.989901, 0.99]$	$q_{14} \in [10^{-6}, 10^{-4}]$
$p_{23} \in [0.9989001, 0.999]$	$q_{23} \in [10^{-7}, 10^{-4}]$
$p_{24} \in [0.99980001, 0.9999]$	$q_{24} \in [10^{-8}, 10^{-4}]$
$p_{34} \in [0.9989001, 0.999]$	$q_{34} \in [10^{-7}, 10^{-4}]$
$p_{123} \in [0.988911099, 0.99]$	$q_{123} \in [10^{-9}, 10^{-4}]$
$p_{124} \in [0.98980201, 0.99]$	$q_{124} \in [10^{-10}, 10^{-4}]$
$p_{134} \in [0.988911099, 0.99]$	$q_{134} \in [10^{-9}, 10^{-4}]$
$p_{234} \in [0.99880021, 0.999]$	$q_{234} \in [10^{-11}, 10^{-4}]$
$p_{1234} \in [0.988812208, 0.99]$	$q_{1234} \in [10^{-13}, 10^{-4}]$

One of the rules for selecting the most important component dependencies in Kristiansen et al. [32], state that including the dependency between the most unreliable data-parallel components gives predictions close to the system's true reliability. In our example, the most unreliable data-parallel components are

components 1 and 3. When only including the dependency between components 1 and 3, the reliability of the complete software system becomes: $P(\phi(\mathbf{x}) = 1) = p_1p_2 + p_3p_4 - p_4p_{3|1}p_2p_1$.

Let's assume that it is required that the reliability of the software system ($P(\phi(\mathbf{x}) = 1)$ is at least 0.9999 with confidence level $C_0 = 0.99$. Based on this requirement and the addition law of probability, a predefined upper bound $q_{0,13}$ for the simultaneous failure probability q_{13} can be calculated, This is shown in Equation 1.

$$\begin{aligned} q_{13} &= 1 - p_1 - p_3 + p_{13} \leq q_{0,13} \\ &= 1 - p_1 - p_3 + (\frac{p_1p_2 + p_3p_4 - 0.9999}{p_2p_4}) \\ &= 0.00009891 \end{aligned} \tag{1}$$

Based on the upper bound $q_{0,13}$ and the restrictions $[a_{13}, b_{13}]$ on the simultaneous failure probability q_{13} given in Table 1, the following hypotheses can be defined (see step 2 in the component-based approach in Section 3 for details):

$$\begin{aligned} &H_0 : 0.00001 \leq q_{13} \leq 0.00009891 \\ &H_1 : 0.00009891 < q_{13} \leq 0.001. \end{aligned} \tag{2}$$

In this case, the null hypothesis states that the simultaneous failure probability q_{13} lies between the lower bound $a_{13} = 0.00001$ and the predefined upper bound $q_{0,13}$, whereas the alternative hypothesis states that the simultaneous failure probability q_{13} lies between the predefined upper bound $q_{0,13}$ and the upper bound $b_{13} = 0.001$.

In the following, the number of fault free tests, n, required to obtain the upper bound $q_{0,13}$ with confidence level $C_{0,13}$ is calculated using:

1. Classical statistical testing, where no prior information about the simultaneous failure probability q_{13} is included.
2. Bayesian hypothesis testing, where only the restrictions imposed on q_{13} by the marginal failure probabilities q_1 and q_3 and the assumption that the states of the software components are associated random variables are taken into account.
3. Bayesian hypothesis testing, where additional prior information about the simultaneous failure probability q_{13} is taken into account.

The number of fault free tests, n, required to obtain the upper bound $q_{0,13}$ at the given predefined confidence level $C_{0,13} = 0.99$, using classical statistical testing, is given in Equation 3 [42].

$$n = \frac{ln(1 - C_{0,13})}{ln(1 - q_{0,13})} = 46557 \tag{3}$$

This means that if no prior information in included, 46557 fault free tests need to be carried out to obtain the upper bound $q_{0,13}$ with confidence level $C_{0,13} = 0.99$.

If we only take into account the restrictions imposed on q_{13} by the marginal failure probabilities q_1 and q_3 and the assumption that the states of the software components are associated random variables, the number of fault free tests required to obtain an adequate upper bound for q_{13}, at the given predefined confidence level $C_{0,13} = 0.99$, can be found by solving Equation 4 [33].

$$\frac{\int_{a_{13}}^{q_{0,13}} (1-q_{13})^n dq_{13}}{\int_{q_{0,13}}^{b_{13}} (1-q_{13})^n dq_{13}} \geq \frac{C_{0,13}}{(1-C_{0,13})} \tag{4}$$

In this equation, q_{13} is uniformally distributed (beta distributed with parameters $\alpha = \beta = 1$) over the interval $[a_{13}, b_{13}]$. Taking only these restrictions into account, 51793 fault free tests need to be carried out to obtain the simultaneous failure probability $q_{0,13}$ at confidence level $C_{0,13}$. This means that the number of fault free tests in fact increases when only the restrictions from the marginal failure probabilities are taken into account.

One way to include additional information about the simultaneous failure probability q_{13}, in addition to the restrictions from the marginal probabilities q_1 and q_3, is to identify values for α and β in the beta distribution by visualizing a controlled experiment. In this experiment, n can be considered as the total number of tests and α as the number of simultaneous failures of components 1 and 3. Assuming this prior information, the number of fault free tests required to obtain an adequate upper bound for the simultaneous failure probability q_{13}, at the given predefined confidence level $C_{0,13} = 0.99$ for different choices of α and β, can be found by solving Equation 5 [33].

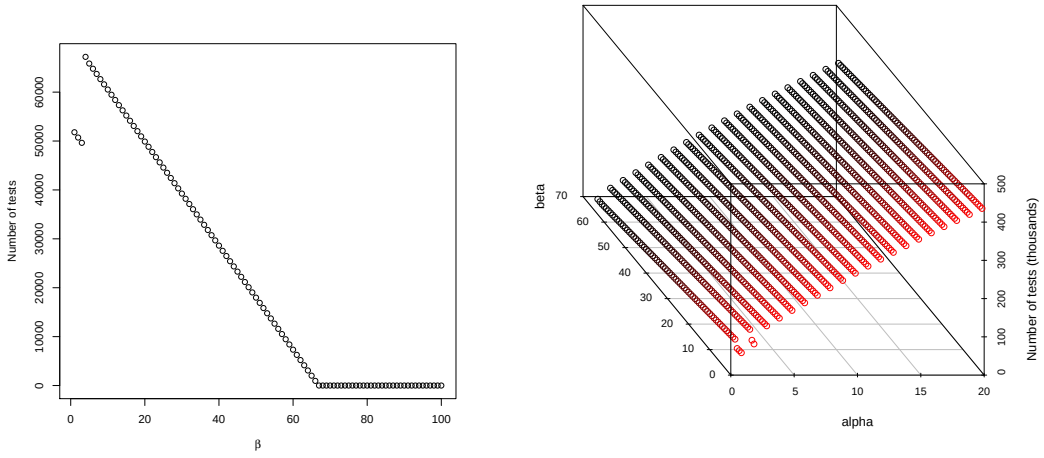
$$\frac{\int_{a_{13}}^{q_{0,13}} (q_{13}-a_{13})^{\alpha-1} (b_{13}-q_{13})^{\beta-1} (1-q_{13})^n dq_{13}}{\int_{q_{0,13}}^{b_{13}} (q_{13}-a_{13})^{\alpha-1} (b_{13}-q_{13})^{\beta-1} (1-q_{13})^n dq_{13}} \geq \frac{C_{0,13}}{(1-C_{0,13})} \tag{5}$$

In Figures 2(a) and 2(b), the number of tests required are illustrated graphically for different choices of α and β in the beta distribution. From these figures, it is clear that the number of fault free tests, n, is highly sensitive to the choices of α and β. For example $n = 951$ fault free tests are required when $\alpha = 1$ and $\beta = 66$.

In addition to the individual software components' failure probabilities q_1 and q_3, information regarding components' architecture, complexity, programming languages, development processes, etc. might as well be available. This information is also relevant for assessing q_{13}, and will be explored further in future work.

5. SUMMARY, CONCLUSIONS AND FURTHER WORK

In this paper, we have presented a component-based approach for assessing reliability of compound software. This approach applies binary reliability theory to explicitly handle failure dependencies between software components. The approach has been elaborated through several experimental studies [31–33, 40]. To illustrate the approach, we have used a backtracking system typically used for helping people to find their way home if they get lost. This system consists of a recovery block containing two independently developed software components capable of establishing and recording the geographical position of a mobile device.



(a) When $\alpha = 1$ and β varies from 1 to 100.

(b) When α varies from 1 to 20 and β varies from 1 to 70.

Figure 2: Number of fault free test required to obtain an adequate upper bound for the simultaneous failure probability q_{13}, at the given predefined confidence level $C_{0,13} = 0.99$.

Using this system, we first illustrate how the components' simultaneous reliabilities and failure probabilities are imposed by a) the components' marginal reliabilities and failure probabilities and b) the assumption that the states of the software components are associated random variables. Secondly, we show how the general rules for selecting the most important component dependencies can be applied on a system consisting of four components. Finally, we calculate the number of fault free tests required to obtain the upper bound for a simultaneous failure probability with a given confidence level. The results show that the number of required tests to obtain an upper bound is highly sensitive to the choices of α and β in the beta distribution. It is noteworthy that the number of tests can both increase and decrease compared to classical statistical testing depending on the choice of these parameters.

Further work includes investigating the main challenges of our component-based approach when we go from binary systems of binary components to multistate systems of binary or multistate components. Especially, we will look at the effect of using only partial dependency information when assessing reliability of multistate systems of binary and multistate components. To find the most important component dependencies in each system state, direct calculation, the Birnbaum measure [38] and Principal Component Analysis [43] will be investigated. A critical question is if these techniques identify the same component dependencies as the most important ones in all system states, or if the most important component dependencies vary between different system states.

References

[1] M. R. Lyu, ed., *Handbook of Software Reliability Engineering*. IEEE Computer Society Press, 1995.

[2] V. Cortellessa and V. Grassi, "A modeling approach to analyze the impact of error propagation on reliability of component-based systems," *Proceedings of the 10th International Conference on Component-based Software Engineering*, pp. 140–156, 2007.

[3] S. S. Gokhale, "Architecture-based software reliability analysis: Overview and limitations," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 1, pp. 32–40, 2007.

[4] S. S. Gokhale and K. S. Trivedi, "Dependency Characterization in Path-Based Approaches to Architecture-Based Software Reliability Prediction," *IEEE Workshop on Application-Specific Software Engineering and*

Technology, pp. 86–90, 1998.

[5] M. Palviainen, A. Evesti, and E. Ovaska, "The reliability estimation, prediction and measuring of component-based software," *Journal of Systems and Software*, vol. 84, no. 6, pp. 1054–1070, 2011.

[6] K. Goseva-Popstojanova and K. S. Trivedi, "Architecture-based approach to reliability assessment of software systems," *Performance Evaluation*, vol. 45, no. 2-3, pp. 179–204, 2001.

[7] D. Hamlet, "Software component composition: a subdomain-based testing-theory foundation," *Software Testing, Verification and Reliability*, vol. 17, no. 4, pp. 243–269, 2007.

[8] D. Hamlet, D. Mason, and D. Woit, "Theory of Software Reliability Based on Components," *International Conference on Software Engineering*, vol. 23, pp. 361–370, 2001.

[9] S. Krishnamurthy and A. Mathur, "On the Estimation of Reliability of a Software System Using Reliabilities of its Components," *Proceedings of the 8th International Symposium on Software Reliability Engineering (ISSRE'97)*, pp. 146–155, 1997.

[10] I. Krka, G. Edwards, L. Cheung, L. Golubchik, and N. Medvidovic, "A comprehensive exploration of challenges in Architecture-Based reliability estimation," *Architecting Dependable Systems VI*, pp. 202–227, 2009.

[11] S. Kuball, J. May, and G. Hughes, "Building a system failure rate estimator by identifying component failure rates," *Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE'99)*, pp. 32–41, 1999.

[12] P. Popic, D. Desovski, W. Abdelmoez, and B. Cukic, "Error Propagation in the Reliability Analysis of Component based Systems," *Proceedings of the 16th IEEE International Symposium on Software Reliability (ISSRE'05)*, pp. 53–62, 2005.

[13] R. H. Reussner, H. W. Schmidt, and I. H. Poernomo, "Reliability prediction for component-based software architectures," *Journal of Systems and Software*, vol. 66, no. 3, pp. 241–252, 2003.

[14] H. Singh, V. Cortellessa, B. Cukic, E. Gunel, and V. Bharadwaj, "A Bayesian approach to reliability prediction and assessment of component based systems," *Proceedings of the 12th IEEE International Symposium on Software Reliability Engineering (ISSRE'01)*, pp. 12–19, 2001.

[15] P. T. Trung and H. Q. Thang, "Building the reliability prediction model of component-based software architectures," *Int'l Journal of Information Technology*, vol. 5, no. 1, pp. 18–25, 2009.

[16] M. Vieira and D. Richardson, "The role of dependencies in component-based systems evolution," *Proceedings of the International Workshop on Principles of Software Evolution*, pp. 62–65, 2002.

[17] S. Yacoub, B. Cukic, and H. Ammar, "A Scenario-Based Reliability Analysis Approach for Component-based Software," *IEEE Transactions on Reliability*, vol. 53, no. 4, pp. 465–480, 2004.

[18] D. E. Eckhardt and L. D. Lee, "A theoretical basis for the analysis of redundant software subject to coincident errors," tech. rep., Memo 86369, NASA, 1985.

[19] J. C. Knight and N. G. Leveson, "An experimental evaluation of the assumption of independence in multiversion programming," *IEEE Transactions on Software Engineering*, vol. 12(1), pp. 96–109, 1986.

[20] B. Littlewood and D. R. Miller, "Conceptual Modeling of Coincident Failures in Multiversion Software," *IEEE Transactions on Software Engineering*, vol. 15(12), pp. 1596–1614, 1989.

[21] Y. Dai, M. Xie, K. Poh, and S. Ng, "A model for correlated failures in N-version programming," *IIE Transactions*, vol. 36, no. 12, pp. 1183–1192, 2004.

[22] P. Guo, X. Liu, and Q. Yin, "Methodology for Reliability Evaluation of N-Version Programming Software Fault Tolerance System," *International Conference on Computer Science and Software Engineering*, pp. 654–657, 2008.

[23] B. Littlewood, P. Popov, and L. Strigini, "Modelling software design diversity: a review," *ACM Computing Surveys*, vol. 33, no. 2, pp. 177–208, 2001.

[24] V. F. Nicola and A. Goyal, "Modeling of correlated failures and community error recovery in multiversion software," *IEEE Transactions on Software Engineering*, vol. 16, no. 3, pp. 350–359, 1990.

[25] P. Popov, L. Strigini, J. May, and S. Kuball, "Estimating Bounds on the Reliability of Diverse Systems," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 345–359, 2003.

[26] L. A. Tomek, J. K. Muppala, and K. S. Trivedi, "Modeling Correlation in Software Recovery Blocks," *IEEE Transactions on Software Engineering*, vol. 19, no. 11, pp. 1071–1086, 1993.

[27] M. Kristiansen, *A component-based approach for assessing reliability of compound software*. PhD thesis, Faculty of Matematics and Natural Sciences, University of Oslo, 2011.

[28] J. O. Berger, *Statistical Decision Theory and Bayesian Analysis*. Springer Verlag, second ed., 1985.

[29] B. Cukic, E. Gunel, H. Singh, and L. Guo, "The Theory of Software Reliability Corroboration," *IEICE Transactions on Information and Systems*, vol. E86-D, no. 10, pp. 2121–2129, 2003.

[30] C. Smidts, B. Cukic, E. Gunel, M. Li, and H. Singh, "Software Reliability Corroboration," *Proceedings of the 27'th Annual NASA Goddard Software Engineering Workshop (SEW-27'02)*, pp. 82–87, 2002.

[31] M. Kristiansen, "Finding Upper Bounds for Software Failure Probabilities - Experiments and Results," *Computer Safety, Reliability and Security (Safecomp 2005)*, pp. 179–193, 2005.

[32] M. Kristiansen, R. Winther, and B. Natvig, "On Component Dependencies in Compound Software," *International Journal of Reliability, Quality and Safety Engineering (IJRQSE)*, vol. 17, no. 5, pp. 465–493, 2010.

[33] M. Kristiansen, R. Winther, and B. Natvig, "A Bayesian Hypothesis Testing Approach for Finding Upper Bounds for Probabilities that Pairs of Software Components Fail Simultaneously," *International Journal of Reliability, Quality and Safety Engineering (IJRQSE)*, vol. 18, no. 3, pp. 209–236, 2011.

[34] M. Kristiansen, R. Winther, and J. E. Simensen, "Identifying the Most Important Component Dependencies in Compound Software," *Risk, Reliability and Safety (ESREL 2009)*, pp. 1333–1340, 2009.

[35] R. E. Barlow and F. Proschan, *Statistical theory of reliability and life testing: probability models*. Holt, Rinehart and Winston, 1975.

[36] B. Littlewood and L. Strigini, "Guidelines for the statistical testing of software," tech. rep., City University, London, 1998.

[37] J. D. Musa, *Software Reliability Engineering*. McGraw-Hill, 1998.

[38] B. Natvig, *Multistate Systems Reliability Theory with Applications*. John Wiley and Sons, Ltd, first ed., 2011.

[39] B. Natvig, *Reliability analysis with technological applications (in Norwegian)*. Department of Mathematics, University of Oslo, 1998.

[40] M. Kristiansen, R. Winther, and B. Natvig, "On Component Dependencies in Compound Software," tech. rep., Department of Mathematics, University of Oslo, 2010.

[41] B. Randell and J. Xu, "The evolution of the recovery block concept," 1995.

[42] J. H. Poore, H. D. Mills, and D. Mutchler, "Planning and Certifying Software System reliability," *IEEE software*, 1993.

[43] R. A. Johnson and D. W. Wichern, *Applied Multivariate Statistical Analysis*. Prentice Hall, New York, 1998.