International Journal of Reliability, Quality and Safety Engineering
© World Scientific Publishing Company

Dept. of Math. University of Oslo
Statistical Research Report No 1
ISSN 0806–3842 March 2011

A BAYESIAN HYPOTHESIS TESTING APPROACH FOR FINDING UPPER BOUNDS FOR PROBABILITIES THAT PAIRS OF SOFTWARE COMPONENTS FAIL SIMULTANEOUSLY

MONICA KRISTIANSEN *
Østfold University College
1757 Halden, Norway
monica.kristiansen@hiof.no

RUNE WINTHER
Consultant at Risikokonsult
Oslo Area, Norway
rune.winther@wintherfamily.net

BENT NATVIG
Department of Mathematics
University of Oslo, Norway
bent@math.uio.no

Received (18 March 2011)

Predicting the reliability of software systems based on a component-based approach is inherently difficult, in particular due to failure dependencies between software components. One possible way to assess and include dependency aspects in software reliability models is to find upper bounds for probabilities that software components fail simultaneously and then include these into the reliability models. In earlier research, it has been shown that including partial dependency information may give substantial improvements in predicting the reliability of compound software compared to assuming independence between all software components. Furthermore, it has been shown that including dependencies between pairs of data-parallel components may give predictions close to the system's true reliability. In this paper, a Bayesian hypothesis testing approach for finding upper bounds for probabilities that pairs of software components fail simultaneously is described. This approach consists of two main steps: 1) establishing prior probability distributions for probabilities that pairs of software components fail simultaneously and 2) updating these prior probability distributions by performing statistical testing. In this paper, the focus is on the first step in the Bayesian hypothesis testing approach, and two possible procedures for establishing a prior probability distribution for the probability that a pair of software components fails simultaneously are proposed.

Keywords: Compound software; component dependencies; Bayesian hypothesis testing; expert judgment; prior information.

*Corresponding author.

1. Introduction

The problem of assessing reliability of software has been a research topic for more than 30 years, and several successful methods for predicting the reliability of an individual software component based on testing have been presented in Frankl et al. [13], Goel [14], Hamlet [19], Lyu [40], Miller et al. [42], Musa [44], Ramamoorthy and Bastani [51], and Voas and Miller [59]. However, there are still no methods fully successful for predicting reliability of compound software [a] based on reliability data on the system's individual software components [15,17,60].

1.1. *Motivation*

For hardware, even in critical systems, it is accepted to base the reliability assessment on failure statistics, i.e. to measure the failure probability of individual hardware components and then compute system reliability on this basis. This is applied for example in safety instrumented systems in petroleum [22]. However, the characteristics of software make it difficult to carry out such a reliability assessment. Software is not subject to aging and any failure that occurs during operation is due to faults that are inherent in the software from the beginning. Any randomness in software failure is due to randomness in input data. It is also a fact that environments such as hardware, operating system and user needs change over time and that software reliability may change over time due to these activities [9].

Furthermore, having a system consisting of several software components [b] explicitly requires an assessment of the software components' failure dependencies [c]. This is discussed more thoroughly in, among others, Cortellessa and Grassi [7], Dai et al. [10], Gokhale and Trivedi [16], Guo et al. [18], Littlewood et al. [38], Lyu [40], Nicola and Goyal [46], Popic et al. [49], Popov et al. [50], and Tomek et al. [56]. In addition to the fact that software reliability assessment is inherently difficult due to software complexity and that software is sensitive to changes in usage, failure dependencies between software components represent a substantial problem.

Although several different approaches to construct component-based software reliability models have been proposed in, among others, Cortellessa and Grassi [7], Gokhale and Trivedi [15], Gokhale [16], Goseva-Popstojanova and Trivedi [17], Hamlet [20,21], Krishnamurthy and Mathur [25], Krka et al. [32], Kuball et al. [33], Popic et al. [49], Reussner et al. [52], Singh et al. [54], Trung and Thang [57], Vieira and Richardson [58], and Yacoub et al. [61], most of these approaches tend to ignore failure dependencies between software components [11,24,36]. In principle, the failure probability of a single software component can be assessed through statistical testing [12,53]. However, since critical software components usually have low failure probabilities [38], in practice the number of tests required to obtain adequate confidence

[a]Software systems consisting of multiple software components.
[b]See Definition 1 in Subsection 1.2.
[c]See Definition 2 in Subsection 1.2.

in such probabilities becomes very large. An even more non-trivial situation arises when probabilities for simultaneous failures [d] of several software components need to be assessed, since they are likely to be significantly smaller than single failure probabilities.

Based on the fact that software components rarely fail independently and that statistical testing (for assessing the probability for software components failing simultaneously) is practically impossible, the main focus of our research has been to develop a practicable component-based approach for assessing reliability of compound software in which failure dependencies between software components are explicitly addressed [27,28,29,30,31].

One possible way to assess and include dependency aspects in software reliability models is to find upper bounds for probabilities that software components fail simultaneously and then include these into the reliability models. In Kristiansen et al. [29] it is shown that including partial dependency information may give substantial improvements in the reliability predictions of compound software compared to assuming independence between all software components. Furthermore, it is shown that including dependencies between pairs of data-parallel components [e] may give predictions close to the true system reliability. It is also shown that dependencies between pairs of data-parallel components are far more important than dependencies between pairs of data-serial components [f].

In this paper, the theory on how to apply Bayesian hypothesis testing [8,26,55] to find upper bounds for probabilities that pairs of software components fail simultaneously is described in detail. This approach can be divided into two main steps. In the first step, prior probability distributions for probabilities that pairs of software components fail simultaneously are established. In the second step, these distributions are updated by performing statistical testing. In this paper, two possible procedures for establishing a prior probability distribution for the probability that a pair of software components fails simultaneously are proposed.

1.2. *Definitions*

In this paper, the following definitions are used:

Definition 1. Software component: a software component (or module or unit) is considered to be an entity that has a predefined and specified boundary and which is atomic in the sense that it can not or will not be divided into sub-components. No special assumptions are made whether the component is available in binary format or as source code. The context is essentially an Off-The-Shelf (OTS) situation where custom-developed and PDS components [g] are combined to achieve a larger piece of software.

[d]See Definition 3 in Subsection 1.2.
[e]See Definition 4 in Subsection 1.2.
[f]See Definition 5 in Subsection 1.2.
[g]See Definition 6 in Subsection 1.2..

Definition 2. Failure dependency between software components: components are said to fail dependently if the knowledge that one software component has failed changes our belief whether or not another software component fails [60]. It is necessary to make a clear distinctions between the *degree of dependency* between software components (expressed through e.g. simultaneous failure probabilities) and the *mechanisms* that either cause or exclude software components to fail dependently (interface complexity, shared resources, programming language, development team, etc.)

Definition 3. Simultaneous failure: the event that more than one software component fails on the same system input. Component failures are not required to occur at the same instant, it is adequate that they are all in a failed state at some point in time.

Definition 4. 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 [29].

$$i \overset{d}{\nrightarrow} j \quad and \quad j \overset{d}{\nrightarrow} i \tag{1}$$

Definition 5. 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 [29].

$$i \overset{d}{\rightarrow} j \quad or \quad j \overset{d}{\rightarrow} i \tag{2}$$

Definition 6. Pre-developed software (PDS): software which already exists, is available as commercial or proprietary product and is being considered for use in a computer-based system [1]. This definition encompasses any kind of reuse of software whether it is black-box, commercially available, from an in-house library, or just happens to be available from another system.

1.3. *Assumptions*

In this paper, only on-demand types of situations are considered, i.e. situations where the system is given an input and execution is considered to be finished when a corresponding output has been produced. The following assumptions are made:

- The states of the software components are positively correlated.
- All data-flow relations between the software components are known.
- The reliabilities of the individual software components are known.
- The system and its components have only two possible states (functioning and failed).
- The system has a monotone structure [45].

The research has been inspired by the following basic problem. Assume a compound software, e.g. a fault tolerant system capable of switching between two redundant software components in case of failure. Furthermore, assume that the failure probabilities of the individual software components are available. Even in this ideal situation, it is difficult to compute the failure probability of the complete system since no information regarding the failure dependencies between the software components is available. In this paper, the main focus is to include dependency aspects in software reliability models by applying Bayesian hypothesis testing.

1.4. *Notation*

In this paper, capital letters are used to denote random variables and lower-case letters are used for their realizations.

To indicate the state of the ith component, a binary value x_i is assigned to component i [2].

$$x_i = \begin{cases} 0 \text{ if component } i \text{ is in the failed state} \\ 1 \text{ if component } i \text{ is in the functioning state} \end{cases} \tag{3}$$

Important notation used throughout this paper is listed in Table 1.

1.5. *The structure of this paper*

Section 2 summarizes some related work regarding the issue of failure dependency between software components. In Section 3, relevant background information needed to understand the Bayesian hypothesis testing approach and our overall approach for assessing reliability of compound software are described [27]. In Section 4, a Bayesian hypothesis testing approach for finding upper bounds for probabilities that pairs of software components fail simultaneously is illustrated. This approach consists of two main steps: 1) establishing prior probability distributions for probabilities that pairs of software components fail simultaneously and 2) updating these prior probability distributions by performing statistical testing. In Section 5, two possible procedures for establishing a prior probability distribution for the probability that a pair of of software components fails simultaneously are proposed. In addition, relevant information sources which may influence this probability distribution are presented. Section 6 summarizes the results, discusses the assumptions made and presents ideas for further work.

2. Related work

Most research and discussions on software component dependency over the years are mainly related to software design diversity, especially to N-version programming where output is decided by a voter using the results from N components as input. The idea behind N-version programming is that by forcing various aspects

Table 1. Notation.

Term		**Explanation**
n	=	number of tests
r	=	number of failures in n tests
θ	=	an unknown quantity, for example the probability that a pair of software components fails simultaneously
q_{ij}	=	the probability that a pair (i, j) of software components fails simultaneously
$q_{0,ij}$	=	an accepted upper bound for $q_{i,j}$
$\pi(\theta)$	=	a prior probability distribution for θ
$\pi(q_{ij})$	=	a prior probability distribution for q_{ij} defined in the sub-intervals $[a_{ij}, q_{0,ij}]$ and $[q_{0,ij}, b_{ij}]$
$g(q_{ij})$	=	a prior probability distribution for q_{ij} defined in the interval $[a_{ij}, b_{ij}]$
D	=	sample information
$\pi(\theta\|D)$	=	a posterior probability distribution for θ
$\pi(q_{ij}\|D)$	=	a posterior probability distribution for q_{ij}
$[a_{ij}, b_{ij}]$	=	interval where the simultaneous failure probability q_{ij} is defined
H_0	=	null hypothesis ($a_{ij} \leq q_{ij} \leq q_{0,ij}$)
H_1	=	alternative hypothesis ($q_{0,ij} < q_{ij} \leq b_{ij}$)
B	=	Bayes factor
$L(\theta\|D)$	=	likelihood function
π_0	=	prior belief in the null hypothesis ($P(H_0) = P(a_{ij} \leq q_{ij} \leq q_{0,ij})$)
π_1	=	prior belief in the alternative hypothesis ($P(H_1) = P(q_{0,ij} < q_{ij} \leq b_{ij})$)
α_0	=	posterior belief in the null hypothesis ($P(H_0\|D)$)
α_1	=	posterior belief in the alternative hypothesis ($P(H_1\|D)$)
$C_{0,ij}$	=	a given predefined confidence level for the simultaneous failure probability of a pair (i, j) of software components ($P(H_0\|D) \geq C_{0,ij}$)
$g_0(q_{ij})$	=	a probability distribution describing how the prior mass is spread out over the null hypothesis
$g_1(q_{ij})$	=	a probability distribution describing how the prior mass is spread out over the alternative hypothesis
α, β	=	parameters in the beta distribution

of the development process to be different, i.e. development team, methods, tools, programming languages, etc., the likelihood of having the same fault in several components would become negligible.

The hypothesis that independently developed components fail independently has been investigated from various perspectives. A direct test of this hypothesis was done in Knight and Leveson [24] where a total of 27 components were developed by different people. Although the results can be debated, this experiment indicated that assuming independence should be done with caution. The experiment showed that the number of tests for which several components failed was much higher than anticipated under the assumption of independence. While there are many different mechanisms that might cause even independently developed components to fail on the same inputs, it does not seem implausible that the simple fact that programmers are likely to approach a problem in much the same way causes them to make the same mistakes and thus generates dependency between the components' failure behavior.

A more theoretical approach on the same issue was presented in Eckhardt and Lee [11] and elaborated on a few years later in Littlewood and Miller [36]. A more comprehensive discussion is provided in Littlewood et al. [38].

In the Eckhardt and Lee model (EL-model) there are two basic sources of uncertainty: 1) the random selection of input from the space of all inputs and 2) the random creation of a component version from the population of all possible component versions that can be written. The key variable in the model is the difficulty function $\phi(x)$ defined to be the probability that a component version chosen at random fails on a particular input demand x. In other words, if many component versions are selected independently, the difficulty function is the proportion of these that fail on a particular input. The idea of the Eckhardt and Lee model is that the difficulty function generally takes different values for different inputs, representing the varying difficulty in correctly processing different inputs. The more difficult an input x is, the greater is the chance that an unknown component version fails.

The main result in the EL-model is that independently developed component versions do not imply independent component versions. The key point is that as long as some inputs are more difficult to process than others, even independently developed component versions fail dependently. In fact, the more the difficulty varies between the inputs, the greater is the dependence in failure behavior between component versions. Only in the special situation where all inputs are equally difficult, i.e. the difficulty function $\phi(x)$ is constant for all $x \in \Omega$, independently developed component versions fail independently.

The Littlewood and Miller model (LM-model) is a generalization of the EL-model in which different component versions are developed using diverse methodologies. In this context, the different development methodologies might represent different development environments, different types of programmers, different languages, different testing regimes, etc. Thus, a component version has a different probability of being developed under methodology A than under methodology B. If the methodologies are very diverse, one expects a component version with a high probability of being developed under one methodology to have a low probability of being developed under another.

The main result of the LM-model is that the use of diverse methodologies decreases the probability of simultaneous failure of several component versions. In fact, they show that it is theoretically possible to obtain component versions which exhibit better than independent failure behavior. So while it is natural to try to justify an assumption of independence, it is worthwhile noticing that having independent component versions is not necessarily the optimal situation with regard to maximizing reliability.

The main problem with both the EL-model and the LM-model is that the predictions of these models are predictions for an "average" multi-version development. These models consider all possible software realizations from the same specification and say nothing about a particular pair of component versions. Thus, actual realizations may be different from these averages.

In Popov et al. [50], the authors extend the previous conceptual models proposed in Eckhardt and Lee [11] and Littlewood and Miller [36] and address the problem of assessing the reliability of a specific set of component versions. All their models refer to the simplest possible diverse-redundant configuration, i.e. two component versions (A and B) with perfect adjudication (1-out-of-2 system). This means that the system behaves correctly provided that either A or B behaves correctly. In Popov et al. [50], the upper bounds are based on knowledge on the input space of the component versions. For each sub-domain S_i $(i = 1, \ldots, n)$, the authors assume that the probability $P(S_i)$ of drawing a random input from S_i is known. Furthermore, they assume that the failure probabilities of component versions A and B for inputs from each sub-domain S_i ($P_{A|S_i}$ and $P_{B|S_i}$) are known.

If it is assumed that component versions fail independently within each sub-domain, the probability of simultaneous failure between component versions on a random input can be calculated directly from the known probabilities, i.e. $P_{A,Bsub-ind} = \sum_i P_{A|S_i} P_{B|S_i} P(S_i)$. This estimate is an intermediate value between the true failure probability $(P_{A,B})$ and the failure probability one gets when assuming independence between the two component versions, i.e. $P_A P_B$. In fact, assuming conditional independence of failures within sub-domains is therefore less optimistic than assuming unconditional independence of the whole input space.

In Littlewood et al. [37], the problem of assessing reliability of a 1-out-of-2 system is also considered. To do this, the authors apply Bayesian inference which includes establishing a prior belief regarding the parameters of interest and then updating these by using Bayes theorem when “hard” evidence becomes available. assuming that A and B are two diverse components, calculation of the system reliability requires that the probabilities $P_A = P(A \text{ fails})$, $P_B = P(B \text{ fails})$ and $P_{AB} = P(\text{Both } A \text{ and } B \text{ fails})$ are determined. This means that the degree of dependence between software components is implicitly addressed through P_{AB}.

In the most general case, the authors' propose that a 3-dimensional prior must be determined for the three probabilities which is conceptually and practicably very hard. In the less difficult situation where both P_A and P_B are assumed to be known, the authors run into a paradox where the posterior probability of system failure becomes higher than the prior probability when no component failures are observed during testing. The reason for this paradox is that the probability of having no failure in any of the components, i.e. $1 - P_A - P_B + P_{AB}$ is large if the probability P_{AB} is also large. Hence, in practice it seems that neither the "full" approach where all the relevant probabilities are handled simultaneously, nor the simplified approach where two out of three probabilities are assumed to be known work.

While the work of Littlewood et al. [37] focuses on all three failure probabilities P_A, P_B and P_{AB}, we solely focus on P_{AB} and suggest a Bayesian hypothesis testing approach to find upper bounds for probabilities that pairs of software components fail simultaneously. This approach uses all relevant information which is available prior to testing and consists of the following main steps:

1. Establishing prior probability distributions for probabilities that pairs of software components fail simultaneously based on all relevant information available prior to testing (Section 5).
2. Updating these prior probability distributions by performing statistical testing (Section 4).

3. Theoretical background

In this section, relevant background information on statistical testing and Bayesian analysis is described in detail. In addition, it is shown how failure probabilities of individual components and the assumption of positive correlation put direct restrictions on the components' simultaneous failure probabilities. All this information is needed to understand the Bayesian hypothesis testing approach for finding upper bounds for probabilities that pairs of software components fail simultaneously. At last, our approach for assessing the reliability of compound software is described in detail [27]. In this approach, failure dependencies between pairs of software components are explicitly addressed by applying Bayesian hypothesis testing on simultaneous failure probabilities.

3.1. *Statistical testing*

Statistical testing [12,53] consists of exposing a piece of software to test cases drawn randomly according to some probability distribution defined over the program's input space. Such testing can be used to assess a software component's failure probability. Typical assumptions in statistical testing are: i) independent test runs, ii) constant failure rate, iii) all failures during testing are detected and iv) the operational profile is known.

One benefit of statistical testing is that it requires no knowledge of the internal structure of the software components being tested. This is of great importance when PDS components [h] are used, for which one might not have all the required information available.

Let p_0 denote the accepted upper failure probability. The number of fault free tests n which must be carried out to satisfy the failure probability p_0 at a given confidence level C_0 using classical statistical testing is given in Equation 4 [48].

$$n = \frac{ln(1 - C_0)}{ln(1 - p_0)} \tag{4}$$

[h]See Definition 6 in Section 1.2.

3.2. *Bayesian analysis*

Bayesian analysis [3] consists of combining prior information $\pi(\theta)$ and sample information D into a posterior distribution $\pi(\theta|D)$ for θ given D. It is from this posterior distribution all decisions and inferences are made in Bayesian analysis. Bayes theorem [3] is expressed in Equation 5 where the prior distribution $\pi(\theta)$ reflects beliefs about θ prior to testing and the posterior distribution $\pi(\theta|D)$ reflects updated beliefs about θ after testing. $L(\theta|D)$ is the likelihood function which expresses the likelihood of θ given sample information D.

$$\pi(\theta|D) = \frac{L(\theta|D)\pi(\theta)}{\int_{\Theta} L(\theta|D)\pi(\theta)d\theta} \tag{5}$$

In hypothesis testing, a null hypothesis (H_0) and an alternative hypothesis (H_1) are specified. In classical statistics, one decides between H_0 and H_1 by examining type I and type II error probabilities. These probabilities of error represent the chance that for an observed sample the test procedure results in the wrong hypothesis being accepted. Type I error occurs when H_0 is rejected when it is true and type II error occurs when H_0 is accepted when it is false. In Bayesian analysis, hypothesis testing is conceptually more straightforward. One calculates the posterior probabilities $\alpha_0 = P(H_0|D)$ and $\alpha_1 = P(H_1|D)$ which combine both test data and prior knowledge and then decide between H_0 and H_1 accordingly [3]. Often it is convenient to summaries the evidence in term of posterior odds. Saying that $\alpha_0/\alpha_1 > R$ clearly says that H_0 is R times as likely to be true as H_1. Although the posterior probabilities of the hypotheses are the primary measures in Bayesian hypothesis testing, the prior probabilities $\pi_0 = P(H_0)$ and $\pi_1 = P(H_1)$ are also of interest. The ratio π_0/π_1 is called the prior odds ratio and the Bayes factor can be expressed by combining the prior and posterior odds ratios (see Equation 6).

$$B = \frac{\alpha_0/\alpha_1}{\pi_0/\pi_1} = \frac{\alpha_0\pi_1}{\alpha_1\pi_0} \tag{6}$$

The Bayes factor is the odds ratio for H_0 to H_1 that is given by the data [3]. If the Bayes factor is greater than one, data helped increasing odds in favor of the null hypothesis [8]. The Bayes factor forms the basis for finding the number of tests required to satisfy a predefined upper bound $q_{0,ij}$ at confidence level $C_{0,ij}$ in the proposed approach for finding upper bounds for probabilities that pairs of software components fail simultaneously. This approach is elaborated in detail in Section 4.

3.3. *Prior Information from the Software Components' failure probabilities*

In this section, it is illustrated how the software components' marginal failure probabilities put direct restrictions on the components' simultaneous failure probabilities. Let q_i and q_j denote the failure probability of components i and j, respectively, and let q_{ij} denote the simultaneous failure probability of components i and j. If it can be assumed that the components' failure probabilities do not change due to changes in operational context, the following is true:

$$q_{ij} \leq min(q_i, q_j), \tag{7}$$

which follows directly from the fact that:

$$\begin{aligned} q_{ij} &= P(X_i = 0 \cap X_j = 0) = P(X_i = 0)P(X_j = 0|X_i = 0) \\ &= P(X_j = 0)P(X_i = 0|X_j = 0). \end{aligned} \tag{8}$$

In general, one would expect positive correlation between components i and j since some inputs are more difficult (more error-prone) than others [38]. Even if diverse components are produced "independently", failures are more likely to happen on certain inputs than others. This means that if component i fails, the failure probability of component j will also increase. However, when components are in parallel this may not always be a reasonable assumption. If components have been developed by different development teams and by using different development methods and languages, it might in fact be natural to assume negative correlation. This means that if one component fails, the failure probability of the other component decreases and visa versa. However, assuming positive correlation is far more conservative than assuming independence between software components when it comes to predicting system's reliability. It follows that:

$$q_{ij} \geq q_i q_j. \tag{9}$$

Reasonable constraints on the simultaneous failure probability q_{ij} under the given assumptions can therefore be expressed as follows:

$$q_i q_j \leq q_{ij} \leq min(q_i, q_j). \tag{10}$$

From Equation 10, it can be clearly seen that information on the software components' marginal failure probabilities can be used directly to specify the upper and lower limit for the simultaneous failure probability q_{ij}. In the following, these limits will be used to:

- specify the hypotheses in the component-based approach (Subsection 3.4 and Section 4).
- specify a starting point when establishing a prior probability distribution for the simultaneous failure probability q_{ij} (Section 5).

The topic on how the reliabilities of individual software components put direct restrictions on the components' conditional reliabilities is elaborated in more detail in Kristiansen et al. [29]. In this paper, the authors present restrictions on the conditional reliabilities in general systems consisting of two and three components.

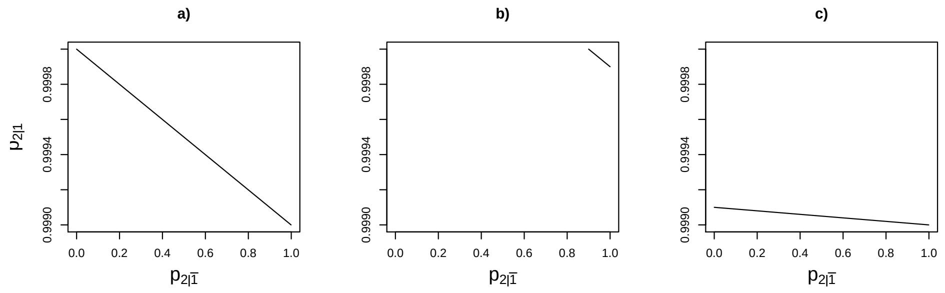

Fig. 1. Possible values for the conditional reliabilities in a two components system when a) $p_1 = 0.999$ and $p_2 = 0.999$, b) $p_1 = 0.999$ and $p_2 = 0.9999$ and c) $p_1 = 0.9999$ and $p_2 = 0.999$.

Examples of how the marginal reliabilities p_1 and p_2 influence the conditional reliabilities $p_{2|1}$ and $p_{2|\bar{1}}$ in a general two components system are illustrated in Figure 1. The graphs clearly show that the restrictions on the conditional reliabilities depend heavily on the values of the marginal reliabilities. In fact, in some cases the conditional reliabilities are restricted into narrow intervals.

In the same way, it is shown how the marginal reliabilities p_1, p_2, and p_3 influence the conditional reliabilities $p_{2|1}$, $p_{2|\bar{1}}$, $p_{3|1}$, $p_{3|\bar{1}}$, $p_{3|2}$, $p_{3|\bar{2}}$, $p_{3|12}$ and $p_{3|\bar{1}\bar{2}}$ in a general three components system. An example is illustrated in Table 2 which should be read

Table 2. Restrictions on the conditional reliabilities $p_{2|1}$, $p_{3|1}$, $p_{3|2}$ and $p_{3|12}$ in a simple three components system when $p_1 = 0.9999$, $p_2 = 0.999$ and $p_3 = 0.99$.

Example 1	
First assumption:	
$p_1 = 0.9999$	
$p_2 = 0.999$	
$p_3 = 0.99$	
Results in:	
$p_{2|1} \in [0.999, 0.9990999]$	$p_{2|\bar{1}} \in [0, 0.999]$
$p_{3|1} \in [0.99, 0.990099]$	$p_{3|\bar{1}} \in [0, 0.99]$
$p_{3|2} \in [0.99, 0.99099099]$	$p_{3|\bar{2}} \in [0, 0.99]$
$p_{3|12} \in [0.99, 0.99099999]$	$p_{3|\bar{1}\bar{2}} \in [0, 0.99]$
Second assumption:	
$p_{2|1} = 0.99905$	
$p_{3|1} = 0.990085$	
Results in:	
$p_{3|2} \in [0.990043, 0.990964]$	$p_{3|\bar{2}} \in [0.026468, 0.947503]$
$p_{3|12} \in [0.990085, 0.990999]$	$p_{3|\bar{1}\bar{2}} \in [0, 0.140085]$
Third assumption:	
$p_{3|2} = 0.9903$	
Results in:	
$p_{3|12} \in [0.990336, 0.990342]$	$p_{3|\bar{1}\bar{2}} \in [0, 0.140085]$

as follows:

- In the first assumption it is assumed that the marginal reliabilities of the components are known. Knowing these reliabilities puts direct restrictions on all the remaining conditional reliabilities. In some cases they limit the conditional reliabilities into small intervals.
- In the second assumption it is assumed that the conditional reliabilities $p_{2|1}$ and $p_{3|1}$ are known in addition to the marginal reliabilities. This puts even more strict restrictions on the remaining conditional reliabilities $p_{3|2}$, $p_{3|\bar{2}}$, $p_{3|12}$ and $p_{3|\bar{1}\bar{2}}$.
- In the third assumption the conditional reliability $p_{3|2}$ is also assumed to be known and it can be easily seen that the more information that is available, the more strict are the restrictions on the remaining reliabilities $p_{3|12}$ and $p_{3|\bar{1}\bar{2}}$.

3.4. *A component-based approach for assessing the reliability of compound software*

The main focus of our research has been to develop a component-based approach for assessing reliability of compound software where failure dependencies between software components are addressed explicitly [27]. One possible way to assess and include dependency aspects in software reliability models is to find upper bounds for probabilities that pairs of software components fail simultaneously and then include these into the reliability models. To find these upper bounds, the suggested approach applies Bayesian hypothesis testing [8,26,55] on simultaneous failure probabilities (see Section 4 for details). It is assumed that failure probabilities of individual software components are known. The approach is illustrated in Figure 2 and consists of five basic steps.

1. *Identify the most important component failure 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 [29]. 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 fail 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:

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

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

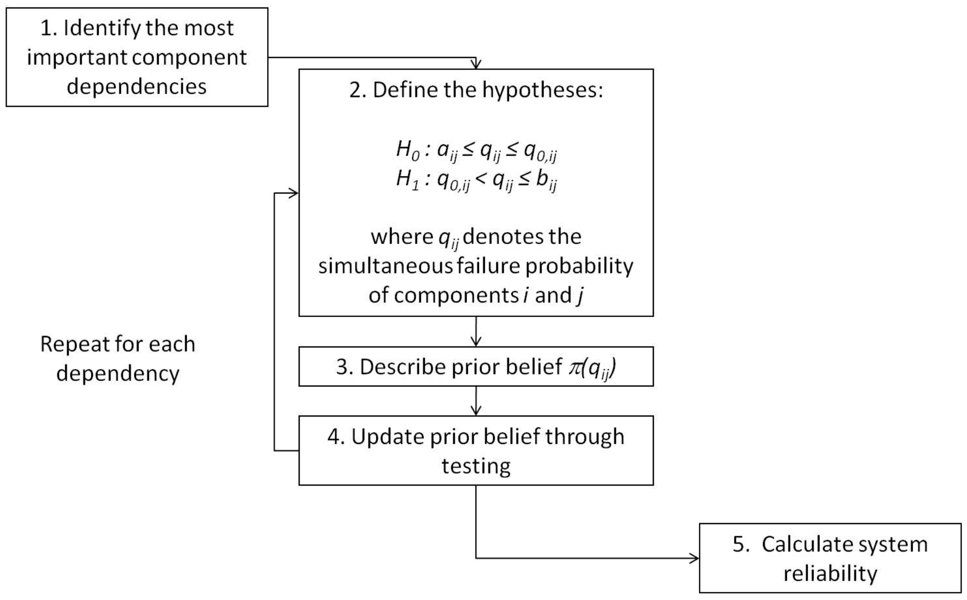
1. Identify the most important component dependencies
2. Define the hypotheses:
$H_0 : a_{ij} \le q_{ij} \le q_{0,ij}$
$H_1 : q_{0,ij} < q_{ij} \le b_{ij}$
where q_{ij} denotes the simultaneous failure probability of components i and j
Repeat for each dependency
3. Describe prior belief $\pi(q_{ij})$
4. Update prior belief through testing
5. Calculate system reliability

Fig. 2. A component-based approach for assessing the reliability of compound software.

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 that the components' marginal failure probabilities put on the components' simultaneous failure probabilities (Subsection 3.3).

3. *Describe prior belief regarding probability* q_{ij}: establish a prior probability distribution $\pi(q_{ij})$ for the probability that a pair of software components fail simultaneously. Based on this probability distribution the prior belief in the null hypothesis $(P(H_0))$ must be quantified.
4. *Update your belief in hypothesis* H_0: 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 r 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 Section 4 or Cukic et al. [8].
5. *Calculate the complete system's failure probability*: information regarding individual software components' failure probabilities (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.

Step 1 in the component-based approach has been discussed in Kristiansen et al. [29]. In that paper, a test system consisting of five components is investigated to identify possible rules for selecting the most important component dependencies. To do this, three different techniques are applied: 1) direct calculation, 2) Birnbaum's importance measure of a component and 3) Principal Component Analysis (PCA). The results from the analyses clearly show that including partial dependency information may give substantial improvements in the reliability predictions compared to assuming independence between all software components. However, this is only true as long as the most important component dependencies are included in the reliability calculations. It is also apparent that dependencies between pairs of data-parallel components are far more important than dependencies between pairs of data-serial components. Furthermore, the analyses indicate that including only dependencies between pairs of data-parallel components may give predictions close to the system's true failure probability as long as the dependency between the most unreliable components is included. Including only dependencies between pairs of data-serial components may however result in predictions even worse than by assuming independence between all software components.

Step 3 in the component-based approach has been discussed in Kristiansen et al. [31]. In that paper, the results from an experimental study which investigates the relations between a set of internal software metrics (McCabe's cyclomatic complexity, Halstead metrics, Source Lines of Code etc.) and stochastic failure dependency between software components are presented. The experiment was performed by analyzing a large collection of program versions submitted to the same specification in a programming competition on the Internet: the Online judge [i]. In the study, pairs of program versions were investigated. To measure the probability that a pair of program versions fails dependently, the study used the simultaneous failure probability of the program versions. If any relations between the probabilities that pairs of software components fail simultaneously and their difference in software metrics can be identified, one possible step forward will be to use this information as prior information in the Bayesian hypothesis testing approach for finding upper bounds for simultaneous failures between pairs of software components. Results from univariate analyses show that if the difference between metric values of two program versions is small, it is impossible to decide the degree of failure dependency between those two program versions. However, given that the metric values for a pair of program versions differ significantly and the program versions are reasonable mature, results indicate that the probability for simultaneous failures is less than the probability calculated if the metric values were similar.

In addition, a simulator which explicitly accounts for failure dependencies between the software components and can be used to calculate the complete system's failure probability when direct calculation becomes too difficult has been devel-

[i] *http://icpcres.ecs.baylor.edu/onlinejudge*

oped. This simulator can be used in step 5 in the component-based approach and is described in Kristiansen et al. [30].

The rest of the present paper focuses on steps 3 and 4 in the component-based approach and describes a Bayesian hypothesis testing approach for finding upper bounds for probabilities that pairs of software components fail simultaneously. First, the theory behind this approach is described in detail in Section 4. Then two procedures for establishing a prior probability distribution for the probability that a pair of software components fails simultaneously are described in Section 5.

4. Bayesian hypothesis testing applied on simultaneous failure probabilities of pairs of software components

One possible approach to find upper bounds for probabilities that pairs of software components fail simultaneously is to use Bayesian hypothesis testing [8,26,27,55].

Assume that H_0 and H_1 are defined as in step 2 in Section 3.4 where q_{ij} is a probability in the interval $[a_{ij}, b_{ij}]$. Here q_{ij} represents the probability that a pair (i, j) of software components fails simultaneously. 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 as described in Section 3.3 (for more information see Kristiansen et al. [29]). In this case, the null and alternative hypotheses state that the probability for a pair of software components to fail simultaneously is lower and higher than the given upper bound $q_{0,ij}$, respectively.

Notice that the hypotheses are defined the opposite way of classical statistical hypothesis testing. In classical hypothesis testing, the alternative hypothesis usually expresses what one wishes to confirm, i.e. that the probability for simultaneous failure is less than a predefined upper bound $q_{0,ij}$, whereas the null hypothesis is the most conservative and expresses the opposite. In this way, the doubt benefits the null hypothesis which is true until the opposite is proved. However, by looking at the mathematics in the Bayesian hypothesis testing approach it can be easily seen that it does not matter which way the hypotheses are defined.

To express the prior belief in the simultaneous failure probability q_{ij}, two separate probability distributions over the intervals $[a_{ij}, q_{0,ij}]$ and $(q_{0,ij}, b_{ij}]$ can be used [3]. This is expressed in Equation 11.

$$\pi(q_{ij}) = \begin{cases} P(H_0)g_0(q_{ij}) \; a_{ij} \leq q_{ij} \leq q_{0,ij} \\ P(H_1)g_1(q_{ij}) \; q_{0,ij} < q_{ij} \leq b_{ij} \end{cases} \tag{11}$$

where $g_0(q_{ij})$ and $g_1(q_{ij})$ are proper probability density functions ($g_0(q_{ij}) > 0$, $\int_{a_{ij}}^{q_{0,ij}} g_0(q_{ij})dq_{ij} = 1$, $g_1(q_{ij}) > 0$, $\int_{q_{0,ij}}^{b_{ij}} g_1(q_{ij})dq_{ij} = 1$) which describe how the prior mass is spread out over the two hypotheses.

Let $g(q_{ij})$ be the prior probability distribution for q_{ij} in the interval $[a_{ij}, b_{ij}]$. The probability density functions $g_0(q_{ij})$ and $g_1(q_{ij})$ can then be defined as shown in Equations 12 and 13.

$$g_0(q_{ij}) = \frac{1}{P(H_0)} g_(q_{ij}) \qquad a_{ij} \leq q_{ij} \leq q_{0,ij} \tag{12}$$

$$g_1(q_{ij}) = \frac{1}{P(H_1)} g_(q_{ij}) \qquad q_{0,ij} < q_{ij} \leq b_{ij} \tag{13}$$

The probability distribution for observing r simultaneous failures during testing given n independent trials and a constant simultaneous failure probability q_{ij} can be expressed by the binomial probability distribution. This is shown in Equation 14.

$$f(r|q_{ij}, n) = \binom{n}{r} q_{ij}^r (1 - q_{ij})^{n-r} \tag{14}$$

The posterior belief in the null hypothesis H_0 is the probability of the null hypothesis in light of data and prior knowledge. For acceptance, this probability must be higher than a given predefined confidence level $C_{0,ij}$.

$$P(H_0|r, n) \geq C_{0,ij} \tag{15}$$

The number of tests n required to satisfy the confidence level $C_{0,ij}$ for different numbers of simultaneous failures encountered r during testing can be found by using Bayes factor. Based on Equations 11 and 14, the posterior odds ratio can be expressed as shown in Equation 16.

$$\frac{\alpha_0}{\alpha_1} = \frac{P(H_0|r, n)}{P(H_1|r, n)} = \frac{\int_{a_{ij}}^{q_{0,ij}} f(r|q_{ij}, n) P(H_0) g_0(q_{ij}) dq_{ij}}{\int_{q_{0,ij}}^{b_{ij}} f(r|q_{ij}, n) P(H_1) g_1(q_{ij}) dq_{ij}} \tag{16}$$

Further, it can easily be shown that the Bayes factor given by Equation 6 can be written as the weighted likelihood ratio as shown in Equation 17.

$$B = \frac{\int_{a_{ij}}^{q_{0,ij}} f(r|q_{ij}, n) g_0(q_{ij}) dq_{ij}}{\int_{q_{0,ij}}^{b_{ij}} f(r|q_{ij}, n) g_1(q_{ij}) dq_{ij}} \tag{17}$$

Based on the acceptance criterion in Equation 15, it can be shown that the Bayes factor given in Equation 6 must satisfy Equation 18 [8].

$$B \geq \frac{C_{0,ij} P(H_1)}{(1 - C_{0,ij}) P(H_0)} \tag{18}$$

The number of tests n required to obtain an adequate upper bound for the probability that a pair of software components fails simultaneously for different numbers of simultaneous failures r encountered during testing can be found by solving Equation 19, which is based on Equations 17 and 18.

$$\frac{\int_{a_{ij}}^{q_{0,ij}} f(r|q_{ij},n)g_0(q_{ij})dq_{ij}}{\int_{q_{0,ij}}^{b_{ij}} f(r|q_{ij},n)g_1(q_{ij})dq_{ij}} = \frac{C_{0,ij}P(H_1)}{(1-C_{0,ij})P(H_0)} \tag{19}$$

Equation 19 can be solved numerically by simply programming a loop which stops and returns the number of tests when the left side exceeds the right side in the equation. The integrals must be solved by using numerical integration.

5. Establishing prior probability distributions for probabilities that pairs of software components fail simultaneously

Before testing can be performed in the Bayesian hypothesis testing approach described in Section 4, a prior probability distribution $g(q_{ij})$ for the simultaneous failure probability q_{ij} defined in the interval $[a_{ij}, b_{ij}]$ must be established. This distribution is needed for establishing $g_0(q_{ij})$ and $g_1(q_{ij})$ in Equations 12 and 13 and for calculating $P(H_0)$ and $P(H_1)$.

The main motivation for establishing a prior probability distribution for q_{ij} is to utilize all relevant information sources available prior to testing in order to compensate for the enormous number of tests which is usually required to satisfy a predefined confidence level $C_{0,ij}$. In the case where reasonable prior information is available, the number of tests which must be run to achieve $C_{0,ij}$ in Equation 15 can be greatly reduced.

In the following, two procedures for establishing a prior probability distribution $g(q_{ij})$ for the simultaneous failure probability q_{ij} are proposed. Both procedures consist of two main steps, the first step being common for both of them.

1. Establish a starting point for q_{ij} based on a transformed beta distribution.
2. Adjust this starting point up or down by applying expert judgment on relevant information sources available prior to testing.

In the first procedure, the prior probability distribution for q_{ij} is determined by letting experts adjust the initial mean and variance of q_{ij} in the transformed beta distribution based on relevant information sources. In the second procedure, the prior transformed beta distribution for q_{ij} is adjusted numerically by letting experts express their belief in the total number of tests and the number of simultaneous failures that all relevant information sources correspond to. A combination of these two procedures can also be applied, since it might be easier for experts to adjust the mean and variance for some information sources, whereas for other information sources it might be easier to express belief about the total number of tests and the number of simultaneous failures.

The challenge of using expert judgment for decision making and how to calibrate experts has been addressed in several books and papers, among others Clemen and Lichtendahl [4], Cooke [5], Cooke and Goossens [6], Lin and Bier [35], Meyer and Booker [41] and Mosleh et al. [43]. There is also a lot of research covering the psychological biases encountered by using expert judgment, among others Kahneman et al. [23]

and Plous [47].

In the following subsection, ideas on how to establish a starting point for the simultaneous failure probability q_{ij} are described. Then, two procedures for adjusting this starting point up or down based on relevant information sources and expert judgment are described in detail. Finally, a set of available information sources relevant for adjusting the starting point for q_{ij} is presented.

5.1. *Establishing a starting point*

As a basis for establishing a starting point for the simultaneous failure probability q_{ij}, the following two assumptions are made:

1. The individual software components' failure probabilities are available.
2. The components are positively correlated.

Additionally, if it can be assumed that the failure probabilities of the individual components do not change due to changes in the operational context, reasonable constraints on the simultaneous failure probability q_{ij} can be found as shown in Equation 10 in Section 3.3. Let $a_{ij} = q_i q_j$ and $b_{ij} = min(q_i, q_j)$ denote the lower and upper limit for the simultaneous failure probability q_{ij}, respectively. Under the assumption that the failure probabilities of the individual components do not change due to changes in the operational context, one possible way to identify the initial values of the mean and variance of q_{ij} is to assume that q_{ij} is uniformally distributed (beta distributed with parameters $\alpha = \beta = 1$ over the interval $[a_{ij}, b_{ij}]$). The initial mean (μ_I) and variance (σ_I^2) are then given by the following two equations:

$$\mu_I = \frac{a_{ij} + b_{ij}}{2} \tag{20}$$

$$\sigma_I^2 = \frac{(b_{ij} - a_{ij})^2}{12} \tag{21}$$

However, since changes in the operational context are likely to change the failure probability of the individual components, a more conservative starting point is to use $b_{ij} = min(q_i, q_j)$ as the initial mean. A possible set of initial values of the mean and variance of q_{ij} can therefore be given by: $\mu_I = b_{ij}$ and $\sigma_I^2 = (b_{ij} - a_{ij})^2/12$. It is important to notice that these initial values are only a starting point for describing the simultaneous failure probability q_{ij}. These values and accordingly the resulting transformed beta distribution describing q_{ij} are adjusted by applying expert judgment on relevant information sources available prior to testing.

When the initial values of the mean and variance of q_{ij} in the transformed beta distribution are known, the initial values of the parameters α_I and β_I in the standard beta distribution can be found directly by applying linear transformation. This is shown in Equations 22 and 23.

$$\mu_I = a_{ij} + (b_{ij} - a_{ij})\frac{\alpha_I}{\alpha_I + \beta_I} \tag{22}$$

$$\sigma_I^2 = (b_{ij} - a_{ij})^2 \frac{\alpha_I \beta_I}{(\alpha_I + \beta_I)^2(\alpha_I + \beta_I + 1)} \tag{23}$$

Another possible way to identify initial values of α and β in the standard beta distribution is to visualize a "fictive" experiment in which n_I is the total number of experiments and α_I is the number of simultaneous failures of components i and j. The simultaneous failure probability q_{ij} can then be assumed to have a transformed beta distribution with parameters α_I and $\beta_I = n_I - \alpha_I$ defined in the interval $[a_{ij}, b_{ij}]$.

In addition to the the individual software components' failure probabilities, information regarding components' architecture, complexity, programming languages, development processes, etc. might as well be available. This information is also relevant for assessing q_{ij} and can be used to adjust its starting point up or down by applying expert judgment. In the following two subsections, two procedures for adjusting the starting point for q_{ij} up or down based on relevant information sources and expert judgment are described.

5.2. *Establishing a prior probability distribution for q_{ij} by adjusting its mean and variance*

In the following section, a procedure on how relevant information sources can be used to adjust the initial mean μ_I and variance σ_I^2 of the simultaneous failure probability q_{ij} in the transformed beta distribution is described. To illustrate this procedure, only two information sources are considered.

Define the following two information sources:

I_1 : Degree of complexity of the interface between the components.

I_2 : Degree of similarity between the programming languages of components i and j.

Assume that both information sources can be assigned values in the interval $[0, 1]$, such that a value close to 0 indicates low complexity (substantial difference in programming language) and a value close to 1 indicates extreme complexity (identical programming languages). In addition, let a value of 0.5 describe the "typical" situation for the components under consideration.

Based on these two information sources, the following interpretations seem plausible:

1. I_1 and I_2 are both close to 0: the mean and variance of the prior distribution are lower than μ_I and ${\sigma_I}^2$, respectively.
2. I_1 and I_2 are both close to 1: the mean of the prior distribution is larger than μ_I, while the variance is smaller than ${\sigma_I}^2$.
3. I_1 is close to 0 and I_2 is close to 1: the mean is dependent on the relative importance of I_1 and I_2, while the variance is larger than ${\sigma_I}^2$.

4. I_1 is close to 0.5 and I_2 is close to 0: the mean of the prior distribution is lower than μ_I, and the variance is close to ${\sigma_I}^2$.

In order to establish a prior probability distribution for q_{ij}, a functional relationship between the initial values defined by μ_I, ${\sigma_I}^2$, all relevant information sources and the parameters α and β in the standard beta distribution must be defined.

Let $I_i \in [0,1]$ denote information source i where $i = 1,\ldots,r$, and let $k_i \in [0,1]$ express the relative importance of each information source. Furthermore, let μ denote the adjusted mean of q_{ij} and σ^2 the adjusted variance in the transformed beta distribution. A simplistic way to update the mean and variance using the information sources might then be as follows:

$$\mu = \frac{2\mu_I(k_1I_1 + k_2I_2 + \cdots + k_rI_r)}{k_1 + k_2 + \cdots + k_r} \tag{24}$$

$$\sigma^2 = {\sigma_I}^2 kVar(I), \tag{25}$$

where $Var(I)$ is the sample variance calculated over the information sources I_i, and k is an impact factor that defines the impact the differences in the I_i values should have. The updated variance σ^2 increases if there are large differences between the I_i's (i.e. they give divergent information) and decreases if the differences between the I_i's are small (i.e. they give similar information). Note that if $I_i = 0.5$ for $i = 1,\ldots,n$, then $\mu = \mu_I$.

In the above procedure, both the I_i's, k_i's and k must be determined by experts. Then when the updated values of the mean and variance are found, the parameters α and β in the standard beta distribution can be found by solving Equations 22 and 23 by replacing α_I by α, β_I by β, μ_I by μ and σ_I^2 by σ^2.

5.3. *Establishing a prior probability distribution for q_{ij} by adjusting it numerically*

Based on the starting point established in Section 5.1, let us assume that the simultaneous failure probability q_{ij} can be expressed by a transformed beta distribution with parameters α_I and β_I. In order to adjust this probability distribution, a functional relationship between the initial parameters defined by α_I and β_I and all relevant information sources must be defined.

As in Section 5.2, let I_i denote a relevant information source i and let k_i express the relative importance of this information source. Further, let n_0 and α_0 represent the total number of tests and the number of simultaneous failures of components i and j that all relevant information sources correspond to, respectively.

A possible way to find α_0 using all relevant information sources is expressed in Equation 26.

$$\alpha_0 = \frac{n_0(k_1I_1 + \cdots + k_rI_r)}{k_1 + \cdots + k_r} \tag{26}$$

From this equation, it can be easily seen that the larger (closer to 1) the values of the relevant information sources I_i are, the larger is the number of simultaneous failures of components i and j.

The information achieved from all relevant information sources correspond to a binomial probability distribution with parameters α_0 and n_0. Since the transformed beta distribution with initial parameters α_I and β_I defined in the interval $[a_{ij}, b_{ij}]$ lacks the pleasant property of being a natural conjugate prior to the binomial distribution, the prior probability distribution for the simultaneous failure probability q_{ij} must be adjusted numerically. In the above procedure the I_i's, k_i's and n_0 must be determined by experts.

5.4. *Relevant information sources used to adjust the starting point*

In the following section, different types of information sources relevant for adjusting the starting point for the simultaneous failure probability q_{ij} are discussed.

Mechanisms that cause software components to fail simultaneously can be split into two distinct categories [60]:

- Development-cultural aspects (DC-aspects): mechanisms which cause different people, tools, methods, etc. to make the same mistakes.
- Structural aspects (S-aspects): mechanisms which allow a failure in one component to affect the execution of another component.

The first category can typically be assessed using component specific information sources $\mathcal{K}_i$. On the other hand, the second category cannot be completely assessed using only component specific information. Information sources on how the components are used in a specific context or in the compound software ($\mathcal{K}_{compound}$) is also needed. While $\mathcal{K}_i$ might be viewed as generic information, meaning that it does not change when the use of the component changes, $\mathcal{K}_{compound}$ is completely context-specific.

To successfully adjust the starting point for q_{ij}, the following information sources must be considered by experts [9,34,62]:

- The parts of a software component's pedigree that are relevant to be compared with another software component's pedigree, i.e. specific elements of $\mathcal{K}_i$ for all components i. This may include:
  - Development methodology.
  - Programming language.
  - Development team.
  - Specification.
  - Producer's reputation.
  - Software metrics (McCabe's cyclomatic complexity, Halstead complexity measures, program depth, source lines of code (SLOC), number of comments and comment lines, number of loops and statements).
  - Development tools (e.g. compiler).
  - Previous use.

  - Previous testing
  - Risk analysis methods.
  - Standards.
  - Software components' history.
- The elements comprising $\mathcal{K}_{compound}$. This may include:
  - Structural isolation of software components.
  - Software component's robustness.
  - Sharing of resources (CPU, library routines, data memory, stack, operating system, etc.).
  - How the software components are structurally related.

All these underlying information sources can possibly indicate if two software components are likely to fail simultaneously or not and can help experts to adjust the starting point for the simultaneous failure probability (q_{ij}) described in Subsection 5.1.

Both procedures for adjusting the starting point described in Subsections 5.2 and 5.3 assume that relevant information sources can be assigned values in the interval $[0, 1]$. A value close to 0 can for example indicate substantial difference in development methodologies, great diversity between development teams or low complexity of the interface between the software components. On the other hand, a value close to 1 can for example indicate use of identical development methodologies, extreme complexity of the interface between the software components or that components are developed by the same development team. The idea is that the larger (closer to 1) the values of the relevant information sources I_i are, the larger is the mean μ for the simultaneous failure probability in the first procedure and the number of simultaneous failures α_0 in the second procedure.

A critical question is if experts are able to express their belief about relevant information sources using a numerical scale from 0 to 1. One possible simplification is to let experts express their beliefs on an ordinal scale first and then map this onto a numerical scale. For example, for a five point ordinal scale {very low, low, medium, high, very high}, "very low" can be associated with the interval $[0, 0.2)$, "low" can be associated with the interval $[0.2, 0.4)$ and so on. Furthermore, the mean values in each interval can be used to perform the calculations in Equations 24 and 26.

6. Summary, discussion and further work

During development of the component-based approach for assessing reliability of compound software, two major challenges were identified:

1. How to identify those dependencies between pairs of software components that are of greatest importance for the calculation of the system reliability. This is necessary since it is not realistic to handle all possible dependencies in compound software.
2. How to establish prior probability distributions for probabilities that pairs of software components fail simultaneously.

Whereas the first challenge has been discussed in detail in Kristiansen et al. [27,29], the main focus of this paper has been to describe the theory behind the Bayesian hypothesis testing approach for finding upper bounds for simultaneous failure probabilities (q_{ij}). This approach consists of two main steps:

1. Establishing prior probability distributions for probabilities that pairs of software components fail simultaneously.
2. Updating these prior probability distributions by performing statistical testing.

In this paper, two possible procedures for establishing a prior probability distribution $g(q_{ij})$ for the simultaneous failure probability q_{ij} have been proposed. In the first procedure, the prior probability distribution for q_{ij} is determined by letting experts adjust the initial mean and variance of q_{ij} in the transformed beta distribution based on relevant information sources. In the second procedure, the prior transformed beta distribution for q_{ij} is adjusted numerically by letting experts express their belief in the total number of tests and the number of simultaneous failures that all relevant information sources correspond to.

By covering the second and last challenge in our approach, we finally come to the definition of a complete component-based approach for assessing reliability of compound software in which failure dependencies are explicitly addressed. To include dependency aspects in the reliability model, the approach uses information on the individual components' failure probabilities (assumed to be known) and other relevant information sources available prior to testing. It should, however, be emphasized that the proposed procedures are only suggestions on how to find prior probability distributions for probabilities that pairs of software components fail simultaneously. The validation of these procedures has not yet been performed and is one of the main tasks for further work. Furthermore, testing the complete component-based approach on a realistic case will be prioritized.

The component-based approach for assessing reliability of compound software is based on a set of assumptions (see Subsection 1.3). In the following, a short discussion regarding these assumptions is given.

Positive correlation between two software components is normally expected essentially because some inputs are more difficult (more error-prone) than others. Even if two diverse software components are developed "independently", failures are more likely to happen on certain inputs than on others. Assuming positive correlation is therefore rather realistic in many cases and far more conservative than assuming independence between software components when it comes to predicting the system's reliability. In addition, recent calculations have shown that assuming positive correlation has only minor influence on the restrictions that the marginal component reliabilities put on the conditional reliabilities in a simple two components system. However, more research on systems consisting of more than two components is needed and will be carried out as further work.

It is natural to assume that some design documents defining the architecture, component interfaces and other characteristics of the system are available when a compound software is assessed. Structure charts which graphically show the flow of data and control information between components in a compound software are of special interest. They give an overview of the software structure and are fundamental for identifying the most important component dependencies in the system, i.e. those dependencies that influence the system reliability the most.

Although the issue on how to predict reliability of individual software components is by no means trivial, our approach assumes that these probabilities are already known. How to assess these probabilities has been studied by several researchers over the years and an overview of different techniques for predicting the reliability of a particular software component based on testing can be found in, among others, Littlewood and Strigini [39], Lyu [40] and Musa [44].

Assuming that the compound software is a monotone system and that the compound software and its components have only two possible states represents a limitation made to simplify our approach. Software components and compound software do usually have a number of possible failure modes and more research on how to include multiple failure modes is needed.

Acknowledgment

This work has been performed as part of a PhD-study in collaboration between Østfold University College and the University of Oslo.

References

1. IEC 60880-2:2000: Software for computers important to safety for nuclear power plants. Software aspects of defence against common cause failures, use of software tools and of pre-developed software. 2000.
2. T. Aven. *Reliability and Risk Analysis.* Elsevier, London, 1992.
3. J. O. Berger. *Statistical Decision Theory and Bayesian Analysis.* Springer Verlag, second edition, 1985.
4. R. T. Clemen and K. C. Lichtendahl. Debiasing expert overconfidence: a Bayesian calibration model. *Working paper*, 2002.
5. R. Cooke. *Experts in Uncertainty; Opinion and Subjective Probability in Science.* Oxford University Press, 1991.
6. R. Cooke and L. Goossens. Expert judgement elicitation for risk assessments of critical infrastructures. *Journal of Risk Research*, 7:643–656, 2004.
7. 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*, pages 140–156, 2007.
8. B. Cukic, E. Gunel, H. Singh, and L. Guo. The Theory of Software Reliability Corroboration. *IEICE Transactions on Information and Systems*, E86-D(10):2121–2129, 2003.
9. G. Dahll and B. A. Gran. The Use of Bayesian Belief Nets in Safety Assessment of Software Based Systems. *International Journal of General Systems*, 29(2):205–229, 2000.

10. Y. Dai, M. Xie, K. Poh, and S. Ng. A model for correlated failures in N-version programming. *IIE Transactions*, 36(12):1183–1192, 2004.
11. D. E. Eckhardt and L. D. Lee. A theoretical basis for the analysis of redundant software subject to coincident errors. Technical report, Memo 86369, NASA, 1985.
12. P. G. Frankl, D. Hamlet, B. Littlewood, and L. Strigini. Choosing a testing method to deliver reliability. *19th International Conference on Software Engineering (ICSE'97)*, pages 68–78, 1997.
13. P. G. Frankl, D. Hamlet, B. Littlewood, and L. Strigini. Evaluating testing methods by delivered reliability. *IEEE Transactions on Software Engineering*, 24(8):586–601, 1998.
14. A. L. Goel. Software reliability models: Assumptions, limitations, and applicability. *IEEE Transactions on Software Engineering*, 11(12):1411–1423, 1985.
15. S. S. Gokhale. Architecture-based software reliability analysis: Overview and limitations. *IEEE Transactions on Dependable and Secure Computing*, 4(1):32–40, 2007.
16. 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*, pages 86–90, 1998.
17. K. Goseva-Popstojanova and K. S. Trivedi. Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, 45(2-3):179–204, 2001.
18. 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*, pages 654–657, 2008.
19. D. Hamlet. Predicting dependability by testing. *Proceedings of the 1996 ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 84–91, 1996.
20. D. Hamlet. Software component composition: a subdomain-based testing-theory foundation. *Software Testing, Verification and Reliability*, 17(4):243–269, 2007.
21. D. Hamlet, D. Mason, and D. Woit. Theory of Software Reliability Based on Components. *International Conference on Software Engineering*, 23:361–370, 2001.
22. S. Hauge, M. A. Lundteigen, P. R. Hokstad, and S. Haabrekke. Reliability Prediction Method for Safety Instrumented Systems - PDS Method Handbook. Technical report, Sintef, 2010.
23. D. Kahneman, P. Slovic, and A. Tversky. *Judgement Under Uncertainty: Heuristics and Biases*. Cambridge University Press, 1982.
24. J. C. Knight and N. G. Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Transactions on Software Engineering*, 12(1):96–109, 1986.
25. 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)*, pages 146–155, 1997.
26. M. Kristiansen. Finding Upper Bounds for Software Failure Probabilities - Experiments and Results. *Computer Safety, Reliability and Security (Safecomp 2005)*, pages 179–193, 2005.
27. M. Kristiansen and R. Winther. Assessing Reliability of Compound Software. *Risk, Reliability and Social Safety (ESREL 2007)*, pages 1731–1738, 2007.
28. M. Kristiansen, R. Winther, and B. Natvig. On Component Dependencies in Compound Software. Technical report, Department of Mathematics, University of Oslo, 2010.
29. M. Kristiansen, R. Winther, and B. Natvig. On Component Dependencies in Compound Software. *International Journal of Reliability, Quality and Safety Engineering (IJRQSE)*, 17(5):465–493, 2010.

30. M. Kristiansen, R. Winther, and J. E. Simensen. Identifying the Most Important Component Dependencies in Compound Software. *Risk, Reliability and Safety (ESREL 2009)*, pages 1333–1340, 2009.
31. M. Kristiansen, R. Winther, M. van der Meulen, and M. Revilla. The Use of Metrics to Assess Software Component Dependencies. *Risk, Reliability and Safety (ESREL 2009)*, pages 1359–1366, 2009.
32. 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*, pages 202–227, 2009.
33. 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)*, pages 32–41, 1999.
34. M. Li and C. Smidts. A Ranking of Software Engineering Measures Based on Expert Opinion. *IEEE Transactions on Software Engineering*, 29(9):811–824, 2003.
35. S. W. Lin and V. M. Bier. A study of expert overconfidence. *Reliability Engineering and System Safety*, 93:711–721, 2008.
36. B. Littlewood and D. R. Miller. Conceptual Modeling of Coincident Failures in Multiversion Software. *IEEE Transactions on Software Engineering*, 15(12):1596–1614, 1989.
37. B. Littlewood, P. Popov, and L. Strigini. Assessing the Reliability of Diverse Fault-Tolerant Systems. *Proceedings of the INucE International Conference on Control and Instrumentation in Nuclear Installations*, 2000.
38. B. Littlewood, P. Popov, and L. Strigini. Modelling software design diversity: a review. *ACM Computing Surveys*, 33(2):177–208, 2001.
39. B. Littlewood and L. Strigini. Guidelines for the statistical testing of software. Technical report, City University, London, 1998.
40. M. R. Lyu, editor. *Handbook of Software Reliability Engineering.* IEEE Computer Society Press, 1995.
41. M. A. Meyer and J. M. Booker, editors. *Eliciting and analyzing expert judgement: A practical guide.* Society of Industrial Mathematics, 2001.
42. K. W. Miller, L. J. Morell, R. E. Noonan, S. K. Park, D. M. Nicol, B. W. Murrill, and J. M. Voas. Estimating the Probability of Failure When Testing Reveals No Failures. *IEEE Transactions of Software Engineering*, 18(1):33–43, 1992.
43. A. Mosleh, V. M. Bier, and G. Apostolakis. A critique of current practice for the use of expert opinion in probabilistic risk assessment. *Reliability Engineering and System Safety*, 20:63–85, 1988.
44. J. D. Musa. *Software Reliability Engineering.* McGraw-Hill, 1998.
45. B. Natvig. *Reliability analysis with technological applications (in Norwegian).* Department of Mathematics, University of Oslo, 1998.
46. V. F. Nicola and A. Goyal. Modeling of correlated failures and community error recovery in multiversion software. *IEEE Transactions on Software Engineering*, 16(3):350–359, 1990.
47. S. Plous, editor. *The psychology of judgement and decision making.* McGraw-Hill, New York, 1995.
48. J. H. Poore, H. D. Mills, and D. Mutchler. Planning and Certifying Software System reliability. *IEEE software*, 1993.
49. 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)*, pages 53–62, 2005.
50. P. Popov, L. Strigini, J. May, and S. Kuball. Estimating Bounds on the Reliability of

Diverse Systems. *IEEE Transactions on Software Engineering*, 29(4):345–359, 2003.
51. C. V. Ramamoorthy and F. B. Bastani. Software reliability—status and perspectives. *IEEE Transactions on Software Engineering*, pages 354–371, 1982.
52. R. H. Reussner, H. W. Schmidt, and I. H. Poernomo. Reliability prediction for component-based software architectures. *Journal of Systems and Software*, 66(3):241–252, 2003.
53. J. A. Scott and J. D. Lawrence. Testing existing software for safety-related applications. Technical report, Lawrence Livermore National Laboratory, 1995.
54. 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)*, pages 12–19, 2001.
55. 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)*, pages 82–87, 2002.
56. L. A. Tomek, J. K. Muppala, and K. S. Trivedi. Modeling Correlation in Software Recovery Blocks. *IEEE Transactions on Software Engineering*, 19(11):1071–1086, 1993.
57. P. T. Trung and H. Q. Thang. Building the reliability prediction model of component-based software architectures. *Int'l Journal of Information Technology*, 5(1):18–25, 2009.
58. M. Vieira and D. Richardson. The role of dependencies in component-based systems evolution. *Proceedings of the International Workshop on Principles of Software Evolution*, pages 62–65, 2002.
59. J. M. Voas and K. W. Miller. Software testability: The new verification. *IEEE Software*, pages 17–28, 1995.
60. R. Winther and M. Kristiansen. On the Modelling of Failure Dependencies Between Software Components. *Safety and Reliability for Managing Risk (ESREL'06)*, pages 1443–1450, 2006.
61. S. Yacoub, B. Cukic, and H. Ammar. A Scenario-Based Reliability Analysis Approach for Component-based Software. *IEEE Transactions on Reliability*, 53(4):465–480, 2004.
62. X. Zhang and H. Pham. An Analysis of Factors Affecting Software Reliability. *Journal of Systems and Software*, 50(1):43–56, 2000.

Monica Kristiansen received a cand. scient degree in applied statistics at the Agricultural University of Norway (now Norwegian University of Life Sciences) in 2000. She started working as a researcher at the Institute of Energy Technology in Halden (Halden Reactor Project) in 2000 and in 2002 she began working on her Ph.D. degree in statistics at the University of Oslo. Kristiansen is now working as an associate professor at Østfold University College. Her main research interests are software reliability and dependability, risk analysis and pre-developed software.

Rune Winther received a cand. scient degree at the University of Oslo in 1991, and a Ph.D. degree from the same university in 1996. He has held positions as associate professor at Rogaland University Centre (now the University of Stavanger) and Østfold University College. Winther has also worked as a researcher at the

Institute for Energy Technology in Halden (the Halden Reactor Project) and worked for several consultant companies. His research activities include reliability theory, software reliability and risk analysis of complex critical systems.

Bent Natvig received a cand. real degree at the University of Oslo in 1970 and a Ph.D. degree at the University of Sheffield in 1976. He became a senior lecturer at the Department of Statistics and Insurance Mathematics at the University of Oslo in 1981 and a full professor in 1986 with special duties in reliability. From 1990 - 1993 he served as a dean of the faculty of Natural Sciences and Mathematics at the University of Oslo and from 2000 - 2005 as chairman of the National Committee for Research Ethics in Science and Technology. He was elected member of the International Statistical Institute in 1997. Furthermore, he is the author of the 2011 Wiley book: Multistate Systems Reliability Theory with Applications.