Reliability Allocation for a Software System with Modular Structure *

Robert Burnett† Tom Anderson‡
R.C.Burnett@newcastle.ac.uk Tom.Anderson@newcastle.ac.uk
Department of Computing Science
University of Newcastle upon Tyne
Newcastle upon Tyne—England NE1 7RU
Fax: +44 91 222 8232

Abstract

This paper proposes a method for allocating an appropriate reliability requirement to each module of a modular software system, using Markov analysis. A formula to calculate an estimate of the overall system reliability is established. From that formula, a procedure to allocate the reliability requirement for each module is derived using an optimization process, taking into account the known overall required level of reliability.

Key-words: Markov Analysis, Optimization, System Reliability, Reliability Allocation

1 Introduction

To avoid the problems of complexity which the design of a single monolithic software system creates, it is usual to divide software into separate components called modules, which are subsequently integrated to satisfy problem requirements [10]. Modular software systems consist of a set of modules which carry out a range of different tasks. Among these modules there exists a pre-defined structure of “who calls who”, which is known from a detailed requirement specification. This report considers a particular form of module interaction, where after a module has completed its execution, the control of the system is passed to another module, on either a deterministic or stochastic basis (as exemplified in fig. 1). The pattern of interconnection among the modules, i.e., the sequence in which modules are executed, is assumed to follow a Markov process.

†This work has been partially funded by CNPq/Brazil (grant no. 200487/92-2) and ESPRIT Basic Research Action PDCS.
‡On leave from Departments of Informatics, CEFET/PR and PUC/PR, Curitiba, Paraná, Brazil.
†Tom Anderson is a Professor in the Department of Computing Science at the University of Newcastle upon Tyne.

†In a Markov process the transitions between states do not depend on past history, nor on the current time, but only on the current state. So the probability of calling a given module is only a function of the module currently being executed and the given module.
The assumption of a Markov process is a good representation of the actual control exchange process in many applications, and is frequently used in software engineering practice [3]. An example can be seen in [7], where it is assumed that transitions between modules follow a Markov process.

When a software system has a modular structure it is apparent that the overall level of system reliability that will be experienced by the user depends on the sequence of modules to be executed and, naturally, on the reliability of each individual module [3, 8]. The reliability of any software system also depends, of course, on the profile of use, that is, the dynamic characteristics of a typical execution of the system in a particular user environment—a system operating in two distinct environments will exhibit different levels of reliability, depending on the utilization of the modules in each of the environments.

A transition matrix can be defined that expresses the pattern of interaction between the modules; this matrix can be used to represent (some aspects of) the behaviour of the modular structure. This matrix underlies the relationship between the overall system reliability and the reliability of each module.

The following definitions are employed in defining the transition matrix:

- $M_i$ represents a generic module $i$ and forms a "state $S_i$" of a system with $n$ modules ($n = 4$ in the example to follow);

- $R_i$ is the reliability of module $M_i$. There are two ways of dealing with software reliability [1]:
  - failure rate over time—reliability is: the probability that a module $M_i$ operates according to specifications for a given period of time before a failure;
  - failure rate per demand for service—reliability is: the probability that module $M_i$ will operate according to its specification when called and will transfer control correctly when finished.

The latter approach is utilized in this work;

- $P_{ij}$ is the probability that the transition between modules $M_i$ and $M_j$ will be taken, given that control is at module $M_i$ and execution is completed according
to its specification. The values $P_{ij}$ have to be obtained from the requirement specification of the system ($0 \leq P_{ij} \leq 1$), as, for example, suggested in [15];

- $R_i P_{ij}$ thus represents the probability that the execution of module $M_i$ completes according to its specification and control of the system is transferred then to module $M_j$;

- $M_1$ is the start module, that is, $S_1$ is the initial state of the system;

- $F$ is an absorbing (terminal) state that is reached when a module produces a result not conforming to its specification, that is, when a failure of that module occurs. This state is reached from module $M_i$ with the probability $(1 - R_i)$;

- $T$ is an absorbing (terminal) state which is reached when the system of software modules completes its overall task successfully. More precisely, a module $M_i$ will make a transition to state $T$, with probability $R_i P_{iT}$, if the execution of $M_i$ completes according to its specification and $M_i$ should not then make a transition to any other module $M_j$. So $\sum_{j=1}^{n} P_{ij} + P_{iT} = 1$;

- $R_{req}$ is the overall reliability of the system that the user needs to achieve. The value for this reliability is known in advance in the design stage;

- $R_{calc}$ is the reliability of the system obtained from the transition matrix using Markov analysis. It is the probability of reaching the terminal state $T$ from the initial state $M_1$. This represents the probability that the system completes its execution without failing.

As an example, the following matrix describes a modular structure having four modules:

$$
\begin{bmatrix}
M_1 & M_2 & M_3 & M_4 & F & T \\
M_1 & 0 & R_1 P_{12} & R_1 P_{13} & R_1 P_{14} & (1 - R_1) & R_1 P_{1T} \\
M_2 & R_2 P_{21} & 0 & R_2 P_{23} & R_2 P_{24} & (1 - R_2) & R_2 P_{2T} \\
M_3 & R_3 P_{31} & R_3 P_{32} & 0 & R_3 P_{34} & (1 - R_3) & R_3 P_{3T} \\
M_4 & R_4 P_{41} & R_4 P_{42} & R_4 P_{43} & 0 & (1 - R_4) & R_4 P_{4T} \\
F & 0 & 0 & 0 & 0 & 1 & 0 \\
T & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
$$

An iterative process can be used to refine estimates of $R_1, \ldots, R_n$ such that Markov analysis would yield a result $R_{calc} \geq R_{req}$ with $R_{calc}$ as close to $R_{req}$ as possible (since this will keep software development costs down).

It is well known that to achieve a higher figure for system reliability than $R_{req}$ would entail spending more time and cost during development; aiming at a minimum acceptable reliability $R_{calc}$ indicates that we seek to keep the development cost of the software to a minimum also.

The values obtained for each of the $R_i$ from this iterative process constitute the proposed reliability allocation for the modules $M_i$.

This problem may be stated in the following form.

Find a value for $R$ such that $R_{calc}$ is minimized

subject to
(i) \( 0 < R_i < 1 \), where \( i = 1, \ldots, n \)

(ii) \( R_{\text{calc}}(R) - R_{\text{req}} \geq 0 \)

Any set of values \( < R_1, \ldots, R_n > \) that satisfies the conditions above would be an acceptable set. The problem is to find an acceptable set which allows a compromise between the reliabilities \( R_i \) and other constraints of the system; a case in point would be cost. The result obtained here addresses this problem.

In the following sections this problem is elaborated and a proposed solution is presented. The paper has the following structure: section 2 derives a formula for calculating the overall reliability \( R_{\text{calc}} \), section 3 explains how to find the values \( R_i \), and section 4 is the conclusion.

2 Determination of the Reliability of a System

The transition matrix defined in section 1 describes a finite Markov process with two absorbing states \( T \) and \( F \), and a set of \( n \) transient states \( S_1, \ldots, S_n \). The matrix can be depicted in the following form:

\[
P = \begin{bmatrix} S & A \\ A & 0 & I \end{bmatrix}
\]

Here \( A \) represents the absorbing states and \( S \) represents the transient states. The matrix \( Q \) contains the probabilities of transitions between the transient states. The matrix \( R \) contains the transition probabilities from the transient states to the absorbing states.

The transition probabilities matrix that represents the transformation of the system after \( k \) steps is given by forming powers of the single step matrix \( P \), that is, \( P^k \) [14]. This \( k \)-step transition probability matrix \( P^k \) has the following form:

\[
P^k = \begin{bmatrix} Q^k & R^k \\ 0 & I \end{bmatrix}
\]

The \( Q_{ij}^k \) entry of the matrix \( Q^k \) denotes the probability of arriving in transient state \( S_j \) after exactly \( k \) steps starting from transient state \( S_i \).

Hence the probability of arriving in transient state \( S_j \) after (exactly 0, or exactly 1, or \( \ldots \), or exactly \( k \)) steps, starting the system from transient state \( S_i \), is then \( W_{ij} \) where

\[
W = Q^0 or Q^1 or Q^2 or \ldots or Q^n
\]

\[
W = I + Q + Q^2 + \ldots + Q^k = \sum_{i=0}^{k} Q^i
\]

It is shown in [6] that if \( Q^k \to 0 \), when \( k \to \infty \) (which the case here since \( Q \) is a matrix of probabilities), then

\[
I + Q + Q^2 + \ldots + Q^k \approx (I - Q)^{-1}
\]

Hence the limiting value of \( W \) as \( k \) increases is very close to the matrix inverse \( (I - Q)^{-1} \); this is called the fundamental matrix of the Markov chain [14].
The matrix \((I - Q)^{-1}\), which we will now refer to as \(W\), enables us to calculate the transition probabilities we need. The probability of the system reaching state \(j\) after some number of steps, starting in state \(i\), is \(W_{ij}\).

Now we can calculate the probability of reaching state \(T\), after starting the system in state \(S_1\) (corresponding to module \(M_1\)).

Given that

- \(W_{1i}\) is the probability of reaching state \(i\) from state \(1\) (after an unspecified number of steps);
- \(R_i P_{IT}\) is the probability of reaching state \(T\) from state \(i\) in one step;

Let \(P_{Si}\) be the probability that starting from state \(1\) the system reaches state \(S_i\) after an arbitrary number \(z\) of steps and then in one further step reaches \(T\) directly from \(S_i\) (fig. 2), then

\[
P_{Si} = W_{1i} \cap R_i P_{iT}
\]

As the probability \(R_i P_{iT}\) does not depend on \(W_{1i}\), then

\[
P_{Si} = W_{1i} R_i P_{iT}
\]

The overall reliability of a system \(R_{\text{calc}}\) will then be the probability that starting in state \(1\) the system enters the absorbing state \(T\), from any state \(S_i\) (fig. 3).

\[
R_{\text{calc}} = P_{S1} \text{or } P_{S2} \text{or } \ldots \text{or } P_{Sn}
\]
\[ R_{\text{calc}} = P_{S1} + P_{S2} + \cdots + P_{Sn} = \sum_{i=1}^{n} P_{Si} \]

\[ R_{\text{calc}} = \sum_{i=1}^{n} W_{1i} R_{i} P_{iT} \]  

(1)

Formula (1) allows us to determine the overall reliability of a system \( R_{\text{calc}} \) from the reliability of each module \( R_i \), and the transitions probabilities \( P_{ij} \) and \( P_{iT} \). In section 3 this formula is utilized to find the values of \( R_i \) corresponding to \( R_{\text{calc}} \), \( P_{ij} \) and \( P_{iT} \), which are known in advance.

Formula (1) is a generalization of that given in [3], where here there are no restrictions on the number of states that can reach state \( T \). In [12] this formula is briefly cited.

3 Allocation of the Reliability

In this section we describe a method for allocating the values \( R_i \) such that if the modules \( M_i \) attain reliability \( R_i \) then the desired overall software system reliability \( R_{\text{req}} \) will be achieved. (see section 1).

This problem can be summarized as follows.

We seek values of \(< R_1, \ldots, R_n >\) which will give a value of \( R_{\text{calc}} \) close or equal to \( R_{\text{req}} \), but we can only use values of \( R_i \) with \( 0 < R_i < 1 \), and such that we obtain \( R_{\text{calc}} \geq R_{\text{req}} \).

We know:

a. the required reliability \( R_{\text{req}} \), which is given in advance;

b. the transition probabilities \( P_{ij} \) and \( P_{iT} \), which are obtained from the requirement specification;

to accomplish this task we have the following formulae, which are described in section 2:

\[ R_{\text{calc}} = \sum_{i=1}^{n} W_{1i} R_{i} P_{iT} \]

where \( W = X^{-1} \) with \( X_{ij} = \begin{cases} 1 & i = j \\ -R_i P_{ij} & i \neq j \end{cases} \)

We can summarize this problem description as follows:

Find values of \( R \) such that \( R_{\text{calc}} \) is minimized, subject to

\[ R_{\text{calc}}(R) - R_{\text{req}} \geq 0; \ 0 < R_i < 1, \ \text{where} \ i = 1, \ldots, n \]

The resulting solution of this minimization problem will be the allocation of the reliabilities \( R_1, \ldots, R_n \).

Framework of the Solution

To deal with the problem stated above, the published NAG routines E04VDF and F04AAF [9] were used as shown in the framework of figure 4. The routine E04VDF is a routine to minimize an arbitrary function subject to constraints, which may include simple bounds on the variables, linear constraints and non-linear constraints. F04AAF solves a system of equations with multiple right-hand sides and thereby allows us to calculate \( W_{1i} \).
Begin Program
Supply Limit on maximum number of iterations in routine E04VDF;
Supply Accuracy required from the solution;
Supply $R_{req}$;
Supply Upper bounds on module reliabilities, as constraints for routine E04VDF;
Supply Transition matrix (values $P_{ij}$ and $P_{IT}$);
Begin E04VDF *optimization routine*
    Count-of-iterations= 0;
Repeat
    Generate $R_1, \ldots, R_n$; *see Note 1*
    Call Routine to calculate $W_i$ *using routine F04AAF*
    Call Routine to evaluate the square residual
        $(R_{calc} - R_{req})^2$; *see Note 2*
    If $R_1, \ldots, R_n$ constitute an optimal solution *see Note 3*
        then $R_1, \ldots, R_n$ are the results needed;
        Return-code= 0;
        Exit E04VDF;
    Endif;
    If E04VDF considers pointless to progress *see Note 4*
        then Return-code $\neq 0$
        Exit E04VDF;
    Endif;
    Return-code= 3; *see Note 5*
    Increment Count-of-iterations;
Until Limit maximum of iterations has been reached;
End E04VDF;
If Return-code $\neq 0$
    then an alternative $R_{req}$ may be suggested *see Note 6*
    else the values $R_1, \ldots, R_n$ produced by E04VDF are the results required;
Endif;
End Program.

Figure 4: Framework of the solution

Notes about fig.4:

1. The routine generates new set of values $R_1, \ldots, R_n$ derived from the values used in the previous iteration—the first estimates are based on the upper bounds for the module reliabilities, as supplied;

2. The function $(R_{calc} - R_{req})^2$ was utilized instead of $R_{calc} - R_{req}$. Using the square term proved, in practice, to be mathematically convenient and easier to handle;

3. An optimal solution is found when:
   a. The partial derivative of the function $R_{calc}$ with respect to $R_i$ is sufficiently small, considering the accuracy required; and
   b. The residuals of constraints (upper bounds on reliabilities and objective function $(R_{calc} - R_{req})^2$) are sufficiently small, again considering the accuracy required; and
c. The values for $R_1, \ldots, R_n$ do not change significantly between iterations.

4. No feasible values for $R_1, \ldots, R_n$ could be found; the routine terminates. In this case one of a number of return codes is generated to indicate the likely cause of this abnormal end.

5. If the program concludes with Return-code equal to 3 then the limit on the maximum of iterations has been reached without any solution being found. If it is thought that the routine needs to perform more iterations, then the value of that limit should be set higher;

6. In this case the user should re-run the program with new values for $R_{req}$ and/or upper bounds on the reliabilities for each module.

The program that executes the procedures outlined in figure 4 is shown elsewhere [2].

To find the first row of $X^{-1}$, that is, $W_{1i}$, we have

$$X^T (X^{-1})^T = I$$

and so solving

$$X^T r = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

will give us $r^T$ as the required answer $W_{1i}$. Putting the problem in this form means we can take advantage of standard procedures, as provided by NAG routine F04AAF. This routine solves the equation $Ax = B$, where, in this case, $A = X^T$, $x = r$ and $B = (1, \ldots, 0)$.

The initial estimates for $R_1, \ldots, R_n$ have a significant effect on the outcome of the program, since the function $R_{calc}(R)$ has several local minima. By experiment it was found that taking initial estimates slightly lower than an upper bound on the acceptable limit produced satisfactory results (see examples). These limits must be chosen less than 1.0. Although the maximum theoretical value for reliability is 1.0, it is not achieved in practical software systems.

Examples

We now give some examples using the program that was developed (see Appendix A). Examples 1 to 3 use a structure with 3 modules, while examples 4 and 5 use a structure with 6 modules (figure 5).

Through variation of the upper bounds, an adjustment of reliabilities is shown in the following examples. In example 2 the required reliability is not feasible, because the upper bounds are too low.

**EXAMPLE 1**

Reliability required ... 0.900

<table>
<thead>
<tr>
<th>Upper Bound for each module</th>
<th>Transition Matrix</th>
</tr>
</thead>
</table>

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor
Figure 5: Example of a system with 3 and 6 modules

Reliability of each module

\[ \begin{align*}
R_1 &= 0.980; \\
R_2 &= 0.950; \\
R_3 &= 0.890
\end{align*} \]

EXAMPLE 2

Reliability required \( \ldots \) 0.900

<table>
<thead>
<tr>
<th>Module</th>
<th>0.9000</th>
<th>0.00</th>
<th>0.60</th>
<th>0.40</th>
<th>0.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module 1</td>
<td>0.9800</td>
<td>0.00</td>
<td>0.60</td>
<td>0.40</td>
<td>0.00</td>
</tr>
<tr>
<td>Module 2</td>
<td>0.9500</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Module 3</td>
<td>0.8900</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Transition Matrix

\[ \begin{align*}
&0.00 & 0.60 & 0.40 & 0.00 \\
&0.00 & 0.00 & 0.00 & 1.00 \\
&0.00 & 0.00 & 0.00 & 1.00
\end{align*} \]

Comment: the following example adopts suggestion 2.
EXAMPLE 3

Reliability required ... 0.900

Upper Bound for each module
Module 1: 0.970
Module 2: 0.955
Module 3: 0.895

Reliability calculated ... 0.9031

Reliability of each module
R 1: 0.9700; R 2: 0.9550; R 3: 0.8950

EXAMPLE 4

Reliability required ... 0.900

Upper Bound for each module
Module 1: 0.990
Module 2: 0.990
Module 3: 0.990
Module 4: 0.870
Module 5: 0.980
Module 6: 0.980

Reliability calculated ... 0.9084

Reliability of each module
R 1: 0.9658; R 2: 0.9543; R 3: 0.9233; R 4: 0.9805; R 5: 0.9334; R 6: 0.9454

Comment: in example 4, all upper bounds were set extremely high. In the following example, the upper bounds were reduced.

EXAMPLE 5

Reliability required ... 0.900

Upper Bound for each module
Module 1: 0.970
Module 2: 0.980
Module 3: 0.970
Module 4: 0.870
Module 5: 0.980
Module 6: 0.980

Reliability calculated ... 0.9084

Reliability of each module
It should be observed that in examples 1 and 4 the required and calculated reliabilities are identical. If it was known in advance that such a result could be achieved, then a more direct method could be used to solve the problem. An appropriate root-finding technique, such as Newton's method, could be employed to find values of $R$ which satisfied

$$f(R) = R_{\text{calc}}(R) - R_{\text{req}} = 0$$

However, the problem of constraining the permitted values for $R_i$ would complicate this approach. In general, we cannot expect a value of $R_{\text{calc}}$ equal to $R_{\text{req}}$ to be attainable (see examples) and hence the minimization approach adopted here has wider applicability.

4 Conclusion

This paper elaborates a proposal for allocating reliability levels to modules of a software system, when a desired overall reliability is known in advance. A formula was obtained that allows us to calculate the overall system reliability using Markov analysis. From that formula and using a minimization approach a reliability level for each module can be selected to ensure that the difference between the overall calculated reliability and the required reliability is a minimum.

It is well known that in software systems a high reliability requirement means that the system will need more time and cost for development. For this reason it is assumed that the minimum acceptable value for the overall reliability of the system is known in advance.

The outcome of this work will be utilized in a tradeoff model between cost and reliability currently being developed. For this purpose, the function that defines the cost constraint must be included in the procedure. Depending on the cost function to be used, the framework of this solution may require some adjustment.

Acknowledgments

We wish to thank Dr. Chris Phillips and Dr. Chris Woodford for their helpful suggestions and useful comments during the development of this work.

References


