Influence of Architecture on Software Reliability

Norman Schneidewind
Email: ieeelife@yahoo.com

Introduction
Today, it is important to integrate reliability with the performance of software architecture because architectural characteristics, such as the complexity of software, can have a profound influence on reliability. The need to deal simultaneously with both performance and dependability (reliability) was initially recognized in the mid-1970s. More specifically, it surfaced as the consequence of then-experimental architectures such as PRIME (developed at the University of California, Berkeley [1]), which exhibited “degradable” performance in the presence of operational faults. Performance refers to how effectively (e.g., throughput, delay) or efficiently (e.g., resource utilization) a system delivers a specified service, presuming it is delivered correctly. On the other hand, dependability or what is also referred to as reliability is the trustworthiness of a computer system [2] with respect to delivery of a specified service. With this view, a system failure occurs if the delivered service no longer complies with its specification. Moreover, since faults are the source of failures, their existence and their effects are principal concerns [3].

Measures to Integrate Reliability and Performance
In order to integrate reliability and performance, the concept of performability was developed that refers to measures that quantify a system’s ability to perform in the presence of faults. The definition of the computational capacity of a system state is the amount of useful computation per unit time available on the system in that state [4]. For example, the amount of computation that could be performed when not in the failed state. In addition, the expected efficiency of the computation should be evaluated, as we do in the following sections.

Definitions
Ta: time available to perform a task
Trj: time required to perform a task r on architecture j
Pj: performance of architecture j
Rrj: software reliability of architecture j during time Tr
N: number of instructions in a program
Erj: computational efficiency of task r on architecture j

Software Architectural Model
The amount of computation that can be performed by architecture j on task r is given by equation (1.1).

\[ T_{ij} = \frac{N}{P_j} \]  

(1.1)

Using equation (1.1), the computational efficiency of task r on architecture j is computed in equation (1.2), where lower values correspond to higher efficiencies.

\[ E_{ij} = \left( \frac{T_{ij}}{T_a} \right) = \left( \frac{N}{P_j} \right) / T_a \]  

(1.2)
Architectural Characteristics

Few techniques consider failure propagation in software architectures for software reliability assessment. One study that considered failure propagation analyzed failure propagation based on architectural service routes (ASRs). An ASR is a sequence of components that are connected through interfaces [5]. We consider failure propagation as generated by instruction and data flow paths, similar to components connected by interfaces. Architectural characteristics, such as data flow paths, influence the reliability that can be achieved in a system. A data flow path that is a sequence of nodes and edges is highlighted in red in Figure 1. This is an example of where failures could occur. In addition to the many avenues for failure that are exposed by an architecture with many data flow paths, this condition makes it difficult to recover from failures and to repair the faults responsible for failures.

Some characteristics of software architecture are related to software system configuration. An example is shown in Figure 1 that shows an operating system architecture based on the Intel XScale Technology [6]. The complexity of the configuration will influence software reliability.

![Operating System Architecture](image)

Figure 1. Operating System Architecture (Based on Intel XScale Technology)

- Number of Nodes: \( N_n = 11 \)
- Number of Edges: \( N_e = 25 \)
- Cyclomatic Complexity: \( CC = N_e - N_n + 1 = 15 \)
We account for this complexity by enumerating the nodes and edges in the configuration, and then computing the cyclomatic complexity (CC) [7] from these elements, as shown in Figure 1. Cyclomatic complexity is used as a weighting factor and applied to reliability to account for the complexity of the architecture.

The cyclomatic complexity of software architecture \( j \) is computed in equation (1.3), where \( N_{je} \) is number of edges and \( N_{jn} \) is number of nodes.

\[
CC_j = (N_{je} - N_{jn}) + 1
\]  

(1.3)

To compute a weighting factor, a maximum value \( CC_m \) of \( CC_j \) is needed to normalize the complexity in equation (1.3). \( CC_m \) is obtained by recognizing that the most complex architecture is one that has a single master node that controls all slave edges through which control and application data must pass. In other words, \( N_{jn} = 1 \) in equation (1.3), so that maximum cyclomatic complexity is expressed in equation (1.4).

\[
CC_m = N_{je}
\]  

(1.4)

Using equations (1.3) and (1.4), the weighting factor for architecture \( j \) is expressed in equation (1.5).

\[
w_j = CC_j / CC_m = (N_{jn} - 1) / N_{je}
\]  

(1.5)

**Integrating Software Reliability**

Now, we include software reliability in the architectural model. To do this, we use the binomial distribution to represent the probability \( P(x) \), of \( x \) number of instructions in a program that have faults and fail, each with a probability of failure \( p \), in a program of size \( N \), in equation (1.6). We define a reliable program as one that has \( x = 0 \) instructions that fail. Therefore, equation (1.6) is compressed to equation (1.7), which is the software reliability of a program with \( N \) instructions.

\[
P(x) = \left( \frac{N!}{(x!)(N-x)!} \right) (p^x)(1-p)^{(N-x)}
\]  

(1.6)

\[
R_{ej}(N) = P(0) = (1 - p)^N
\]  

(1.7)

Now, the weighting factor in equation (1.5) can be integrated with the equation for software reliability to produce equation (1.8).

\[
E_{ej} = ((1 - p)^N \cdot (N/P_j) / T_a) \cdot ((N_{jn} - 1) / N_{je})
\]  

(1.8)

The software architecture (operating system) is implemented on the Intel XScale processor, with the clock and instruction execution rates shown below. In order to do the evaluation of computational efficiency, using equation (1.8), a range of performances are used, as shown in Table 1, where the bolded quantities are the clock rates listed below. Note that in addition to computing computational efficiency, software reliability could be predicted for given values of computational efficiency.
Intel XScale Processor
Clock rates:
• 266 MHz, 266 million instructions per second
• 400 MHz, 400 million instructions per second
• 533 MHz, 533 million instructions per second

Result of Integrating Reliability with Computational Efficiency
The result of the integration of reliability with computational efficiency is shown in Figure 2. Fortuitously, both efficiency and reliability improve, as performance increases. The practical application of this analysis is that several architectures of operating systems, or application software, could be evaluated, using this process, to identify the one with the best efficiency and reliability. Using this approach, the engineer can do a tradeoff analysis to see how much processor computational power is warranted to achieve efficiency and reliability goals. For example, looking at Figure 2 you can see that increasing the processor clock rate beyond 800 Mhz (800 million instructions per second) would not significantly improve efficiency and reliability. Therefore, the additional cost would not be justified.

Table 1 Software Architecture Efficiency and Reliability

<table>
<thead>
<tr>
<th>Mhz</th>
<th>pj</th>
<th>p</th>
<th>Erij</th>
<th>R_e (N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.000100</td>
<td>1.47</td>
<td>0.3679</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>0.000090</td>
<td>0.81</td>
<td>0.4066</td>
<td></td>
</tr>
<tr>
<td>266</td>
<td>0.000080</td>
<td>0.68</td>
<td>0.4493</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>0.000070</td>
<td>0.66</td>
<td>0.4966</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>0.000060</td>
<td>0.55</td>
<td>0.5488</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>0.000050</td>
<td>0.49</td>
<td>0.6065</td>
<td></td>
</tr>
<tr>
<td>533</td>
<td>0.000040</td>
<td>0.50</td>
<td>0.6703</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>0.000030</td>
<td>0.49</td>
<td>0.7408</td>
<td></td>
</tr>
<tr>
<td>700</td>
<td>0.000020</td>
<td>0.47</td>
<td>0.8187</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>0.000010</td>
<td>0.45</td>
<td>0.9048</td>
<td></td>
</tr>
<tr>
<td>900</td>
<td>0.000009</td>
<td>0.41</td>
<td>0.9139</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0.000008</td>
<td>0.37</td>
<td>0.9231</td>
<td></td>
</tr>
</tbody>
</table>

Nje, 25 number of edges
Njn, 11 number of nodes
N, 10000 number of instructions
Ta, 10 time available (hours)
Summary
Recognizing the important need to consider the interaction between reliability and performance a model for integrating software reliability with software architecture has been developed. In addition to the reliability of the architecture, the computational efficiency of the architecture was evaluated. In order to illustrate the evaluation process, a microprocessor and its operating system from industry were used. This example showed that both efficiency and reliability improved as performance increased. Due to a space limitation, only one architecture was evaluated, but other architectures could be evaluated using the same process. A major determinant of efficiency and reliability is the complexity of the architecture. The reliability engineer can use this model to predict whether, for a given architecture, reliability goals are likely to be achieved. Conversely, given reliability goals, the engineer can assess whether computational efficiency could be achieved for an architecture.

References