Exploring Robust Component-Based Software

Xiaobu Yuan  
School of Computer Science  
University of Windsor  
N9B 3P4  
1-519-253-3000 Ext 3790  
xyuan@uwindsor.ca  

Shangwei Duan  
School of Computer Science  
University of Windsor  
N9B 3P4  
1-519-253-3000 Ext 4407  
swduan@cs.uwindsor.ca  

Zhiyong Liu  
School of Computer Science  
University of Windsor  
N9B 3P4  
1-519-253-3000 Ext 3790  
Liu12s@uwindsor.ca  

ABSTRACT
The emerging technology of component-based software engineering offers huge potentials for quicker and easier software development. Meanwhile, it imposes big challenges on performance optimization of component-based software systems. Tolerance design deals with software robustness, and robust design of component-based software systems facilitates performance optimization. This paper investigates the application of statistical methods in robust design of component-based software systems. The objective of this research is to create a quantitative method supporting software performance evaluation in the design stage, thus supplying guidelines towards the design of component-based software systems. The efficacy of the proposed approach is illustrated in experiments on a practical component-based system.

Categories and Subject Descriptors
D.2.8 [Software Engineering]: Metrics - Performance measures

General Terms
Design, Performance, Experimentation

Keywords
Parameter design, robustness, signal to noise

1. INTRODUCTION
Component-based software development is an emerging software technology that is about to take the software industry by storm [1]. Rooted in the philosophy of separation of concerns and built on the existing object-oriented (OO) approach, component-based software systems are constructed with pre-developed components rather than from scratch. In comparison to objects in OO, components are more adaptive in respect to reusability: Components can be reused across more than one application without code modification, while objects are generally limited to a specific type of applications. A specific object becomes a component after being evolved to a general reusable unit.

Components are more adaptive than objects, but the performance of component systems is also facing bigger challenges OO systems. The separation of concerns, which helps to divide big problems into smaller pieces, lacks the capability of global optimization that is vital for improving system performance. Though having the same problem of global optimization, OO systems can fortunately offset the negative impact through OO compilers. Evidence shows that the speed of C++ programs is within ±10% of their C counterparts with the help of compilers [2].

Theoretically, component-based software could also use compilers to optimize performance. However, its differences from OO technology create two big problems. The first problem is that components are designed for general applications. It becomes too late for optimization in the compiling phase as performance problems start to arise as early as in the design phase. The second problem is related to the fact that components are software units developed by third parties and published in the format of binary codes. It is impossible for compilers to optimize “off-the-shelf” components.

In fact, software engineers have started working on the performance issues in the design phase even before component-based software engineering emerges. Performance analysis in software design phase typically involves three steps [3]. The first step is to describe software units and their relationships in United Model Language (UML) [4] or Architecture Describe Language (ADL) [5] [6] [7]. The second step then transforms the design of a software system in forms of descriptive UML models or ADL to a performance analytic or simulated model, such as the queuing network model (QN) [8], layered queuing network model (LQN) [9], and Stochastic Rendezvous Network Model (SRN) [10]. Experiments on the performance model are conducted by a performance analyzer or simulation tool. The experiment results are fed back in the final step for software designers to refine UML or ADL models according to the result of performance analysis.

This typical procedure of software performance optimization also applies to component-based software systems. Actually, this practice is even more important for component-based software systems due to the fact that components cannot be customized in the code level for a given implementation. The codes of components usually cannot be modified for a specific application, but there are some parameters of each component that are controllable by designers. The efficiency of co-operations and interactions between components is greatly affected by the adjustment of the controllable component parameters. In such a sense, performance optimization of a component-based software system is mainly determined by parameters design.
Parameter design needs to set suitable values of controllable component parameters for optimized performance of the whole system. The control parameters design is therefore also called sensitivity design. However, not all input parameters are controllable in practice. There are noise parameters too. For example, the noise signal of communication transmission is a noise parameter of communication systems. In addition, some parameters of components are too costly to configure. A typical example is whether or not to implement synchronous writing in Network File System (NFS) for Linux. These noise parameters have great influences on system performance as well.

Consequently, the main objective of parameters design is to achieve a desirable performance of the whole system by setting suitable values of controllable parameters in relation to noise parameters. A design is a robust design when it takes into consideration both the control and noise parameters. This paper investigates the application of statistical methods in robust design of component-based software systems. It develops a statistical approach that embodies parameter design between the second and third steps of the performance optimization process. The goal is to provide quantitative feedback on parameters selection for software architect design.

In the rest of this paper, Section 2 discusses previous works related to sensitivity design and tolerance design. Section 3 introduces the concept of software robustness first, and then incorporates tolerance design into the three-step process of performance evaluation to determine the control parameters for robust component-based software systems. Section 4 illustrates the robustness adjustment strategy with an experiment on a real world software system. Conclusions are finally drawn in the last section.

2. RELATED WORKS

Sensitivity analysis is a widely mentioned phrase in associated with software performance optimization, and two approaches have been reported in the literature. By establishing the relationship of output (performance metric) and a particular input parameter with a formula, the first approach decides the sensitivity to this parameter according to derivations of performance metrics with respect to the parameter [11]. Alternatively, the second approach determines sensitivity by analyzing a relationship curve between performance metric and the input parameter plotted with results from a large number of experiments [12]. Both approaches work with only one parameter at one time. When more parameters are studied simultaneously, it becomes either too complex to resolve the formula or too time-consuming to conduct experiments.

Tolerance analysis is seldom studied in software research literature. In the area of software quality assurance, research on robustness is closely related to tolerance analysis. Lung initially proposed a comprehensive list of software architectural metrics and applied them to analyze software architecture robustness [13]. His research focused on architecture’s tolerance on changes of functionality, whereas the tolerance of software performance on the noise parameters was not addressed.

There are also efforts to introduce statistical methods of experiments analysis into computer-related studies, including software engineering. For example, Moore and Ray investigated the statistical methods of sensitivity and performance analysis based on complex computer simulation experiments, and discussed the adaptability of those methods [13]. Chen and Stromberg used regression method to obtain estimations on software robustness [14]. Akingbehin proposed a set of metrics based on Taguchi philosophy to measure software quality [15]. These research works demonstrated the powerful abilities of statistical methods in software engineering.

The approach to be presented in this paper explores the application of statistical methods in the newly emerged software development paradigm of component-based software engineering. It identifies control and noise parameters in parameters design, and uses them to construct an analysis matrix with experiments on an analytic or simulated model. The key contribution of this work is the suggested quantiative analysis on signal-to-noise ratio that provides the feedback information to determine a configuration with the best possibility of producing a robust system.

3. METHODOLOGY

3.1 Software System Robustness: A Definition

Software system robustness is based on the performance sensitivity of a software system in relation to its internal or external unpredictable changes. It defines the degree of sensitivity of the system’s performance metrics, subject to the effects of change in the noise parameters. Fig.1 uses a parameter diagram, or P-Diagram [16], to describe how the control and noise parameters affect the performance of a software system. The control parameters could be the number of duplicates of critical components, the sampling rate of sensor components, the priorities of tasks, etc. In comparison, the noise parameters could be the abnormal signals of hardware, unpredictable synchronization behaviors among components, or even the customer’s unexpected practices. The software performance metrics illustrated in the diagram include response time, throughputs, memory space, scalability, or other customized metrics.

![Figure 1. Software System Robustness of P-Diagram](image)

3.2 Evaluation Matrix

As shown in Table 1, the template of an evaluation matrix consists of three regions. The left region contains control parameters (cf), n combinations of the control parameters (cf), and an n × u array of control parameter combination values (cfv). Similarly, the right-upper region contains noise parameters (nf), m combinations of the noise parameters (nfc), and a v×m array of noise parameter combination values (nfv). The right lower region is an array R whose elements r_{ij}, 1 ≤ i ≤ n and 1 ≤ j ≤ m+2, record experiment results and analysis values of a performance metric.
When noise parameters and performance metrics are determined, the levels (test values) for each parameter will be chosen to build the evaluation matrix for performance analysis. If a parameter has continuous values or its values are uncertain in the design phase, the levels could be expressed as interval values [17]. In particular, if control parameter \( c_i \) has \( S_i \) level, \( 1 \leq k \leq u \), and noise parameter \( n_{cf_p} \) has \( T_p \) level, \( 1 \leq p \leq v \), the combinations \( n \) and \( m \) are calculated with Eq.1 and Eq.2.

\[
\begin{align*}
    n &= \prod_{k=1}^{u} S_k \\
    m &= \prod_{p=1}^{v} T_p
\end{align*}
\]

Since the combinations are the arithmetic product of the level number of each parameter, they might become huge, making the evaluation matrix very costly to construct. Fortunately, statistical analysis offers the mechanism to help reducing the size of arrays with orthogonal arrays [16]. The selection of orthogonal arrays is shown in the table 2. Note that each parameter has the same number of levels in table 2. Designers select the same number of levels for each parameter to satisfy orthogonal array conditions. In addition, adjacent parameters with the same levels might share the same orthogonal array. For example, the two and three parameters at level 2 in Table 2 use the common orthogonal array \( L_4 \). The orthogonal array \( L_4 \) has three columns and four rows, and the right-most column is ignored for parameter 2. The orthogonal array can greatly reduce the number of combinations. For four parameters with three levels, for example, the number of combinations is \( 3^4 = 81 \). In orthogonal array \( L_9 \), the number of combinations is reduced to 9, which is around 11% of the original.
3.3 Signal to Noise (S/N) Ratio Check
The diversity of noise parameters is studied by crossing the control parameters array with the noise parameters array. This procedure simulates the variation in the performance due to the noise parameters. After establishing the evaluation matrix as in Table 1, experiments with an analytic or simulated model produce the results to fill in the left region of array \( R = \{ r_{i,j} \} \), \( 1 \leq i \leq n \) and \( 1 \leq j \leq m \). The analytic or simulated model to be used in experiments is the one that has been determined in the second step of performance analysis as described in the introduction section of the paper.

The next column of array \( R \), which consists of elements \( r_{i,m+1} \) for \( 1 \leq i \leq n \), saves the means of each row from \( r_{1,j} \) to \( r_{i,j} \). Based upon the results from analytic or simulated experiments, the robust configuration for the design is finally determined with a signal-to-noise (S/N) ratio check [16]. The signal-to-noise(S/N) ratio is a transformation of the repetition data to another value which measures variations. Typically, three of S/N ratios are applicable depending on the characteristics of the system under evaluation. They are called lower is better (LB), nominal is the best (NB), and higher is better (HB). For some performance metrics in software system, for example, response time and space, LB is appropriate. For other metrics, such as throughput, HB is appropriate. NB is not appropriate in this practice.

Finally, elements \( r_{i,m+2} \), for \( 1 \leq i \leq n \), in the last column of array \( R \) are calculated by the formulas given in Eq.3 for HB or Eq.4 for LB. The S/N ratios take both the mean and the variability into consideration. The robust configuration of a design corresponds to the combination of control parameters that offers the best result to hand noises.

\[
r_{i,m+2} = -10 \log \left( \frac{1}{m} \sum_{t=1}^{m} \left( \frac{1}{r_{i,t}} \right)^2 \right) \quad (3)
\]
\[
r_{i,m+2} = -10 \log \left( \frac{1}{m} \sum_{t=1}^{m} r_{i,t}^2 \right) \quad (4)
\]

4. CASE STUDY: SURGE
This section illustrates the application of the presented method to an existing component-based software system, i.e., Surge. Surge is built on TinyOS, a component-based operating system designed for wireless and embedded sensor networks [18].

4.1 Application Overview
Surge is an application of wireless sensor networks. Each Surge node collects its environment data, such as light readings, and forwards them to the base station. To effectively forward data packets, all Surge nodes form a dynamic routing spanning tree, rooted at the base station. Each node maintains the information of its parent and its own depth. The parent is selected according to link quality and parent’s depth. A Surge node always attempts to forward the packet in shortest path and best quality path.

Surge is a typical component-based software system. It is developed with NesC, which is a component-based language [19] that needs to be translated to C language before compiling to the binary code. As shown in Fig. 2, Surge consists of ten components. Components Photo and Sounder are sensor components that are responsible for collecting the environment data, including lighting and sound. Component LedsC maintains the current working status of nodes. Component GenericCommPromiscuous provides generic miscellaneous services. Component TimerC synchronizes all other components. Component SurgeM provides the basic control flow of the Surge system. Component QueueSend serves as a buffer for packets to be sent out. Components Bcast and MultiHopRouter maintain the routing tree, and forward the packets from its children to its parent. The Main component wraps all components together.

4.2 Robustness Analysis
Performance analysis takes place right after architecture design. For the Surge system, its robustness is defined as the sensitivity degree of its performance metric to the effect of changes in the noise parameters.

4.2.1 Determining Parameters
![Figure 3](image)

As shows in Fig.3, the light and voice signals are inputs to the Surge system from components Photo and Sounder. Packet CRC failure rate is an observable performance metric as it responds to the system’s performance status, such as energy utilization, response time, and the throughput of a single Surge node. Due to the retransmission mechanism of Surge, higher Packet CRC failure rate makes Surge node to consume more energy, take longer response time, and reduce throughput.

There are three control parameters that affect the Packet CRC failure rate of Surge. The first is the sampling rate of components Photo and Sounder as it determines the period between readings. The cycle time of routing protocol in Surge is also limited by the sampling rate [20]. The second is the buffer size of component QueueSend as its increase (or decrease) in the possibility of queue overflow determines the increase (or decrease) of packet loss. The last is the timeout scale of component MultiHopRouter as it decides the interval of retransmission.

Similarly, there are two noise parameters that affect the Packet CRC failure rate of Surge. One is the link quality as it is determined by the uncontrollable environment conditions. The other is network density. This is a parameter that depends on specific applications, and it indicates the distance of sensor nodes. Although network density can be configured, it is an external parameter for the Surge software system located in the sensor node. Additionally, Surge is supposed to behave robustly in
different applications. Thus, network density is regarded a noise parameter in this condition.

4.2.2 Evaluation Matrix

To build evaluation matrix, the assignment of levels for each parameter is as follows:

- 'A': Sampling rate (0.20/0.25/0.33) unit: times/second
- 'B': Buffer Size (15/16/17) unit: packets
- 'C': Timeout-Scale (3/4/5)
- 'a': Link Quality measured by radio model (simple(s)/lossy(l))
- 'b': Network density (10/20) unit: feet

Because there are three control parameters and three levels for each parameter, L9 orthogonal array is suitable. The 'x' mark in Table 3 stands for ignored elements of the matrix. For the noise parameters, their number is two and each parameter has two values. It is therefore not necessary to apply orthogonal arrays to reduce the size of noise parameter array, and the noise parameter combinations are four. The evaluation matrix is shown in Table 3.

Table 3 Evaluation Matrix for Surge

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>a</th>
<th>s / 1</th>
<th>s / 1</th>
<th>s / 1</th>
<th>s / mean</th>
<th>S/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.20</td>
<td>15</td>
<td>3</td>
<td>x</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td>2</td>
<td>0.20</td>
<td>16</td>
<td>4</td>
<td>x</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td>3</td>
<td>0.20</td>
<td>17</td>
<td>5</td>
<td>x</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>15</td>
<td>4</td>
<td>x</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
</tr>
<tr>
<td>5</td>
<td>0.25</td>
<td>16</td>
<td>5</td>
<td>x</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
</tr>
<tr>
<td>6</td>
<td>0.25</td>
<td>17</td>
<td>6</td>
<td>x</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
</tr>
<tr>
<td>7</td>
<td>0.33</td>
<td>15</td>
<td>5</td>
<td>x</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
</tr>
<tr>
<td>8</td>
<td>0.33</td>
<td>16</td>
<td>6</td>
<td>x</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
</tr>
<tr>
<td>9</td>
<td>0.33</td>
<td>17</td>
<td>7</td>
<td>x</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
</tr>
</tbody>
</table>

4.2.3 Signal to Noise (S/N)

A Surge system was built on TOSIM with 5*5 mote grids to observe its behaviors and performance. The system was configured as table 3. It ran on the TOSIM, a discrete event simulator for TinyOS wireless sensor networks. Because Surge with lower Packet CRC loss rates demonstrates better performance, Eq.4 is appropriate in this case. The experiments data and analyzed results are given in table 4. By investigating the results, it is concluded that the fourth combination of control factors is a good candidate for robust design. It is also noted that the selection of levels affects the evaluation results. Architects should guarantee that all selected levels are acceptable for the system under evaluation.

Table 4 Matrix Evaluation Result

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>a</th>
<th>s / 1</th>
<th>s / 1</th>
<th>s / 1</th>
<th>s / mean</th>
<th>S/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.20</td>
<td>15</td>
<td>3</td>
<td>x</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td>2</td>
<td>0.20</td>
<td>16</td>
<td>4</td>
<td>x</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td>3</td>
<td>0.20</td>
<td>17</td>
<td>5</td>
<td>x</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>15</td>
<td>4</td>
<td>x</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
</tr>
<tr>
<td>5</td>
<td>0.25</td>
<td>16</td>
<td>5</td>
<td>x</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
</tr>
<tr>
<td>6</td>
<td>0.25</td>
<td>17</td>
<td>6</td>
<td>x</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
</tr>
<tr>
<td>7</td>
<td>0.33</td>
<td>15</td>
<td>5</td>
<td>x</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
</tr>
<tr>
<td>8</td>
<td>0.33</td>
<td>16</td>
<td>6</td>
<td>x</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
</tr>
<tr>
<td>9</td>
<td>0.33</td>
<td>17</td>
<td>7</td>
<td>x</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
<td>0.017</td>
</tr>
</tbody>
</table>

5. CONCLUSIONS

Presented in this paper is a new approach that applies statistical tools in the application of component-based software performance engineering. This approach helps software architects to configure components in the design stage with robust performance that are insensitive to unexpected environments. The orthogonal arrays could significantly decrease the time and cost on the proposed approach when they are applicable. Although this work could be applied to other software system, it is especially important for component-based systems because such systems are hard to be optimized with compilers. Active investigation is being conducted to make the approach easy for end users with computer assisted interaction and automation.

6. ACKNOWLEDGEMENTS

This work was supported by Natural Sciences and Engineering Research Council (NSERC) of Canada.

7. REFERENCES


