Error Propagation In Software Architectures


Lane Department of Computer Science, West Virginia University
Morgantown WV 26506
{rabie, dmnassar, smark, grani, gunnalan, ammar}@csee.wvu.edu

Bo Yu, A. Mili
College of Computer Science, New Jersey Institute of Technology
Newark NJ 07102
{yubo, mili}@cis.njit.edu

Abstract

The study of software architectures is emerging as an important discipline in software engineering, due to its emphasis on large scale composition of software products, and its support for emerging software engineering paradigms such as product line engineering, component based software engineering, and software evolution. Architectural attributes differ from code-level software attributes in that they focus on the level of components and connectors, and that they are meaningful for an architecture. In this paper, we focus on a specific architectural attribute, which is the error propagation probability throughout the architecture, i.e. the probability that an error that arises in one component propagates to other components. We introduce, analyze, and validate formulas for estimating these probabilities using architectural level information.

1. Introduction

The study of software architectures is becoming an important discipline in software engineering, because software architectures support many emerging paradigms of software development (product line engineering, component based software engineering, COTS-based software development) as well as the increasingly prevalent paradigm of software evolution. The shift from the traditional functional view of software development to the architectural view was first advocated by Shaw in [17], and has been widely recognized/adopted since. As architectures emerge, so does the need to quantify them in a way that reflects their relevant quality attributes; relevant architectural attributes include features of the architecture that have an impact on the quality of software products that are instantiated from it.

In this paper, we study the specific architecture-level attribute of Error Propagation Probability, which reflects the probability that an error that arises in one component of the architecture is propagated (rather than masked) to other components. This study is part of a larger project which investigates a wide range of attributes, including Change Propagation Probability, Requirements Propagation Probability, Diagonality, etc [13].

In the spirit of the Goal/Question/Metric of Basili and Rombach [2], we map our attribute of interest onto a computable metric that can be evaluated on the basis of information that is available at the architectural level. At this level, we do not usually have the wealth of structural and semantic information that is available at the code level, but we do have information about the flow of control and data within components and between components. This precludes using traditional software metrics, which are based on such code-level features as tokens [6], flow graphs

1  This work is supported by the National Science Foundation through ITR program grant No CCR 0296082, and by NASA through a grant from the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) managed through the NASA Independent Verification and Validation (IV&V) Facility, Fairmont, West Virginia.
2  Correspondence author. E-mail: rabie@csee.wvu.edu
data dependencies [3] and control dependencies, to mention a few.

Our approach can further be characterized by a combined Bottom-Up/Top Down discipline, whereby we complement the top-down approach advocated by Basili and Rombach’s Goal/Quality/Metric paradigm with a bottom-up approach that analyzes the architecture and derives a matrix that quantifies the flow of information within components and between components of the architecture. In the absence of structural and semantic information, we cannot analyze the flow of information within an architecture deterministically, hence we resort to a stochastic approach. This stochastic approach captures information flow within the architecture by means of random variables, and quantifies this flow by means of entropy functions applied to these random variables [15], [16], [18].

Software metrics has been a very active field for several decades, and it is impossible to do justice to all the relevant work in this area. We will briefly mention some of the research efforts that we find most closely related to ours, highlighting in what way we are different. Many authors have used information theory to derive software metrics. Some, such as Allen and Khoshghoftaar [9], Chapin [4], and Harrison [7] do so explicitly; others, such as Halstead [6], do not invoke information theory explicitly, but their metrics can be interpreted in terms of this theory. Whereas all these authors define metrics by interpreting the source text of the program as the message, we derive metrics by interpreting information flow throughout the architecture as messages whose entropy we are estimating.

Basili and Rombach [2] introduce an analytical paradigm for the derivation of software metrics, called the Goal/Question/Metric paradigm. It is based on a systematic, goal-oriented, procedure for the derivation of software metrics. In [5], Fenton presents a Representational Theory of Measurement, and argues that software metrics work must adhere to it. Also, he evaluates various metrics efforts with respect to the guidelines of this theory, and argues that Basili’s GQM is but an instance of it.

In our work we combine the top-down goal oriented approach advocated by Basili and Rombach [2] and Fenton [5] with a bottom-up approach, which analyzes the architecture and produces a matrix of information theoretic measures, where each cell in the matrix is associated either with a component (for diagonal elements) or with a connector (outside the diagonal) in the architecture. These measures are used for the purposes of estimating error propagation. They can also represent other quantitative measures that are beyond the scope of this paper such as change propagation and requirements propagation.

In [19], Voas analyzes error propagations between COTS components and presents an automated tool to simulate error propagation, which is used to deploy a fault injection experiment. Michael et al [12] present an empirical study of data state error propagation behavior. The authors argue that at a given location either all data state errors injected tend to propagate to the output, or else none of them does. In [8] Hiller et al analyze error propagation conceptually, introducing the concept of error permeability and discussing means to measure it using fault injection techniques.

In this paper, we focus our attention on the study of error propagation probabilities in an architecture. Given an architecture made up of N components, we derive an NxN matrix whose entry at row A and column B contains the probability that an error in component A propagates to component B at run time. If we claim that this quantitative function is related to dependability, we do not mean that we represent dependability by this function; we merely mean that if we are interested in the attribute of dependability, then we may want to look at this function as it reflects some relevant aspect of dependability.

The contributions of this paper are outlined as follows:

- We introduce the definition of error propagation between two components of an architecture.
- We derive an analytical expression for estimating error propagation probabilities.
- We derive an upper-bound on the error propagation probability, which can be estimated using architectural information.
- We present an empirical approach for measuring error propagation probabilities and use it to validate the analytical measures.

This paper is organized as follows. In section 2, we introduce the quantitative function of Error Propagation Probability then we discuss how to evaluate the unconditional probability, and in section 3 we discuss analytical means to derive the error
propagation matrix of a software product from architectural information. In order to validate our analytical formula, we consider a sample software architecture, which we analyze to derive its error propagation matrix, in section 4; then, in section 5, we run a fault injection experiment on the system and observe the actual error propagations; in section 6 we compare our analytical results against our empirical observations and assess the statistical validity of our formulas. We conclude in section 7 by summarizing our results and sketching future directions of research.

2. Error Propagation Probabilities

In this section, we first introduce and discuss (in section 2.1) the feature of error propagation in an architecture. Then we review a derivative of this feature (in section 2.2).

2.1. Error Propagation: Definition

We consider two components, say A and B, of an architecture, and we let X be the connector that carries information from A to B; for the purposes of our current discussion, the specific form of connector X is not important, we will merely model it as a set (of values that A may transmit to B). Also, the specific form of components A and B is not important for the purposes of our discussion; we will merely model them as functions that map an internal state and an input stimulus into a new state and an output.

Definition 1. The Error Propagation Probability from component A to component B is denoted by EP(A,B) and defined by:

\[ EP(A,B) = \text{Prob}(B(x) \neq B(x') \mid x \neq x'), \quad (1) \]

where [B] denotes the function of component B, and x is an element of the connector X from A to B. We interpret [B] to capture all the effects of executing component B, including the effect on the state of B and the effect on any outputs produced by B.

We interpret EP(A,B) as the probability that an error in A is propagated by B (as opposed to being masked by B) because the outcome of executing B will be affected by the error in A. By extension of this definition, we let EP(A,A) be equal to 1, which is the probability that an error in A causes an error in A. Given an architecture with N components, we let EP be an N×N matrix such that the entry at row A and column B be the error propagation probability from A to B.

Note that nothing in our definition above indicates that \( x' \) is an erroneous message; all the definition says is that \( x' \) is different from \( x \) --- as far as this definition is concerned, both could be correct. While this may seem to be an anomaly, all it means is that we are measuring error propagation probabilities by a wider property, which is the probability that different arguments are mapped by function \([B]\) to different images (a measure of injectivity of \([B]\)).

2.2. Unconditional Error Propagation

Note that the definition of the error propagation given above uses the concept of conditional probability, i.e. we calculate the probability that an error propagates from A to B under the condition that A actually transmits a message to B. It is often useful, however, to use the unconditional error propagation which we will denote simply as \( E(A,B) \), and define as the probability that an error propagates from A to B not conditioned upon the event that A sends a message to B. Function \( E(A,B) \) is clearly dependent on \( EP(A,B) \), but it further integrates the probability that A does send a message to B.

In order to bridge the gap between the original (conditional) error propagation and the newly introduced unconditional error propagation, let us consider the transmission probability matrix \( T \) where the entry \( T(A, B) \) reflects the probability with which the connector \((A \rightarrow B)\) gets activated during a typical/canonical execution. \( T \) is the N×N matrix whose entry \( T(A, B) \) is the probability that the component \( A \) sends a message to component \( B \) given that the \( A \) is expected to transmit a message to some component. Note that:

- It is reasonable to assume that \( T(A, A) = 0 \) for all components \( A \),
- Clearly, \( T \) is a stochastic matrix, i.e. \( \sum_B T(A, B) = 1 \) for every component \( A \).

The matrix \( T \) is used to distinguish between a connector that is invoked intensively in each execution and one that is invoked only occasionally, under exceptional circumstances. The matrix \( T \) reflects the variance in frequency of activations of different connectors during a typical execution.
By virtue of simple probabilistic identities, we find that the unconditional error propagation is obtained as the product of the conditional error propagation probability with the probability that the connector over which the error propagates is activated, i.e.

\[ E(A, B) = EP(A, B) \times T(A, B). \]  

(2)

3. Estimating Error Propagation

We have found that analytically, the error propagation probability (as defined in Section 2), can be expressed in terms of the probabilities of the individual A-to-B messages and states, via the following formula:

\[ EP(A \rightarrow B) = \frac{1 - \sum_{x \in S_B} P_B(x) \sum_{y \in S_A} P_{A \rightarrow B}[F^{-1}_x(y)]^2}{1 - \sum_{v \in V_{A \rightarrow B}} [P_{A \rightarrow B}[v]]^2} \]  

(3)

where \( F^{-1}_x(y) = \{ v \in V_{A \rightarrow B} \mid F_x(v) = y \} \), and we assume a probability distribution \( P_B \) on the set of states \( S_B \) of component \( B \), and a probability distribution \( P_{A \rightarrow B} \) on the set of messages \( V_{A \rightarrow B} \) passed from \( A \) to \( B \).

The term \( \sum_{v \in V_{A \rightarrow B}} [P_{A \rightarrow B}[v]]^2 \) in the denominator of (3) is an exponent of the 2nd order Renyi entropy [15], which according to the recent studies [20] is closely related to the classical Shannon entropy [16]. If we assume that the states of \( B \), as well the messages passing through the connector from \( A \) to \( B \) are equiprobable, then the formula (3) for error propagation is simplified into

\[ EP(A \rightarrow B) = \frac{1}{|S_B| |V_{A \rightarrow B}|} \left[ \sum_{x \in S_B} \sum_{y \in S_A} |F^{-1}_x(y)|^2 \right] \]  

(4)

Notice that the inequality (5) can be used as a close approximation of the actual \( EP(A \rightarrow B) \) value whenever for every initial state of \( B \) the number of messages that trigger its transition to a new state is approximately the same, no matter what that new state is.

4. Case Study

4.1. Command and Control System Specification

The example we use to illustrate our work is a large command and control system that is used in a life-critical, mission-critical application. This system was modeled using the Rational Rose Realtime CASE tool [14]. It is a Computer Software Configuration Item (CSCI) that provides the following functions:

- Facilitating Communication, Control, Cautions and Warnings including subsystem Configuration Management, C&DH (Communication and Data Handling) Communications Control, Processing, Memory Transfer, C&DH Failure Detection, Isolation, and Recovery and Time Management,
- Controlling a Secondary Electrical Power System, and
- Environmental Control, which provides Temperature and Humidity Control.

We will concentrate on the Thermal Control part of the system, which is a rather complex system with operations setting controller, fault recovery procedures, and pump control functionalities. The system is responsible for providing overall management of pumps and performing the necessary monitoring and response to sensors data. Furthermore, it is responsible for performing automated startup, and controlling Thermal System reconfigurations.
During each execution cycle, a check is performed for incoming commands. Received commands are validated in the same execution cycle. Mode change commands, which will reconfigure the Internal Thermal System, are also accepted from other components of Thermal System to compensate for system component failures or coolant leaks. A failure recovery system detects failure conditions and performs recovery operations in response to the detected failures. Failure conditions include combinations of Pump failures and Shutoff Valve failures.

The software architecture of this system is shown in Figure 1. Using these artifacts, one can identify the components and the connectors that describe the components-based system architecture and label the EP matrix rows and columns with the components names.

Table 1 shows a sample sanitized message protocol between a pair of components in our system. This artifact provides us with the message set \( V_{A \rightarrow B} \) and \( V_{B \rightarrow A} \) that is going between the two components A and B. Similarly, using the Rose-RT tool or similar tools, we can extract all the sets of messages that are exchanged between each pair of components in the system.

The state chart shown in Figure 2 is a sample of state chart of a component in the system. This provides us with the state set \( S_{B} \) for this sample component. Using the Rose-RT tool or similar tools, we can identify the triggering messages from one state to another. In a similar way, one can get all the state sets for all the components.

Figure 1: Software architecture of the system.
Table 1: A sample of a sanitized message protocol (components 1 and 5).

<table>
<thead>
<tr>
<th>In Signals</th>
<th>Out Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Msg 1 (C1_C5)</td>
<td>Msg 1 (C5_C1)</td>
</tr>
<tr>
<td></td>
<td>Msg 2 (C5_C1)</td>
</tr>
<tr>
<td></td>
<td>Msg 3 (C5_C1)</td>
</tr>
<tr>
<td></td>
<td>Msg 4 (C5_C1)</td>
</tr>
<tr>
<td></td>
<td>Msg 5 (C5_C1)</td>
</tr>
</tbody>
</table>

Figure 2: A state diagram of a component 5

4.2. Analytical Error Propagation Results

Considering the CSCI system discussed above, we get the set of states $S_B$ and messages $V_{A\rightarrow B}$ from the artifacts of the system specification. We obtain the matrix $EP$ of (conditional) error propagation probabilities of this system, using the approximation (5). We assume equi-probability of states and messages.

In order to demonstrate how to compute one value of $EP(A,B)$, let $A=1$ and $B=5$. Component 5 has $S_B = 2$ from Figure 2, and $V_{A\rightarrow B} = 5$ from Table 1. So using the approximation (5), we get $EP(A,B) = (1-0.5)/(1-0.2) = 0.625$. Thus, the error propagation from component 1 to component 5 cannot exceed 0.625.

For this particular case study, we have derived the connector activation matrix $T$ as a stochastic matrix of probabilities that contains for each entry $(A,B)$, the probability that connector $(A,B)$ is activated, given that component $A$ is broadcasting a message. Using
this connector activation matrix, we derive the *unconditional error propagation matrix* \( E_A \), also referred to as the 1-step error propagation matrix of the system. We get the matrix \( T \) through a simulation of the system representing the operational profile of the execution.

Continuing our example, we found \( T(A, B) = 0.023 \). So, the probability that connector \((1,5)\) is activated, given that component 1 is broadcasting a message is 0.023. Then, the *unconditional error propagation* \( E_A(A, B) = 0.625 \times 0.023 = 0.0146 \).

Table 2 shows the non-trivial values of error propagation (\( E_A \) values) that we derived analytically on our case study.

### 5. Empirical Error Propagation Results

In order to validate our analytical study, we developed a framework for experimental error propagation analysis in which we utilize fault injection experiments to alter architecture specifications. We then simulate the corrupted specifications and record component traces as “faulty-run” logs. Finally, we compare the faulty-run logs against a fault-free “golden-run” log obtained by simulating the uncorrupted architecture specifications [8]. We perform the simulation-based error propagation analysis in two phases: an acquisition phase and an analysis phase.

**In the acquisition phase:**
- We extract architecture information about the components and connectors that make up the software system.
- We use a message swapping fault model [1] to generate fault injection experiments. In each of the fault injection experiments, we replace all occurrences of a message nominally flowing over a connector (from component A to component B) by a different message (as a result of an error in component A) that belongs to the set of messages that A may send to B.

We simulate the corrupted specifications and record simulation traces for the different experiments that cover all messages for the different connectors present in the architecture.

**In the analysis phase:**
- We conduct post-simulation comparison between the faulty-run logs and the reference (fault-free) log. The comparisons are based on state transitions at simulation time instances following a fault injection. Immediately before injection of a fault, there is no difference between the state of component B as recorded on the reference log and its state recorded on the faulty log. After a fault is injected, any discrepancy between the two logs is due to error propagation from A to B. A faulty-message propagating (from A to B) will at most cause a single instance of error propagation (from A to B).
- We compute the experimental error propagation probability from component A to B as the ratio of the fault injections (corresponding to errors in A) that propagate to component B over the total number of faults injected from component A to component B.

Table 2 shows the error propagation probabilities (\( E_E \) values) that we have empirically observed for the non-trivial connectors of the architecture considering the same mode of operation.

### 6. Correlating Analytical and Empirical Results

In this section, we confront the results computed by the analytical formula of section 3 against the results derived from the fault injection experiment of section 5 to assess the validity of our analytical formulas. We let \( E_A \) and \( E_E \) be (respectively) the analytical matrix and the empirical matrix of (unconditional) error propagation for our sample architecture; these are both 10×10 matrices. We use a number of criteria to this effect:

- The first possible criterion is simply the correlation between the entries of the two matrices; because these matrices contain 100 values each, the correlations do bear some significance.
- The second possible criterion is to correlate, not all the values of the matrices, but rather the non-trivial values (other than those that are either 0 or 1 by definition); the rationale behind this criterion is that trivial values do not really test our analytical results.
The third criterion discriminates between empirical values that were derived from a small number of fault injections and those that were derived from a large number of fault injections. If our analytical results are accurate, we should find empirical values that stem from large numbers of fault injections to be highly correlated to their corresponding analytical values, whereas those values than stem from small numbers of fault injections are not guaranteed to correlate to their corresponding analytical results.

We apply these criteria to E matrices in order to determine to what extent the values obtained empirically are consistent with those found analytically. The correlation coefficient between all the cells of the analytical $E_A$ matrix and the experimental $E_E$ matrix is:

$$\text{Cor}(E_A, E_E) = 0.628 \quad (\text{r value}) \quad (6)$$

where ‘r’ denotes the Pearson product-moment correlation coefficient.

We note, however, that there are only 15 non-trivial entries in each of the two matrices. Trivial entries correspond to self-loops from a component to itself (with error propagation probability of 1 by definition) and to the non-directly connected components (with error propagation probability of 0 by definition). It may be useful to evaluate the correlation between the set of non-trivial values of matrices $E_A$ and $E_E$ having a significant number of fault injections (>20). The connectors that has fewer than 20 fault injections are shaded in Table 2. We find:

$$\text{Cor}'(E_A, E_E) = 0.5576 \quad (\text{r value}) \quad (7)$$

Table 2 contains the 15 non-trivial entries corresponding to the 15 connectors over which faults were injected during the controlled experiment. Note that the number of injected faults over connectors varies considerably across the entries. The connectors in the table follow a descending order with respect to the number faults injected over each connector.

Overall, the correlation decreases as the number of injected faults drop, although not monotonically. The results in this table are interesting, in that they show a fairly high correlation between experimental results and analytical results in those cases where the experimental result is based on a large number of fault injections. Also, predictably, the correlation drops (as shown in Table 2) as the number of fault injections drop (though not monotonically).

### Table 2: Correlation between analytical and experimental EP probabilities

<table>
<thead>
<tr>
<th>Entry</th>
<th>Connector From</th>
<th>Connector To</th>
<th>$E_A$</th>
<th>$E_E$</th>
<th># Injected Faults</th>
<th>Correlation Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>0.0146</td>
<td>0.1331</td>
<td>1067</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>6</td>
<td>0.0145</td>
<td>0.1280</td>
<td>1055</td>
<td>1.0000 Entries 1 through 2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0.0132</td>
<td>0.0557</td>
<td>592</td>
<td>0.9999 Entries 1 through 3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0.0060</td>
<td>0.0161</td>
<td>559</td>
<td>0.8708 Entries 1 through 4</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>1</td>
<td>0.3761</td>
<td>0.7500</td>
<td>64</td>
<td>0.9894 Entries 1 through 5</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>4</td>
<td>0.0102</td>
<td>0.4912</td>
<td>57</td>
<td>0.8160 Entries 1 through 6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1</td>
<td>0.1104</td>
<td>0.7838</td>
<td>37</td>
<td>0.7153 Entries 1 through 7</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2</td>
<td>0.0012</td>
<td>0.5000</td>
<td>36</td>
<td>0.6488 Entries 1 through 8</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>2</td>
<td>0.1026</td>
<td>0.6429</td>
<td>28</td>
<td>0.6433 Entries 1 through 9</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>2</td>
<td>0.2024</td>
<td>0.7083</td>
<td>24</td>
<td>0.6829 Entries 1 through 10</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>1</td>
<td>0.0107</td>
<td>0.7917</td>
<td>24</td>
<td>0.5576 Entries 1 through 11</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>8</td>
<td>0.1264</td>
<td>0.4286</td>
<td>7</td>
<td>0.5501 Entries 1 through 12</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>2</td>
<td>0.1005</td>
<td>1.0000</td>
<td>4</td>
<td>0.5068 Entries 1 through 13</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>2</td>
<td>0.0506</td>
<td>1.0000</td>
<td>4</td>
<td>0.4291 Entries 1 through 14</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>1</td>
<td>0.0014</td>
<td>1.0000</td>
<td>4</td>
<td>0.3240 Entries 1 through 15</td>
</tr>
</tbody>
</table>
We have computed the value of t statistic [11] for the non-trivial values of analytical and experimental error propagation matrices $t_{ob} = 2.015$ (n=11), and the corresponding $P < 0.05$; whence we infer that the correlation of 0.628 is statistically significant. The T-test results showed that there is a linear association between analytical error propagation and experimental error propagation.

7. Conclusion

In this paper, we have derived an analytical approach to estimate the probability of error propagation between components in a software architecture. Further, we have illustrated our proposed formula by means of a fault injection experiment, applied on a large command and control system, and found a fairly meaningful correlation between our analytical estimates and our experimental observations.

Given that our analytical approach is based on architecture specifications, and uses exclusively information that is typically available at an architectural level, we submit that our result can be used to estimate the error propagation behavior of an architecture, at a time when relatively little is known about the actual execution of products that instantiate the architecture. In addition to providing the basic conditional probability of error propagation over a given connector (conditioned on the activation of the connector), we have also provided analytical formulas for unconditional error propagation (which incorporate the probability of connector activation). Finally we have briefly explored ways for a software architect to analyze and use the information provided by our analytical estimates.

Among our venues of further research, we are considering to carry out more experiments on validating our analytical formulas, our future validation studies will also consider comparison between error propagation data obtained from the field for systems that were deployed against our analytical estimates of error propagation probabilities that could have been computed for these systems from their architecture specifications. We are also considering to augment our architectural analysis tool to support the automatic computation of error propagation probabilities.

This work is part of a wider effort to analyze and quantify attributes of software architecture. Attributes of interest include, among others, change propagation probabilities (likelihood that a change in one component affects other components), and requirements propagation probabilities (likelihood that a change in the requirements of one component mandate a change in the requirements of other components). Work is continuing on these other attributes.

References


