Predicting Requirements Change Propagation Based on Software Architecture

Yun Fu

Minqiang Li

Fuzan Chen

PREDICTING REQUIREMENTS CHANGE PROPAGATION BASED ON SOFTWARE ARCHITECTURE

Yun Fu, Minqiang Li and Fuzan Chen
School of Management, Tianjin University
Tianjin 300072, P. R. China
E-mail: fy8266@163.com, mqli@tju.edu.cn, fzchen@tju.edu.cn

Abstract

Change propagation is a central issue in software development process. In early stages of software development, software architecture facilitates the component-based software development process and provides a platform for prediction of requirements change. This paper aims to predict change propagation in early stages of software development, and evaluate the architecture based on architectural pattern. In order to achieve this goal, the change propagation probability is formally defined, and the change propagation in five architectural patterns is discussed. Moreover, change propagation density is defined to extend the pattern-based propagation, which incorporates design metrics into software architecture evaluation. The efficiency of the proposed method is demonstrated through a computational experiment.

Keywords: software architecture, requirements change propagation, architectural pattern

1. Introduction

Software architecture plays an important role in the software development process. It explicates the structure of the software system in terms of a collection of interacting components to accomplish the required tasks [1]. Software architecture facilitates the component-based software development process and provides a platform for the prediction of requirements change. This paper focuses on requirements change propagation that is a central issue of software development [2] based on architectural view,

Software requirements always evolve throughout the whole software development process, which is recognized to be a major source of software development risk [3]. Due to the direct/indirect impacts of changes and a lot of iterative activities, the development schedule is prolonged and the project cost overruns the budget. However, the impacts of changes are hard to be predicted and quantified due to the uncertainty that is associated with requirements change and entails the risk of the project [4]. Requirements change usually leads to the ripple changes among connected components, which means that a change arising in requirements can propagate to the components that are related directly or indirectly. Change propagation researches and literatures cover change management, engineering design, product development, complexity, graph theory and design for flexibility [5]. This research aims to predict the change propagation probability based on software architecture in early stages of software development (including requirements specification and design), which reflects the probability that a change originating in one component of software architecture propagates to other components.

The rest of paper is organized as follows. Section 2 discusses the change propagation process, and defines the change propagation probability in a formal format. Section 3 deduces a computation formula to estimate the approximate change propagation probability, whose variables can be calculated through UML models of the software architecture. Considering the characteristics of different architectural patterns, section 4 discusses the change propagation probability based on architectural pattern. Section 5 demonstrates the proposed method via a computational experiment, and section 6 summarizes the paper.

2. Change Propagation

2.1 Change Propagation Process

Due to the interdependencies among the entities of a software system, it is difficult for developers to manage change propagation in software development process accurately. Failure to update any of the related entities would cause system inconsistencies. In early stages of software development, software architecture is the most important entity of a system. When requirements of stakeholders change, designers should determine the initial components in the architecture that must be changed. Once the designs of the initial components are changed, designers should determine whether the changes propagate to other components, and modify them if so. This process is repeated until every change for each component has been checked out. After that, designers should check the propagation with requirements to ensure that the changed architecture is consistent with the changed requirements. If there are inconsistencies, the change propagation process is repeated. Fig.1 shows the model of change propagation process.

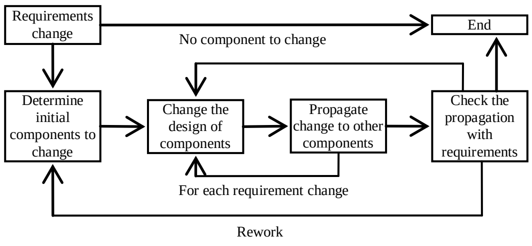
Requirements change
End
No component to change
Determine initial components to change
Change the design of components
Propagate change to other components
Check the propagation with requirements
For each requirement change
Rework

Fig. 1 Model of change propagation process

The first step of predicting change propagation within the architecture is to understand the change propagation between a given pair of components. This paper proposes a probabilistic method based on UML diagrams to estimate the probability that each change will propagate.

2.2 Change Propagation Probability

Software architecture is composed of components and interactions among them, where component interactions are mainly composed by the transfers of information and data. Traditional software metrics usually adopted tokens, flow graph, data dependencies and control dependencies base on code level features [6]. But in early stages of software development process, it is impossible to obtain efficient information that is available at the code level. Instead, information represented at the architectural level is taken into account. In order to predict change propagation probability of the system, we first calculate the propagation between a given pair of components A and B.

Several semantic and formal definitions of change propagation have been put forward [5] [6] [7] [8] [9]. This paper adopts the formal definition to interpret this computable metric.

Suppose that the software architecture has n components, in which components A and B are a pair of the components that communicate through a connector. The component modeled by state transition maintains a series of internal states that reflect the invocation, as in Fig. 2.

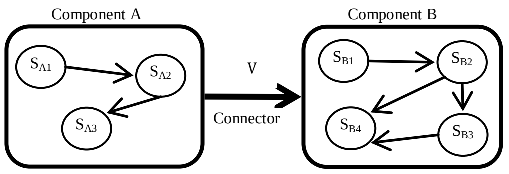

Fig. 2 States transition for components A and B

Let V be the range of values or messages which transmit from A to B, S_A be the set of states of component A, and S_B be the set of states of component B. When component B receives a value $v \in V$ from component A, it changes its internal state and produces some outputs which will propagate to other components. Function $f : S_B \times V \to S_B$ defines this state transition. $y_B = f(x_B, v)$ ($f_B(v)$ for simple) represents the state of component B after receiving $v \in V$, where x_B is the state of component B before transmission, and y_B is the state after transmission. Following a change request, the state of component A, S_A changes into $S_A^{'}$ and the outputs of A also change to $v^{'}$ in order to accommodate the variation. Suppose that for a fixed initial state x_B, the final state resulting from unchanged value v is different from that resulting from changed value $v^{'}$, which implies that $f_B(v) \neq f_B(v^{'})$ infers $v \neq v^{'}$ for any x_B. Under this situation, how the state of component B changes after receiving $v^{'}$ is the key problem in predicting change propagation.

We define the change propagation probability from A to B as a conditional probability which reflects the likelihood that a change occurring in A causes a variation of the state of B. It is denoted by $cpp(A,B)$.

$$cpp(A,B) = P(S_B \neq S_B^{'} \mid S_A \neq S_A^{'}) \quad (1)$$

In fact, $cpp(A,B)$ is under the situation that A changes its state and indeed transmits a value to B. Let $t(A,B)$ be the reachability probability from A to B. Then, unconditional change probability denoted by $cp(A,B)$ is:

$$cp(A,B) = cp\ p(A,B) \cdot t(A,B) \quad (2)$$

$cp(A,B)$ is unconditional probability of change propagation which integrates $cpp(A,B)$ with the probability that A transmits a message to B.

The impact of change between components may propagate directly or indirectly. Since $cp(A,B)$ only represents the direct propagation from A to B, indirect propagation between components must be also considered.

The n-step change propagation probability from A to B is the probability that a change occurring in component A propagates to component B after n transmissions through $n-1$ other different components.

$$\begin{aligned} cp_n(A,B) = cp(A,C_1) \cdot cp(C_1,C_2) \cdot \ldots \cdot \\ cp(C_{n-2},C_{n-1}) \cdot cp(C_{n-1},B) \end{aligned} \quad (3)$$

It is noticed that this formula does not contain the propagation along the path including loops.

After calculating all the n-step change propagation probabilities for each pair of the components, the cumulative change propagation probability which is the sum of the direct and indirect propagation can be obtained.

3. Estimating Change Propagation Probability

In general, there is a tendency for designers to rely on accumulated experience and statistical data to determine the change propagation probability between components. This section uses the theory of probability and statistics to access the approximate value of that.

Assume that the changed value v' and unchanged value v are both in the set V. According to the discussion about change propagation above and the definition of conditional probability, it is easy to see that:

$$\begin{aligned} cpp(A,B) &= P(f_B(v) \neq f_B(v') \mid v \neq v', v \in V, v' \in V) \\ &= \frac{P(f_B(v) \neq f_B(v'))}{P(v \neq v')} \\ &= \frac{1 - P(f_B(v) = f_B(v'))}{1 - P(v = v')} \\ &= \frac{1 - \sum_{x_B \in S_B} P(x_B)(\sum_{v_B, v'_B \in V} P(v_B) \cdot P(v'_B))}{1 - \sum_{v, v' \in V} P(v) \cdot P(v')} \end{aligned} \tag{4}$$

Where, $v_B = \{v \in V \mid f(x_B, v) = y_B\}$ and $v'_B = \{v' \in V \mid f(x_B, v') = y'_B\}$. $P(x_B)$ reflects the probability that the component B is in state $x_B \in S_B$. $P(v)$ is the probability that unchanged value v is received by B. $P(v_B)$ represents the probability that $v_B \in V$ causes B to transit from x_B to y_B. In the formula (4), it is assumed that the changed value $v' \in V$ is statistically independent of the unchanged value v.

Suppose that the changed value v' has the same probability distribution with unchanged value v, the formula (4) can be written as:

$$cpp(A,B) = \frac{1 - \sum_{x_B \in S_B} P(x_B)[\sum_{v_B \in V} P^2(v_B)]}{1 - \sum_{v \in V} P^2(v)} \tag{5}$$

It is obviously that the value of $\sum_{v \in V} P^2(v) \in (0,1)$ will be minimum if and only if $P(v)$ is uniform distribution, and its value is $\frac{1}{|V|}$, where $|V|$ is the number of the values transmitted from A to B. Assume that the value v and the state S_B are both distributed according to uniform distribution. Then, formula (5) can be written as:

$$cpp(A,B) = \frac{1 - \frac{1}{|S_B|} \sum_{x_B \in S_B} \sum_{v_B \in V} P^2(v_B)}{1 - \frac{1}{|V|}} \tag{6}$$

Where $|S_B|$ is the number of the states that component B has. When software architecture is described by UML, all the available models such as architecture diagram based on components, state charts of the component and sequence charts can be applied in calculating change propagation probability. The determination of $P(v_B)$ in formula (6) will use state charts and sequence charts. Firstly, count the values that cause state transmission from x_B to y_B in state charts. Next, estimate the probability distribution by tracing paths of messages transmission from A to B in sequence charts.

Completing all the change propagation probabilities for each pair of the components, a $n \times n$ matrix of change propagation probability $CPP = [cpp_{ij}]_{n \times n}$ for the software architecture is obtained. When $i = j$, $cpp_{ii} = 1$, which means that a change occurred within the component A is determined.

Similarly, a $n \times n$ reachability probability matrix $T = [t_{ij}]_{n \times n}$ can also be obtained. Using the models of UML, it is easy to get the information about transitive messages. Assume that the transitive variable v is distributed according to uniform distribution. Then,

$$t(A,B) = \frac{|V|}{\sum_{k=1}^{n} |V|_{Ak}} \tag{7}$$

Where $|V|$ represents the number of variables transmitted from A to B, n is the number of the architectural components, and $|V|_{Ak}$ is the number of variables transmitted from A to component $C_k (C_k \neq A)$.

According to the discussion above, the unconditional change propagation probability of the software architecture is:

$$CP(cp_{ij}) = cpp_{ij} \cdot t_{ij} \tag{8}$$

4. Pattern-based Change Propagation

We have discussed the component-based change propagation above. For a complex architectural environment, the attributes of different architectural patterns must be also taken into account. Architectural pattern is "a description of element and relation types together with a set of constraints on how they may be used" [10]. Choosing an architectural pattern is often the developers' major design choice. So, it is necessary for designers to yield a better estimation by predicting change propagation of different architectural pattern. In this section, five kinds of patterns [11], including sequential, branching, parallel, pipe-filter, and fault tolerance patterns are discussed.

4.1 Sequential and Branching Pattern

The components in a sequential pattern are executed in a sequential order, as Fig. 3. In branching pattern, the execution only go to one of its branching subsequent components. Both of these two patterns are similar in the characteristics that only one component is executed at a time, and the component proceeds until last component has completed.

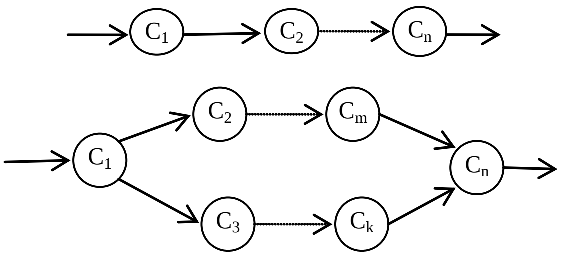

Fig. 3 Sequential and branching pattern

For the purpose of describing conveniently, we only consider the simple environment in which each component is regarded as a module. Let matrix $CMP(cmp_{ij})$ be the change propagation probability based on architectural pattern. For architecture with n components, the change propagation probability in sequential or branching pattern is:

$$cmp_{ij} = \begin{cases} cpp_{ij} \cdot t_{ij},\ M_i \text{ can reach } M_j \\ 0,\ M_i \text{ can not reach } M_j \end{cases},\ \text{for } 1 \le i, j \le n \tag{9}$$

4.2 Parallel and Pipe-filter pattern

In concurrent execution environment, components are usually executed simultaneously, like parallel pattern and pipe-filter pattern. These two patterns have similar description showed as Fig. 4.

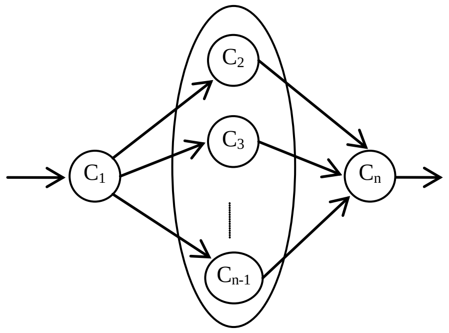

Fig. 4 Parallel, pipe-filter pattern

As Fig. 4, the architecture consists of n components, where $n-2$ components are executed concurrently. With the feature that all the concurrent components (from C_2 to C_{n-1}) should be performed successfully before executing the next component (C_n), the set of concurrent components can be considered as a module, which conforms to the essence of simultaneous transition of concurrency. Fig. 5 shows the modules of the architecture, where components C_2 to C_{n-1} are congregated into the module M_2, C_1 and C_n are modeled as M_1 and M_3 respectively. Then, the parallel pattern is converted to a sequential pattern based on architectural module.

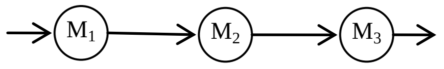

Fig. 5 Modules of the architecture

We use the simple model with 3 modules (showed as Fig. 5) to illustrate the change propagation probabilities between different modules. Let $cmp(i,j)$ be the change propagation probability from module M_i to module M_j. Then,

$$\begin{cases} cmp(1,2) = \sum_{k=n}^{n-1} cpp_{1k} \cdot t_{1k} \\ cmp(2,3) = 1 - \prod_{k=2}^{n-1} (1 - cpp_{kn}) \cdot t_{kn}, \\ \qquad C_k \in M_2 \text{ and } 1 \le k \le n \\ cmp(i,j) = 0, M_i \text{ can not reach } M_j \\ \qquad \text{and } 1 \le i, j \le 3 \end{cases} \tag{10}$$

4.3 Fault tolerance pattern

Fault tolerance pattern consists of a primary component and a set of backup components. The assignment of these components is similar with the parallel pattern showed in Fig. 4. The difference is that in fault tolerance pattern, the components is not executed concurrently as in parallel pattern, but one component works at a time until it fails, then another component will take over.

As Fig. 4, when primary component C_2 is activated, other backup components are inactive. The change propagation probability between M_2 and M_3 is $1-(1-cpp_{2n})\cdot t_{2n}$. When primary component C_2 is failure, the backup component C_3 is activated. Once C_2 and C_3 both fail, the backup component C_4 is activated, and so on. We can get the $cmp(i,j)$ in fault tolerance pattern.

$$\begin{cases} cmp(1,2)=\sum_{k=2}^{n-1} cpp_{1k}\cdot t_{1k} \\ cmp(2,3)=1-(1-cpp_{2n})\cdot t_{2n}- \\ \quad \sum_{q=2}^{n-2}[\prod_{k=2}^{q}(1-t_{kn})]\cdot(1-cpp_{k+1,n})\cdot t_{k+1,n} \\ \quad ,C_k\in M_2 \\ cmp(i,j)=0, M_i \text{ can not reach } M_j \\ \quad \text{and } 1\le i,j\le 3 \end{cases} \tag{11}$$

5. Computational experiments

Previous section has shown how to compute the change propagation probability of an architecture based on architectural pattern. Following experiment demonstrates it.

Fig. 6 represents the architecture with 10 components, where C_2 and C_3 are parallel, and C_7 is the backup component of C_6. Other components are executed in sequential or branching pattern.

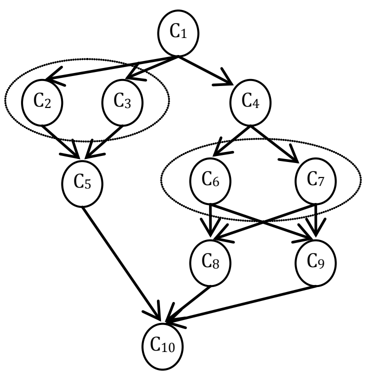

Fig. 6 Example of Architecture

Suppose that matrix of change propagation probability $CP(cp_{ij})$ is calculated out as follow.

$$\begin{array}{c|cccccccccc} C_1 & 1 & 0.084 & 0.022 & 0.312 & 0 & 0 & 0 & 0 & 0 & 0 \\ C_2 & 0 & 1 & 0 & 0 & 0 & 0. & 0\,2 & 0 & 0 & 0 \\ C_3 & 0 & 0 & 1 & 0 & 0.47 & 0 & 0 & 0 & 0 & 0 \\ C_4 & 0 & 0 & 0 & 1 & 0 & 0.14 & 0.065 & 0 & 0 & 0 \\ C_5 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0.34 \\ C_6 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0.265 & 0.225 & 0 \\ C_7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.23 & 0.115 & 0 \\ C_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0.12 \\ C_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.26 \\ C_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Since $CP(cp_{ij})$ only represents the direct propagation between a pair of components, the cumulative change propagation probability which is the sum of the 1-step and multi-step propagation is derived as follow.

$$\begin{array}{c|cccccccccc} C_1 & 1 & 0.084 & 0.022 & 0.312 & 0.0271 & 0.0437 & 0.0203 & 0.0162 & 0.0123 & 0.0143 \\ C_2 & 0 & 1 & 0 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0.068 \\ C_3 & 0 & 0 & 1 & 0 & 0.47 & 0 & 0 & 0 & 0 & 0.1598 \\ C_4 & 0 & 0 & 0 & 1 & 0 & 0.14 & 0.065 & 0.5205 & 0.039 & 0.0164 \\ C_5 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0.34 \\ C_6 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0.265 & 0.225 & 0.0903 \\ C_7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.23 & 0.115 & 0.0575 \\ C_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0.12 \\ C_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.26 \\ C_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Using the formula (9), (10) and (11), the change propagation probability based on architectural pattern $CMP(cmp_{ij})$ is:

$$\begin{array}{c|cccccccc} C_1 & 1 & 0.106 & 0.312 & 0 & 0 & 0 & 0 & 0 \\ C_2,C_3 & 0 & 1 & 0 & 0.576 & 0 & 0 & 0 & 0 \\ C_4 & 0 & 0 & 1 & 0 & 0.205 & 0 & 0 & 0 \\ C_5 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0.34 \\ C_6,C_7 & 0 & 0 & 0 & 0 & 1 & 0.63 & 0.53 & 0 \\ C_8 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0.12 \\ C_9 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.26 \\ C_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Similarly, the cumulative change propagation probability of architectural patterns is:

$$\begin{array}{c|cccccccc} C_1 & 1 & 0.106 & 0.312 & 0.0611 & 0.064 & 0.0403 & 0.0339 & 0.0344 \\ C_2,C_3 & 0 & 1 & 0 & 0.576 & 0 & 0 & 0 & 0.1958 \\ C_4 & 0 & 0 & 1 & 0 & 0.205 & 0.1292 & 0.1087 & 0.0437 \\ C_5 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0.34 \\ C_6,C_7 & 0 & 0 & 0 & 0 & 1 & 0.63 & 0.53 & 0.2134 \\ C_8 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0.12 \\ C_9 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.26 \\ C_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Define change propagation density to reflect the potential of the architecture to expose its components to the changes.

$$CPD=\frac{\sum_i\sum_{j\ne i} cp_{ij}}{n(n-1)} \tag{12}$$

Where, n is the number of the components in the architecture. The idealistic change propagation density corresponds to an identity matrix I, which indicates that no component propagates changes to others. The worst extreme is that any change in any component propagates to all other components. With the formula (12), the CPD of the original architecture based on component is 0.1496, and the CPD of the architecture based on pattern is 0.1333. The difference between these two values implies that choosing architectural pattern reasonably is an effective measure to lower the risk of change propagation.

The approach proposed here introduces change propagation probability as design metric to evaluate the different architectural patterns. It is useful in following two directions: (1) incorporating the change propagation into the software architecture evaluation; (2) decreasing the risk of software development by choosing proper architectural patterns.

6. Conclusions

This paper aims to predict change propagation in early stages of software development, and to evaluate the software architecture based on pattern. We discuss the process of change propagation in software development, and highlight the necessity of predicting change propagation probability. The formal definition of change propagation probability between a pair of components is introduced, and a probabilistic method based on UML diagrams to estimate the probability of change propagation is deduced. Due to the characteristics of different architectural patterns, this paper extends the pattern-based propagation, which incorporates design metrics into software architecture evaluation. The method and the results of the computational experiment will promote more effective architecture design process and will contribute to improve the software development process, especially for early stages of software development.

This study is part of a wider work that considers the impact of software requirements change. The future works are currently in progress, including analyzing requirements change propagation in complex architectures that contain iterations, determining the impact of change propagation on cost and schedule, and simulating the risk of requirements change.

Acknowledgements

This work was supported by the National Science Foundation of China (No. 70771074), the Self-Innovation Foundation of Tianjin University, and the Ph.D. Programs Foundation of Ministry of Education of China (No. 20090032110065).

References

[1] Garlan, D., Software architecture: a roadmap, *Proceedings of the Conference on the Future of Software Engineering*, pp.91-101, 2000.

[2] Hassan, A. E. and Holt, R. C., Predicting Change Propagation in Software Systems, *Proceedings of the 20th IEEE International Conference on Software Maintenance*, pp. 284-293, 2004.

[3] Strens, M. R. and Sugden, R. C., Change analysis: a step towards meeting the challenge of changing requirements, *Proceedings of IEEE Symposium and Workshop on Engineering of Computer-Based Systems*, pp.278-283, 1996.

[4] Ebert, C. and Man, J. D., Requirements uncertainty: influencing factors and concrete improvements, *Proceedings of the 27th international conference on Software engineering*, pp. 553-560, 2005.

[5] Giffin, M., Weck,O. d., et al, Change Propagation Analysis in Complex Technical Systems, *Journal of Mechanical Design* , vol.131, no.8, pp.1-14, 2009.

[6] Abdelmoez, W., Shereshevsky, M. et al, Quantifying software architectures: an analysis of change propagation probabilities, T*he 3rd ACS/IEEE International Conference on Computer Systems and Applications*, pp. 124-131, 2005.

[7] Shereshevsky, M., Ammari, H., et al. Information Theoretic Metrics for Software Architectures, *Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development*, pp. 151-157, 2001.

[8] Sharafat, A. R. and Tahvildari, L., Changes Prediction in Object-Oriented Software Systems: A Probabilistic Approach, *Journal of Software*, vol. 3, no. 5, pp. 26-39, 2008.

[9] Abdelmoez, W., Nassar, D. M., et al. Error Propagation in Software Architectures, *Proceedings of the Software Metrics, 10th International Symposium*, pp. 384-393, 2004.

[10] Bass, L., Clements, P. and Kazman, R., Software Architecture in Practice, Addison Wesley, 1998.

[11] Wang, W. L., Pan, D., et al. Architecture-based software reliability modeling, *Journal of Systems and Software*, vol.79, no.1, pp. 132-146, 2006.