http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (21): 6288-6294, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.6288.6294

Software Reliability Evaluation Approaches using Weighted Software Network

Wang Bei-Yang
School of Computer Science, Yangtze University, Jingzhou, Hubei, 434023, China

Abstract: Software reliability evaluation and growth models are mostly based on software testing. These methods need to be carried out after coding. It will be great benefit to the cost of software development, if evaluate the software reliability and improve it earlier. This study proposed a software reliability evaluation method. The method is based on the software network, evaluate the reliability at the architecture level of software. Using this method can evaluate the software reliability before coding. It can provide reference for the optimization of software structure design and improve the software reliability in the early stage of software development.

Key words: Software reliability evaluation, software architecture, software network, weighted software network, software engineering, software quality control

INTRODUCTION

Software reliability is an important factor that affects the whole system reliability and is generally accepted as the key factor in software quality. The level of the software reliability is closely related to the probability of failure-free operation of a software system. "Software reliability is often defined as the probability of failure-free operation of a computer program for a specified time under specified conditions" (Eusgeld *et al*., 2008). The propagation of the software error directly affects the probability of failure of the software system operation and the errors propagation is closely related to the architecture of software. Therefore, to study the relation between software architecture and software reliability is very important.

E. Adams' study (Adams, 1984) revealed that a great proportion of latent software faults lead to very rare failures in practice while the vast majority of observed failures are caused by a tiny proportion of the latent faults. Similarly, Norman E. Fenton *et al*. in their study (Fenton and Ohlsson, 2000) found strong evidence that a small number of modules contain most of the faults discovered in prerelease testing and that a very small number of modules contain most of the faults discovered in operation. They confirmed that the number of faults discovered in prerelease testing is an order of magnitude greater than the number discovered in 12 months of operational use.

"The software architecture provides the basis for implementation of fault tolerance." (Wilfredo, 2000) Therefore, the ability of fault tolerance is different with different software architecture.

A large-scale software system is a complex system and is a scale-free network with power law distribution (Liu *et al*., 2008). Therefore, using the software network represents the software structure will help us to study the characteristics of the software structure.

Architecture-Based software reliability prediction model are mainly three types: State-based approach, path-based approach and additive approach (Goseva-Popstojanova and Trivedi, 2003). Software reliability prediction in testing stage is the mainstream. These methods statistic software errors when it running (or testing) and predict the reliability of the software. Our method predicts the software reliability in the design stage. This study introduces a weighted software network model which named WSNNI, it can reflect the software structure and the impact between the nodes of the software network more accurately. On this basis, this study proposes an architecture-based reliability evaluation method Based on WSNNI. This method do not able to replace the traditional software reliability prediction models, but it can help software developers to understand the software structure which their designed and to determine the impact of the software structure on software reliability.

RELATED WORK

Software reliability model specifies the form of a random process that describes the behavior of software failures with respect to time (Lyu, 2007). There are three main reliability modeling approaches: (1) The error seeding and tagging approach, (2) The data domain approach and (3) The time domain approach. The third is considered to be the most popular one.

Since Jelinsky and Moranda proposed the first SRGM (Jelinsky and Moranda, 1972), numerous SRGMs have been proposed in the past 40 years, such as exponential failure time class models, Weibull and Gamma failure time class models, infinite failure category models, Bayesian models and so on (Lyu, 1996). In 2010, Brosch *et al* model cost relevant business processes as well as the associated IT layer and then connect them to failure probabilities. Based on this they conduct a reliability and cost estimation (Brosch *et al*., 2010).

Architecture-based reliability prediction model is not much, in 2003, Gova *et al* proposed a architecture-based software reliability prediction model (Goseva-Popstojanova and Trivedi, 2003). In the same year, Gokhale and Swapna *et al* also proposed one (Gokhale, 2003). In 2004, Gokhale and Swapna *et al* proposed a dynamic architecture-based software reliability prediction model again (Gokhale and Trivedi, 2006; Gokhale *et al*., 2004).

In this study we propose an architecture-based reliability prediction model too. In our model, we analyze the structure of software system using software network. Software system is complex network (Liu *et al*., 2008; Moyano *et al*., 2011; Wang and Wang, 2012) software network reflect the characteristic of software structure better, so it is appropriate to use software network in architecture-based reliability prediction model. In our previous study, we proposed a weighted software network model which named WSNNI (Wang and Wang, 2012). This model is able to reflect the software structure more accurately. In this study, we use the WSNNI to make our reliability prediction model. We will introduce software network and the WSNNI model at following.

SOFTWARE NETWORK AND WSNNI MODEL

Software network: When use network to describe the software structure, we can extract the nodes from the software in the different granularities, including packages, classes, methods and features; and extract the edges from the dependencies between these units. The software network is structured by these nodes and edges. The definition of software network as following:

Definition 1: Software network is generally described by two-tuples: $\Omega S = (W_s, E_s)$, where $W_s = \{s_i\}$, $(i = 1, 2, \ldots, N)$, is a sets of N classes, E_s is the sets of dependencies between software modules (classes, components, etc).

Weighted Software Network for Node Impact (WSNNI): WSNNI is a weighted software network model which proposed in our previous study (Wang and Wang, 2012). Using WSNNI can well reflect the defects propagation of software. In this study, we use WSNNI to analyze the relation between software reliability and the defects propagation. Now we will introduce WSNNI.

Definition of WSNNI: WSNNI is used to represent the impact between nodes of the software network, it can be defined as following:

Definition 2: Weighted Software Network for Node Impact (WSNNI) described by two-tupels:

$$WSNNI = (G, M_I) \quad (1)$$

In the expression, G = (N, W), the N is a set of nodes, W is a set of weighted edges. In WSNNI, we use directed and weighted network, so the w_{ij} and w_{ji} is not the same. M_I is a matrix that stores the impact measurement value between the nodes in graph G.

Definition of the Weight in WSNNI

Definition 3: The definition of the weight of the edge between tow nodes.

Let Nfi denotes the sum number of methods of the node i, Nfij denotes the sum number of the methods of the node i dependent on the method of the node j; N_{fj} is the sum number of methods of the node j, $NF_{j\leftarrow i}$ is the sum number of methods of the node j are depended by the node i. The calculation of the weight w_{ij} as follows:

$$w_{ij} = \frac{Nf_{ij}}{Nf_i} \times \frac{Nf_{j\leftarrow i}}{Nf_j} \quad (2)$$

Node Impact Matrix in WSNNI: The definition of the weight w_{ij} represents not only the impact of adjacent nodes, but also these nodes which are not adjacent.

Let I_{ij} represents the impact measurement of the j to I, then I_{ij} is the product of the weights of the path i to j.

After calculated the impact measurement value of each node, the following defines the Nodes Impact Measurement Value Matrix (M_I) of the WSNNI.

M_I stores the impact measurement values I_{ij} between the nodes of the directed graph G. The rows of the matrix arranged with a nodes sequence of the node set of the G, the columns arranged as the same sequence of the rows. The value at the junction of the row and column is the impact measurement value.

In the MI, the I_{ij} denotes the value that node i impacted by the j, $0 \le I_{ij} \le 1$. $I_{ij} = 0$ represents node i do not depend on the node j; $I_{ij} = 1$ represents node i depend on j completely. The dependence of the node itself is completely, so in the matrix, each value on the main-diagonal is 1.

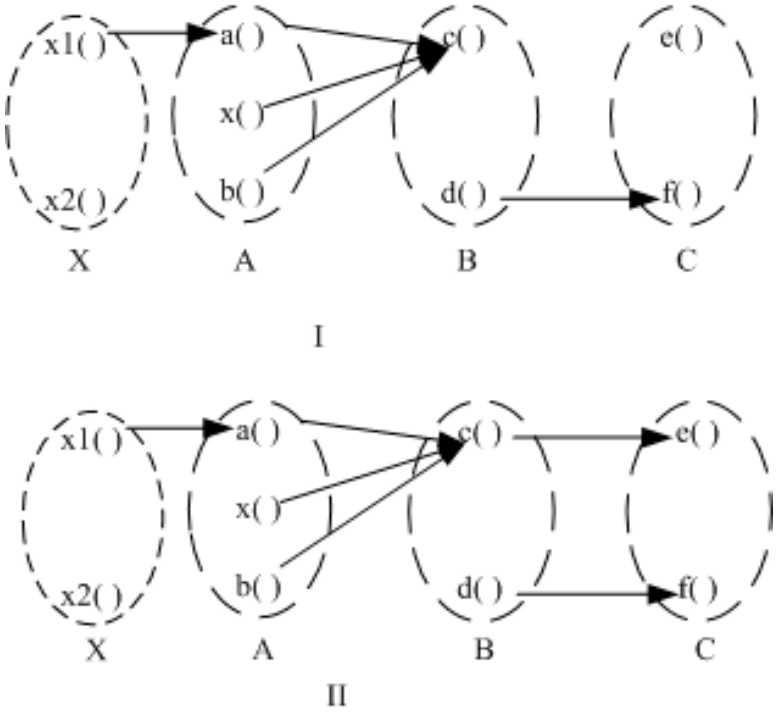

Fig. 1: Illustration for the relation between software structure and reliability

More of the introduction about software network and WSNNI can be seen in our study (Wang and Wang, 2012).

RELIABILITY ANALYSIS BASED ON WSNNI MODEL

In WSNNI model, based on the weight of each edge, we give the node impact values between two nodes, it is the measurement of impact intensity that between two nodes. Here we give an example to illustrate the impact values how to influence the reliability, at the same time to illustrate the software structure how to influence the reliability, shown in Fig. 1.

Suppose that there are 3 defects in each node in Fig. 1, then in the graph I, the path from the node X to C, there will propagate 3 defects from the other nodes to C. However, in graph II, there will be 18 (3+(3+3+3)+(3+3) = 18) defects been propagated to C at the same path. Despite we just added a dependency from B to C. We can see from this example that in Fig. 2, we just added a dependency, but the defects in the C increased by 500%. Therefore, we can see that how much the structure of software impact on the reliability and using the WSNNI model to analyze the reliability is appropriate.

RELIABILITY PREDICTION METHOD BASED ON WSNNI

Architecture-based analysis techniques can be classified into two categories (Gokhale, 2007), namely, path-based and state-based. Our method is path-based technique. Path-based technique includes measuring software components defects and the defects of propagation in the software. The proposed method in this study is based on WSNNI. In traditional path-based technique, loop problem is an obstacle. Further, it is not accuracy to get the probability of defects propagation in traditional path-based technique.

Figure 2 shows a weighted software network which is build based on WSNNI, contains 10 nodes and the value on each edge represents the weight of the edge. We will introduce our method via Fig. 2.

Before introducing our reliability prediction methodology, we need to give two assumptions for our methodology.

Assumption 1: The numbers in all software components are determined.

Because our method is focus on the software structure impact on reliability, predicting the reliability in the early stages of software development, it is acceptable to ignore the number of defects in components.

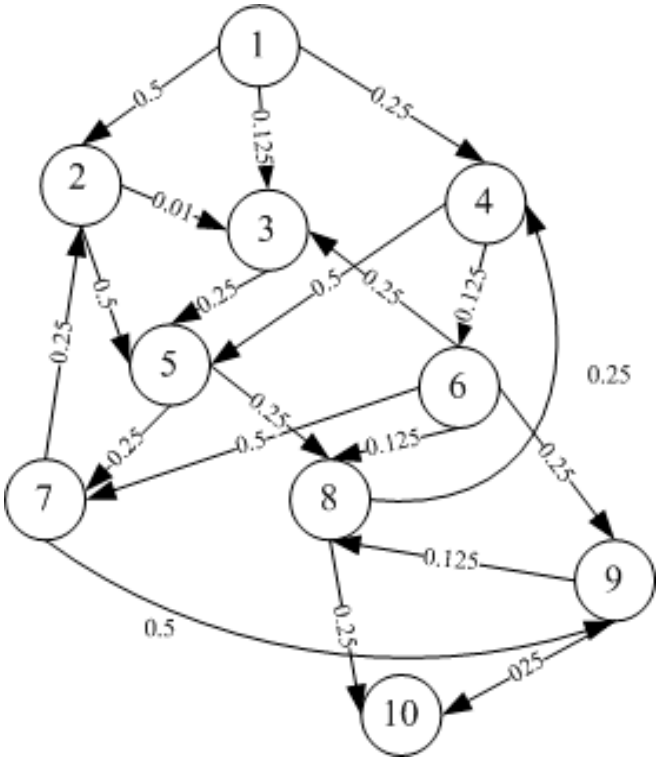

Fig. 2: Example graph for WSNNI

Assumption 2: The defects in all software components are not interrelated.

This assumption is used by all of path-based methods. In fact, this assumption leads to all of path-based methods are approximated.

Individual component reliability and impact on its adjacent components: The individual components represented as nodes in the software network. A node impact the reliability of its adjacent nodes depends on probability that its defects propagation to these adjacent nodes. From the analysis in our study (Wang and Wang, 2012), we can see that as a node in software network, the higher s_i the higher impact on its adjacent nodes. From the definition of s_i we can seen that the weight is the primary factor that a node impact on its adjacent nodes. The expression 3 used to predict a node impact its adjacent reliability.

Lets node i contains number of defects is f_i, then the number of defects from i propagation to a adjacent node j is f_{ij}:

$$f_{ij} = f_i \times w_{ij} \quad (3)$$

The w_{ij} is the weight of node j dependents on node i. The weight reflects the probability of node i propagate its defects to node j. From the definition of weight we can see that even if the node i has many defects, if the weight is lower the defects in node i propagate to j will be not much.

Reliability prediction of a single path: According to literature (Wang and Wang, 2012), on the edge which node i to node j (j connected to i, but not adjacent), the number of defects from i propagate to j can be predicted by the expression 4.

Assume that the number of defects in node i is f_i, then the number of defects f_{ij} form i propagation to j predicts as the expression 4:

$$f_{ij} = f_i \times I_{ij} \quad (4)$$

Here the I_{ij} is the impact measurement value which defined in WSNNI. In expression 4, the directed edge (i, j) contains the other nodes and these nodes will propagate their defects to node j. However, we do not count these defects into the f_{ij}. Because of these defects will counted in the other paths. For example, the defects in node n propagate to j will be counted in (n, j). When we assume that the number of defects in a component is a constant, affect the reliability of a path mainly is the impact measurement value I_{ij}. Software development practice has proven that when the coupling between components is weaker the smaller the impact between the components and vice versa.

As shown in Fig. 2, let node 1 contains n defects, calculate the number of defects of the path which from node 1 to node 10 (1, 3, 5, 8, 10) as following:

$$f_{1,2,3,8,10} = f_i \times I_{1,3,5,8,10} = n \times 0.125 \times 0.25 \times 0.25 \times 0.25 = 10 \times 0.01953125 = 0.01953125n$$

Our method uses simulation for execution the edges of the software network, the simulation algorithm is given below. Let the number of simulation times is N, where the path P_{ij} is executed n times in N times simulation, then get the total number of defects of P_{ij} in N times simulation is $n \times f_{ij}$. We can calculate the reliability R_{ij} of path P_{ij} as follows:

$$R_{ij} = \frac{n}{N} \times \frac{f_{ij}}{\sum_{\forall(i',j')\in E} f_{i'j'}} \quad (5)$$

Where:

$$\sum_{\forall(i',j')\in E} f_{i'j'}$$

is the sum of defects in the all paths.

Entire software system reliability analysis and prediction: Path-based reliability prediction techniques commonly use average reliability of all paths to denote the system reliability (Goseva-Popstojanova and Trivedi, 2003). In this study, we use the same way to calculate the entire software reliability.

In order to fully reflect the reliability of the software, the number must be sufficiently large in the simulation. The reliability of the entire system is denoted by the average of results of these simulations. Let the times of the simulations is N, then the entire software system reliability R_s is calculated as follows:

$$R_s = \frac{\sum_{1}^{N} \sum_{\forall(i,j)\in E} f_{ij}}{N} \quad (6)$$

In Eq. 6:

$$\sum_{\forall(i,j)\in E} f_{ij}$$

is the system reliability obtained from a complete system simulation. Take the average of N times simulations results, thus avoiding the difference for one times simulation.

Our method is using simulation to predict the reliability, the algorithm is given below.

Simulation algorithm: In the simulation algorithm, we just calculate the number of defects on paths, accumulate all the number of defects of the all components can predict the system reliability.

The simulation algorithm is as follows:

Input:	WSNNI of the software system
Output:	The software reliability R_s
Step 1:	Input the WSNNI of the system and the number of simulation N, the current number of simulation n is 0
Step 2:	Set the visited status of all nodes are 0 (do not be visited)
Step3:	Select a node by probability p (v_i) as the initial visited node V_{init}, set the visited status of V_{init} is 1 (that has been visited) and then add node V_{init} to the visited queue VistedNodeQueue;
Step 4:	Select the first node v_r in the VistedNodeQueue, get all the precursor node of v_r (that all nodes depend on v_r) whose visited status are 0 and add them into set S
Step 5:	Set the visited status of the v_t in the S is 1 by weight of $<v_r, v_t>$ as the probability, if the visited status of v_t changed from 0 to 1, then add it to the end of the VistedNodeQueue and set $v_t = v_r$; calculate the number of defects of this path ($f_{r->t}$) according to expression 4 and set $f = f+f_{r->t}$
Step 6:	Delete the first node v_r of the Visted NodeQueue, set the set S is empty
Step 7:	Repeat steps 4 to 6, until the VistedNodeQueue is empty
Step 8:	Let $f_n = f_n+f$, calculate the average of n times simulations: $f_a = f_n/n$; set the current number of simulation is n=n+1; if n<N, repeat the steps 2 to 7, else the simulation end

EXPERIMENT

In the architecture design stage, we can build the WSNNI according to collaboration diagram. Also, we can directly extract the software network from the source code and then build the WSNNI. So, our method is able to predict the software reliability in early stage.

We selected two open source software as the object of our experiment.

Data sources: This study selects the open-source software Junit and Jedit as the experimental objects, using tool software to extract respectively the dependencies in the granularity of packages, classes and methods, to build the WSNNI and the node impact matrixes M_I. The statistical information of the Unit and Jedit about the package, class and method are shown in Table 1. In the statistics information, we exclude the codes related to Java for more accurately reflect the features of the software itself.

Experimental settings: In the simulation, the probability of choosing each node as the start node is related to the number of the elements of the set, set the number is M, the probability is calculated as follows:

$$p(v_i) = \frac{1}{|M|}$$

where, $v_i \in M$.

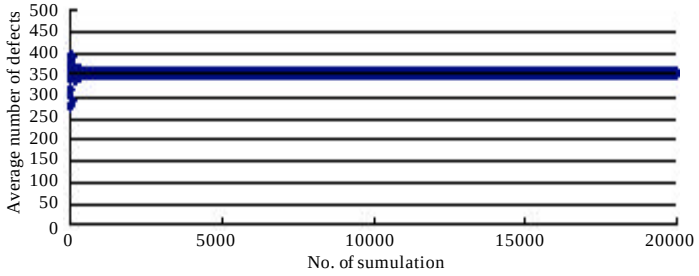

Fig. 3: Results of simulation for Jedit 4.3

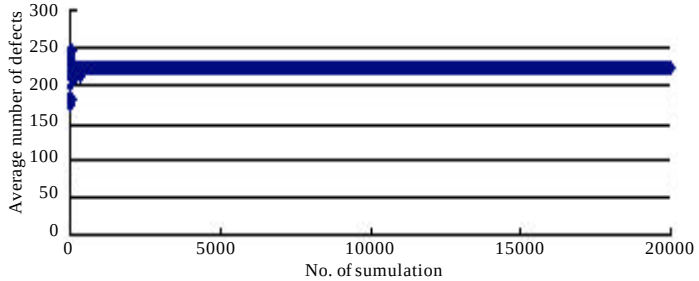

Fig. 4: Results of simulation for Junit 4.9

Table 1: Statistics information of Junit and Jedit

Unit	Packages		Classes				Methods			
Statistics items	Junit	Jedit	Junit		Jedit		Junit		Jedit	
Number of the units (nodes)	30	29	248		1132		1562		12105	
Dependencies of the units (edges)	0	0	From class	To class	From class	To class	From method	To method	From method	To method
			393	3809	1655	27437	6957	3541	6957	43381

In order to reduce the inaccuracy of the simulation, the simulation times N is set to 20000. At the same time, using this simulation algorithm can avoids loop path.

Results and analysis: We experiment on the two open-source software Jedit and Junit and set the simulation times N = 20000.

In simulation, let the current simulation times is n, when the simulation is end, according to the algorithm to calculate the average of reliability get from the n times simulation. Thus, the experimental results is the average of n times simulation.

The experimental results show in Fig. 3 and 4.

We can seen from Fig. 3 and 4 that early in the simulation, the reliability of software system is somewhat different which is related to initial node of the path. With the increase of the simulation times, the simulation results also tend to stabilize. This proves that our simulation algorithm is validity. Our analysis shows that, when the number of simulation times is much larger than the number of nodes of the software network, the average of simulation results is tend to a same value, this value is the software reliability evaluation value.

There are relatively large difference in size of Jedit and Junit. We can seen from the table 1, that the nodes number of Jedit is 5 times of the Junits'. Hover, the structural reliability of Jedit only 1.5 times than Junit. From the comparison we can deduce the following conclusions: (1) Software size is larger, the lower the reliability of the software; (2) The trend of software reliability decreasing is ease than the size of the software increasing.

CONCLUSION AND FUTURE RESEARCH

In order to better study the characteristics of the software structure, we introduced the software network method; and then introduced a weighted software network model WSNNI which proposed in our previous research; based on the WSNNI, we analyzed the impact of software reliability. We analyzed these characteristics impact on software defects propagation and then proposed the reliability evaluation model. Our method is focus on software structure and predict the software reliability in software design phase, so we using simulation to predict the software reliability. And then we give the simulation algorithm. And then, we use an

example to validate our method and the simulation algorithm. In the software practice, software structure has great influence on many development practice activities, such as integration testing, software portability, stability and so on. We will focus on these problems in our future work.

REFERENCES

Adams, E.N., 1984. Optimizing preventive service of software products. IBM J. Res. Dev., 28: 2-14.

Brosch, F., R. Gitzel, H. Koziolek and S. Krug, 2010. Combining architecture-based software reliability predictions with financial impact calculations. Electron. Notes Theor. Comput. Sci., 264: 3-17.

Eusgeld, I., F. Fraikin, M. Rohr, F. Salfner and U. Wappler, 2008. Software Reliability. In: Dependability Metrics: Advanced Lectures, Eusgeld, I., F.C. Freiling and R. Reussner (Eds.). Vol. 4909, Springer, New York, ISBN-13: 978-3540689461, pp: 104-125.

Fenton, N.E. and N. Ohlsson, 2000. Quantitative analysis of faults and failures in a complex software system. IEEE Trans. Software Eng., 26: 797-814.

Gokhale, S.S. and K.S. Trivedi, 2006. Analytical models for architecture-based software reliability prediction: A unification framework. IEEE Trans. Reliability, 55: 578-590.

Gokhale, S.S., 2003. Accurate reliability prediction based on software structure. Proceedings of the 7th IASTED International Conference on Software Engineering and Applications, November 3-5, 2003, Marina del Rey, CA., USA., pp: 122-127.

Gokhale, S.S., 2007. Architecture-based software reliability analysis: Overview and limitations. IEEE Trans. Dependable Secure Comput., 4: 32-40.

Gokhale, S.S., E. Wong, J. Horgan and K.S. Trivedi, 2004. An analytical approach to architecture-based software reliability prediction. Perform. Evaluation, 58: 391-412.

Goseva-Popstojanova, K. and K.S. Trivedi, 2003. Architecture-based approaches to software reliability prediction. Comput. Math. Appl., 46: 1023-1036.

Jelinsky, Z. and P. Moranda, 1972. Software Reliability Research. In: Statistica Computer Performance EvaZuation, Freiberger, W. (Ed.). Academic Press, New York, ISBN: 981-02-0640-2, pp: 465-484.

Liu, J., J. Lu, K. He, B. Li and C.K. Tse, 2008. Characterizing the structural quality of general complex software networks. Int. J. Bifurcation Chaos, 18: 605-613.

Lyu, M.R., 1996. The Handbook of Software Reliability Engineering. Mc Graw Hill Publications, New York.

Lyu, M.R., 2007. Software reliability engineering: A roadmap. Proceedings of the Future of Software Engineering, May 23-25, 2007, Minneapolis, Minnesota, pp: 153-170.

Moyano, L.G., M.L. Mouronte and M.L. Vargas, 2011. Communities and dynamical processes in a complex software network. Phys. A: Stat. Mech. Appl., 390: 741-748.

Wang, B. and L. Wang, 2012. Analysis of defects propagation in software system based on weighted software networks. J. Converg. Inform. Technol., 7: 63-77.

Wilfredo, T.P., 2000. Software fault tolerance: A tutorial. Technical Report: NASA-2000-tm210616.