5th SASTech
Iran |Mashhad
May 12 - 17 | 2011
5th Symposium on Advances in Science & Technology

Component Based Software Engineering: Reliability

Photograph of Presenter

S.Bashavard1, N.Khalili Safa
1Shahid Beheshti university, Tehran, Iran,
S.bashavard@mail.sbu.ac.ir;
2Tabriz university, Tabriz, Iran,
nazli.khalili@tabrizu.ac.ir;

Paper Reference Number: 0106-748
Name of the Presenter: N.khalili safa

Abstract

Component Based Software Engineering (CBSE) spans the most advantages in reusability and distributed programming. Despite of these advantages, software composers and component end users are still very much concerned about the security impact of foreign components on their systems. In such component based software development, functionality and quality of service of software components must be sufficiently clear and determined through its interfaces so that the system features assembled can be analyzed according to its requirements. So, it requires that the component reliability ensured in this systems in order to guarantee the security. In this paper, we invest the reliability properties in component based software engineering. Also, we demonstrate the impact of reliability on software as important as hardware.

Key words: Component based development, reliability, security

1. Introduction

Design process in most engineering systems, based on component reuse. Mechanical or electronic engineers plan according to components that have been created and tested in other systems. The idea of using functions, modules and applications programs has spread for years in Software Engineering. Advantages of idea include: reducing of development cost, increasing the reliability, reducing the risk process, effective use of experts, standardization, rapid development and increasing application software. In contrast, the reusability spans problems such as: maintenance costs, lack of tool support, maintenance of component libraries, and finding the matching component. Hence, these problems effectively prevent component reusing [7]. Basically, Software Reuse must be considered in requirements engineering or in the design stage.

A major technical challenge posed by CBD is the security of third-party software components, typically commercial-off-the-shelf (COTS). The lack of security in component-based systems (CBS) may result in breaches of its own integrity, as well as of confidentiality and integrity of the underlying information assets [4]. Therefore the role of reliability for the product that is to be used in the intended system development would be of most importance.

In the second section we would offer some explanations on the history of component-oriented and its infrastructure. In section 3 we would explain the role of reliability on component-based system development and in section 4 we would discuss and review previous literature in this field.

2. Component oriented

One of the main goals of object-oriented technologies in early 1990s was to offer a technical framework to model the process based on reusing the object to develop the software because object-oriented pattern focuses on building the classes that hide both data and algorithms used for manipulation of these data. It was assumed that if the object oriented classes be designed correctly they could be used in different architectures and consumptions. But the object classes are so special and detailed and bound to a particular program while compiling and connecting. Detailed knowledge of these classes i.e. the program code was necessary in using them. This issue was accompanied with the problems and costs like maintaining the object library, finding and conforming the objects. Following the object-oriented technology problems in reusing ready objects in late 1990s, Component Based Software Engineering CBSE) emerged [2].

In recent decade, component-based designing has become an important technique to build and develop complicated systems. This method deals with the component designing as reusable units, designing the systems by reusable components, maintain and updating the system designs by replacing or building new components. Though components based designing has been used widely, but there is no standard definition for component. There are some suggestions in the literatures each of which point to different aspects of the component [3].In this article we consider below definition proposed by Sezi Perski:

- A software component is a unit to participate in combination that has described intermediates and clear and definite environmental dependants. A software component can be built and developed independently and used in the combination by a third party.

An important feature of a component is its interface separation from implementation. This separation differs from the classes definition separation from their implementation in the object-oriented programming. Here it is needed that a component compiling in software be independent from the component development life cycle and there would be no need for compiling or connecting the software contained a new component again. Indeed the component is an independent execution unit; it does not provide resource code for the software and won't be compiled with the other system components. All the interoperations are done by the interfaces that a component interface is the only communication way of that component with environment and the other components. In other words, the component interface indicates input, output and visible behavior of that component [3]. In most commercial and research works, two kinds of interfaces have been defined for the component:

1) Supplying interface that defines the services provided by the component
2) Demand interface identifies that which services should be offered by the user system to provide its service using component well.

Components may be in several levels of abstraction. Mayo [3] has proposed the following 5 abstract levels:

1) Functional Abstraction: implements one functional component like arithmetic function. In fact, the supplier interface is itself a function.
2) Casual Grouping: component is a set of units that connects with each other poorly. It may be data declarations, functions and etc. statement. The supplier interface is composed of the names of all the unit existed in the group.
3) Data Abstraction: the component shows data abstraction or the class of object –oriented language .The supplier interface is composed of the operation to establish reform and accomplish to data abstraction.
4) Cluster Abstraction: a group of classes related to each other constitute a component. The supplier interface is a combination of all the object interfaces.
5) System Abstraction: component is a complete self-contained system. The supplier interfaces an API that allows the programs to reach to the order and operations of the system.

Regarding to several abstraction levels and the other significant advantages of independent development of the component, the component based design yields other new problems:

1) How will we be assured that these independent developed components work together?
2) How can we know that component based system will perform whatever we order?

Solving these problems is essential for being reliable of the functionality and quality of the final systems. Basically this task is done in 4 levels:

1) Data type compatibility: means that the related components agree on the exchanged data types. For example, if a component accepts correct data, the other component will send correct data, too.
2) Behavior type compatibility: means interoperable components described in several models of calculation are compatible in common pattern of communication[3].For example , in natural world a calculation model called "physic rules "manages interoperations among the components. The components written in several languages may have several models of calculation. Compatibility in this level is like compatibility among the component interoperation rules. In comparison to the following behavior compatibility, this compatibility level is determined as statically.
3) Behavior compatibility: means that there is no unexpected interoperation among the components. This assures that the environmental dependency of each component has been completed. Particularly, because component and its used environmental develops independently; this contact assures sound interaction among them. The type of interoperation shows communication protocol that a component knows about its environment. To perform a component its own function correctly, its used environment should follow the component protocol.
4) Proving Desired Properties: means the system does whatever we want. This issue needs to prove the accuracy of the system security and liveness features.

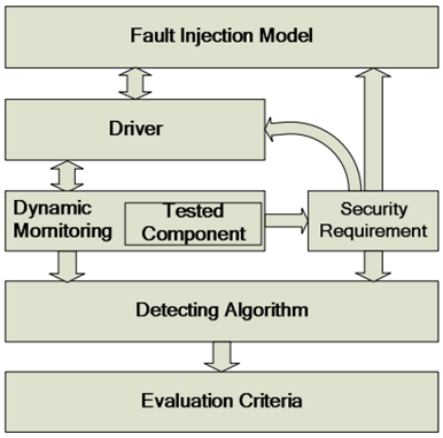
In the existed industrial standards like CORBA and DCOM and JavaBeans that are middlewares to manage communication among distributed objects, the component interfaces consider just semantic information and leave aside the important syntax information. For example, description of dynamic features and syntax constrations of the components presented in their interfaces are not supported directly. So, these standards could not answer the questions related the above last three levels. This issue in designing component based systems leads to several problems in establishing coordination among the component. That means inaccurate assumptions about the component services do mostly lead to incorrect usage of them and the software failure. Zung [3] has examined the results of first level. Many researchers attempt to analyze higher level of component based systems through enriching component interface with syntax information based on the industrial standards [3].

3. The Role of trust and Reliability

In terms of origin, components can be software acquired from vendors and appliers, open source software and, increasingly, freely downloadable software. In terms of security, this raises two important issues: trustworthiness of the supplier and trustworthiness of the product (component) the intention of the original supplier, or the source, of the software. Issues of trust have received considerable attention; see. On the one hand, trustworthiness of the product is related to: a) component reliability and integrity, that is, whether a component is well-engineered, well-documented, tested and validated, and b) absence of any malicious code in the component that is deliberately designed to damage the system., there are many software vulnerabilities that could be exploited by malicious attackers. On the other hand, there are also inadvertent vulnerabilities, such as component mismatch and errors encountered in the adaptation of a third–party software. Failures in both categories of trustworthiness could jeopardise the goals of security, namely, confidentiality, privacy, integrity of data, and availability of services [4].

4. Review of Related Works

Today CBSE has become one of the main aspects of software engineering. But the problems like reliability and security of the components are still unsolved and these problems lead to worriness in software developers and users. The software component testing is an important task that can cause security and reliability guarantee and improvement [1]. Many researches have been done in this field. Berto lino [1] has offered a framework to test the component development in which a spy class added to testing components to compare and collect the resources allocation and the component state, while performing .M.Hadox[1] has offered a general testing viewpoint about third party components. A view is that adding interfaces to t components testing in order to test both input and output that allows data to flow inside and outside of the component in a general interface level and as a result third party components could be recognized and tested better [1]. Jinfo Chen [1] et.al have suggested a method to test the components that are based on the fault injection. In this method build on environment to test alone component and enters the component to this environment .It injects types of expectable faults to this component. It compares functional graph of this component with the one expected by the component function and identifies the component shortcoming. Figure1 shows the fault injection environment.



Fiure1.The method of testing fault injection based component security

Two years later in 2009, Bin Bin G et.al [5] suggested a developed model of fault injection method. In this method in contrary to the previous method in which the fault injected just on one component unit, the fault injection operation was done on both the component and the environment. Of environment we mean the interoperation of two components related to each other and shows the injected fault types and the component function .This method leads to test the integrity and compatibility of component based systems. The function of this method has been shown in figure 2.

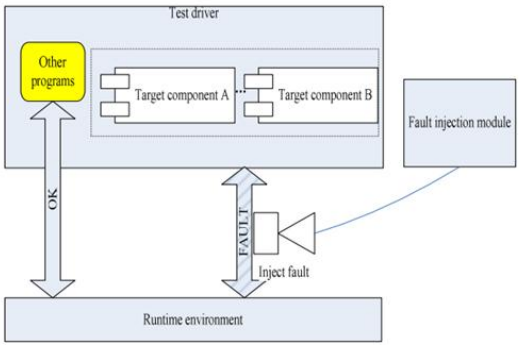


Figure2. The environment of developed fault injection architecture to test the component security.

Stafford and et.al [8] tried to predict the selecting which component can give more reliability to the system by using graph and determining the component via this graph. They depended reliability of the system and the software to time. However, most authors don't relate this qualitative demand to time. The reason is that they relate this demand to hardware not software. While the software destruction and expiration date of the software is a reason to lowering the software system reliability. Roshnak Roshndel and et.al [6] by reviewing the features and challenges of reliability in the software system and then component-based systems concluded that it is better to guarantee the reliability before producing the software and in architecture stage. Their discussion criteria was on this issue that reliability could be supplied using hardware. The following figure shows the function method of this group.

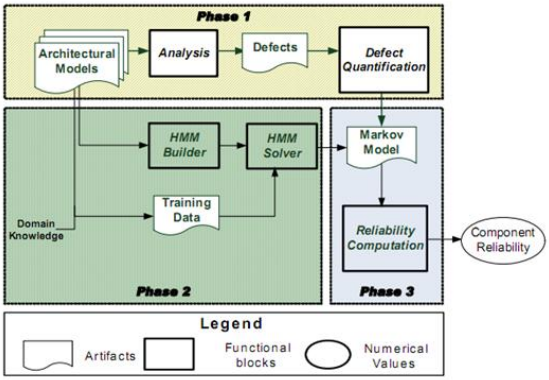


Figure3.Estimation framework of the component reliability

5. Conclusion

The aim of software reliability methods is to decrease or eliminate the faults existed in the software systems .reliability in software system are mainly evaluated during the system function or after it. Traditional software engineering suggests that the reliability estimation while the software performance time may be too late. If problems in this field would be identified, the system may have to be redesigned and work that would be very costly. Therefore it is better to evaluate the software reliability throughout the system life. In component based systems in which the system reliability is under question because of the components black box, regarding performed researches, it is better to test this feature not only before selection of the component but also the time o f combination and even after that and while starting the software function so that the system would be a secure system with complete security and assured ground to be able to use the system and also alternative develops with more reliability.

References

[1] Judith Stafford, John D. McGregor. Issues in Predicting the Reliability of Composed Components, 5th ICSE Workshop on Component-Based Software Engineering Orlando Florida, 2002
[2] G.T. Heineman ,etc. Component based Software Engineering, 8th International Symposium, CBSE 2005
[3] Jaber Karimpour. A New Formalism for Mathematical Description and Verification of Component-Based Systems, PHD theses,2009
[4] Nimal Nissanke. Component Security-Issues and an Approach,29th annual International computer software and Application Conference, 2005IEEE

[5] Binbin Qu,etc. A Developed Dynamic Environment Fault Injection Tool for Component Security Testing, Third IEEE International Conference on Secure Software Integration and Reliability Improvement, 2009
[6] Roshanak Roshandel, Somo Banerjee, etc. Estimating Software Component Reliability by Leveraging Architectural Models, ACM,2006
[7] I.Sommerville. software Engineering. Addison Wesley, 2001
[8] Judith Stafford, John D. McGregor. Issues in Predicting the Reliability of Composed Components, 5th ICSE Workshop on Component-Based Software Engineering Orlando Florida, 2002