EVALUATING TRUSTWORTHINESS OF SOFTWARE COMPONENT

Beni Suranto

Department of Informatics, Faculty of Industrial Technology, Universitas Islam Indonesia
Jalan Kaliurang Km.14,5 Sleman, Yogyakarta 55184
beni.suranto@uii.ac.id

ABSTRAK

Makalah ini membahas tentang konsep keterpercayaan komponen perangkat lunak yang merupakan salah satu pertimbangan utama bagi pengembang perangkat lunak dalam mengimplementasikan metode pengembangan perangkat lunak berbasis komponen. Pada bagian awal makalah, penulis menjelaskan mengenai konsep penggunaan ulang perangkat lunak dan kaitannya dengan keterpercayaan komponen perangkat lunak. Selanjutnya, bagian inti makalah membahas secara detail mengenai metode pengujian komponen perangkat lunak dan 4 (empat) metode yang dapat digunakan untuk mengevaluasi tingkat keterpercayaan dari komponen perangkat lunak. Di akhir makalah, penulis memberi gambaran mengenai proses seleksi komponen perangkat lunak pada domain industri.

Kata kunci: penggunaan ulang perangkat lunak, pengembangan perangkat lunak berbasis komponen, keterpercayaan komponen, seleksi komponen

1. COMPONENT-BASED SOFTWARE ENGINEERING

Today, software has an important role in many industrial systems. Software provides added value for products and can be used to effectively reduce the production cost. The use of software is now essential in manufacturing, medical systems, automotive, and process control industries. Generally, products in the current industry are systems consisting of software and hardware. The software part is a software system incorporating many software programs or applications that must cooperate to provide the intended functionalities without any defects. The most critical concern for software organizations is capability to deliver a software product on time, within budget, and to an agreed level of quality. In this context, underestimating software costs will lead to detrimental effects on the quality of the software product and thus on a company's business reputation and competitiveness. On the other hand, the opportunities to funds in other projects will be missed when the company overestimates the software cost(Andreessen, 2011).

Component-based Software Engineering (CBSE) is a popular concept in software engineering field which represents a technology for rapid assembly of flexible software systems. CBSE relies on software reuse and combines elements of software architecture, modular software design, software verification, configuration and deployment. Actually, software development approach with CBSE emerged from the failure of object-oriented development to support effective software reuse. Components can be considered to be standalone service providers and are more abstract than object classes. In CBSE, a software product are built as an assembly of software components already developed and prepared for integration. The main advantages of the this approach include increased productivity, effective management of complexity, a wider range

of usability and extendibility, a greater degree of consistency, and reduced time to market(Kaur & Mann, 2010).

CBSE adopts the component-based engineering method from other reengineering fields (e.g. mechanical or electrical engineering). In context of CBSE comes Component-Based Development (CBD) with the main task is to build systems from software units or components which are already built. By composing a system from prebuilt or existing components, this development method reduces both production cost and production time. Also, the already prebuilt components can be reused in many different software systems(Panunzio & Vardanega, 2009).

To realize the great benefits of CBD technology, it is necessary to have software components that can be easily reused and can be integrated in a systematic way. As CBSE is based on the concept of component. The most commonly used definition for software component was proposed by Szyperski et. al. (2002):

"*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties*".

For software engineers, the main challenge is reusing software components fot building the intended systems. A software component has to maintains its functionality as it deployed and executed after installation in different systems. Software engineers have to use a mechanism for connecting software components at run-time or dynamically. In the other words, a software component must be independently deployable. This approach allows software engineers to use the software components as and when required for maximizing the utilization of resources(Shareef & Pandey, 2012).

2. SOFTWARE COMPONENT TESTING

It is sure that where will be great benefits in effectiveness of project development when component based software engineering techniques are used, however, both reliability of selected components and safety when components communicate with each other should be concerned. Moreover, if software defects are discovered in the late part of life cycle of software development, great cost including time, labour and budget will be spent on correcting those software faults with no promise that those faults will be fixed perfectly. Thus testing in component based software engineering should be implemented during both individual component development and component integration (Bertolino, 2007).

On 4th June 1996, the Ariane 5 rocket veered off and exploded disastrously 40 seconds after initiation of its flight sequence, costing nearly $370 million directly. An Inquiry Board led by Professor J. L. Lions was convoked by the Chairman of CNES and the Director General of ESA to identify the reasons for the launch failure(Lann, 2007). One month later, an analysis report presented by the team demonstrated that insufficient software testing when software engineers reused software from the Ariane 4 as a component cause this explosion. The development team did not test the value of horizontal velocity which the Ariane 5 could reach 40 seconds after initiation to check whether that value might be out of calculation boundary set in the software of the Ariane 4 after changing the ignition hardware into a high initial acceleration system. Consequently, an exception failed to be caught when an out of calculation boundary value was past to the software method, which led to the crash of the Ariane 5 system catastrophically. The analysis report strongly recommended that entire simulations should be fully tested before any real mission. Unfortunately,

software engineers omitted those test cases, which led to the launch failure(Lions, 1996).

There are three steps through which component based software should be tested. First, each component should be tested fully when it is developed as an individual unit. Secondly, integration testing should be applied on subsystems which consist of no-defect-found components as single items after unit testing. Thirdly, once all the subsystems have been integrated into a whole system, the system should be tested fully and sufficiently to check whether all the components work well together in terms of their requirements. Additionally, systems which are developed by using the component based software techniques can also be tested by some other testing methodology such as stability testing, reliability testing, robustness testing, loading testing and so on which may all be helpful in identifying the quality of the systems.

The source codes of component may not be available in many circumstances such as using in-house components or purchasing commercial off-the-shelf components (Bertolino, 2007). Consequently difficulties may be brought into component testing because those software engineers who hardly familiar with the inner-construction of components can only use black-box testing instead of white-box testing, which implies that test cases may not be chosen properly or sufficiently. This in turn means that component software testing techniques should cover the area of non-availability of source code.

Garlan et. al (1995) identified several problems when they reuse some components to generate a new system. In their report, they demonstrated that there were a lot of troubles while they tried to integrate

components together; sometimes rework on the components might cost significantly to make sure that those components met their requirements and worked properly as a group. Moreover, the authors also reported that a lot of work should be done to test and maintain the integrated system especially when they attempted to generate appropriate and sufficient test cases because of the low level understanding of some reused components. Consequently, the stability and reliability of component based software can be greatly influenced and hardly controlled.

3. COMPONENT TRUSTWORTHI-NESS EVALUATION METHODS

3.1. Reference Model for Trustworthy Proof

By definition, Trusted component is "*a reusable element of software, it has a quality character which is designed and guaranteed*"(Alvaro, et. al, 2010). According to this definition, software engineers have a problem about how they guarantee and evaluate the trustworthiness of components.

JiuSong et. al (2009) propose a reference model which can be used to investigate trustworthy proof in component-based development process. They define trustworthy proof as "all the real facts that is with a specific form, certificated and used to prove the case of components' quality". They also define proof item as "the assembly of all the trusted components' trustworthy proof".

Based on the proposed reference model, there are two level of trustworthy proof: the first level proof item and second level proof item. second level proof item is a smaller granularity of first level.

Trustworthy proof have some specific characters: objectivity (must be an objective fact and independent of stakeholders' will), relevance (there must be significant a relationship between proof and quality of the component that needs to be verified), availability (the proof can be evaluated by a spesific procedure), and

diversity (the proof can exist in many different forms).

Considering from the view of software development life cycle, the process of developing the components affect the trustworthy of components. The trustworthy of components then will be reflected through its character and will define the user satisfaction. So, from this perspective, there are three aspect of the trustworthy proof to verify trusted components: trustworthy proof of development phase (to provide and ensure the trusted components during the process of componentsdevelopment, requirement analysis, design and realize), submission phase (to verify whether components are correctly usable), application phase (verify the quality components in the run-time environtment). The proposed trustworthy proof reference model is shown in Fig. 1.

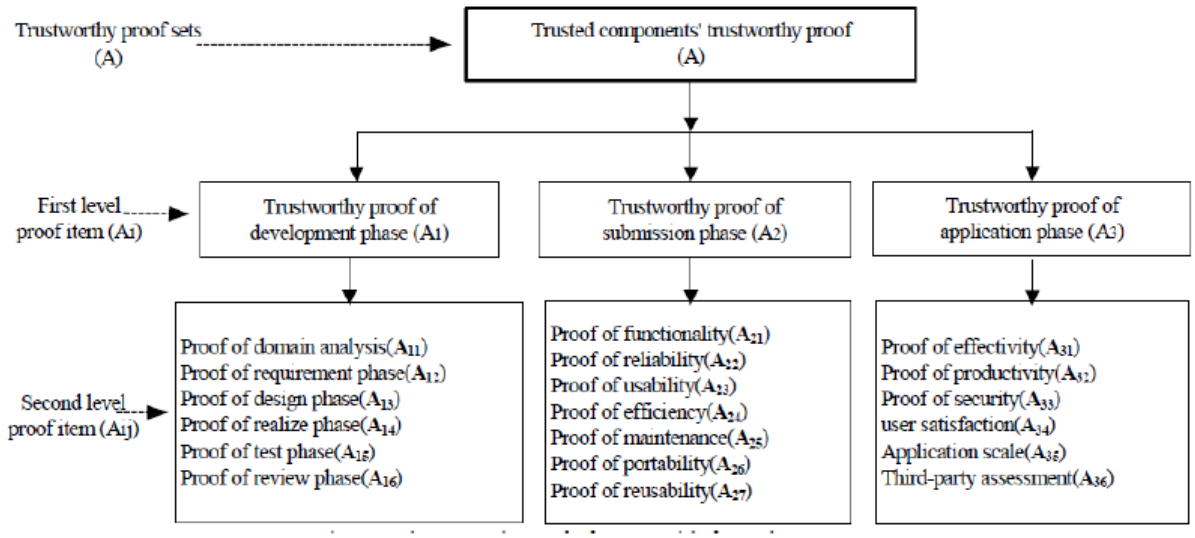

Figure 1. The trustworthy proof reference model(JiuSong, 2009)

There are two method defined in this model which can be used to obtain trustworthy proof in order to verify the trusted components:

a. Static obtaining methods: This method relatively easy to achieve, involves in more artificial participations (user feedback, expert review, the third party assessment).
b. Dynamic obtaining method: This method is more difficult to achieve, involved in fewer artificial participations (process simulation, automated testing, QoA monitoring).

3.2. A Formal Verification Model to Verify The Trustworthiness of Component Interface

When software engineers want to develop component-based systems, they consider components as black boxes, they can't access the inner structure of components. Software engineers can only access components information from their interfaces. According to this situation, specification of component interface need to be defined correctly, otherwise software engineers will have some problems when they want to integrate component to their system. Also, the correctness of specification of component interface has strong relationship with the effectiveness of the reusability of the system.

Dan & Jin (2009) propose a model which can be used as the basis for the verification mechanism of the trustworthiness of software components. They combine two powerful tools B Method and UML to model the component

interfaces so the component interfaces can be correctly verified.

The B method is a popular approach to specify system based on set theory that consider the safety and the reliability aspect. This method using some mathematical proofs for the basis of three main processes in the implementation stage of software development (specification, design, and coding) to ensure that the system is coherent and fault-free. One of the main objectives of B method is to formalize specification. This objective is significantly related with the requirement of correct specification of components interfaces. One of the advantage of using B method is that it uses abstract machine notations to model the component interfaces, so we can understand more easily about the specification of the component interface. Also, there are some powerful tools for B method (AtelierB, B-Toolkit, BEditor).

The first step to develop formal verification model is describe the component interface using UML class diagram and state diagram. Those diagrams can intuitively inform the detail information of component interface, from both syntactic and semantic aspect. From UML class diagram and state diagram of component interface then we use B abstract machine to define the formal specification of component interface. The final step of this method is verify the trustworthiness of component connection using B refinement mechanism [12].

In component-based software methodology, component have two kinds of interface to communicate with each other: required interface (to define what interfaces component requires from other components) and provided interface (to define what interfaces component can be accessed by another components). An interface need to be specified based on its data model and its operations. One tool that can be used to model the component interfaces is UML. Nowadays, UML is a de facto standard notation in object-oriented system development. The interface data model can be described by UML class diagram according to the definitions of its attribute and its operation. We can consider component protocols as a state set and we can use a UML state diagram to describe the usage protocol of component interfaces according to some related informations (pre and post conditions of the operation, call sequence of operations, transition rule of component state). The class diagram for a "Steam Boiler control system" benchmark problem is shown in Fig. 2. And the state diagram is shown in Fig. 3.

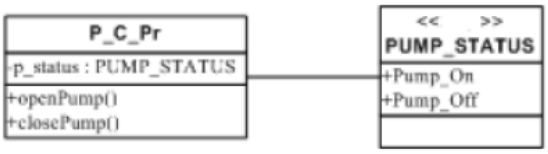

Figure 2. A class diagram for steam boiler control system (Dan & Jing, 2009)

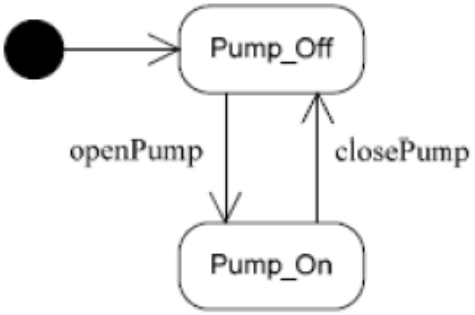

Figure 3. A state diagram for steam boiler control system(Dan & Jing, 2009)

From the class diagram and state diagram we can specify an interface using B abstract machine notation to describe both the static and the dynamic information of component interface. The B machine of the interface for the steam boiler control system is shown in Fig. 4.

```
MACHINE
  P_C_Pr
SETS
  PUMP _STATUS = {Pump_On, Pump_Off}
VARIABLES
  p_status
INVARIANT
  p_status∈ PUMP _STATUS
INITIALIZATION
  p_status:=Pump_Off
OPERATIONS
  openPump ≙
    PRE p_status = Pump_Off
    THEN p_status:= Pump_On
    END;
  turnOff ≙
    PRE p_status = Pump_On
    THEN p_status:= Pump_Off
    END
```

Figure 4. A B machine for steam boiler control system(Dan & Jing, 2009)

In this method, the most important criteria for the trustworthiness of the connection between two components is compatibility of their interfaces. The compatibility aspect is considered on three levels: syntactic level (the description of static information of component interface), semantic level (the description of dynamic behaviors of component interface operations), and protocol level (the description how to call the component interface operations).

In B method, refinement technique is used to create mathematical model of the system based on its abstract model. We can verify the compatibility of two component interfaces by verify the abstract machine of them. If we can prove that the "provided interface" machineis a correct implementation (i.e. refinement) of the "required interface" machine we can say that both of them are compatible each other. The abstract machine and the corresponding refinement is shown in Fig. 5.

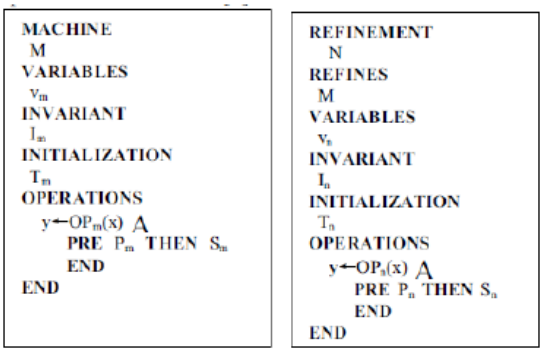

Figure 5. An abstrat machine and refinement for steam boiler control system(Dan & Jing, 2009)

3.3. Thrustworthiness Evaluation Method Using Entropy

Once software engineers finish their code, they need to perform design review to evaluate the code and remove defects in their code. Bacchelli & Bird (2013) found that almost all the software engineers included finding defects as one of the reasons for doing code reviews.

At present, its difficult to measure the trustworthiness of software components because there are very small number of standard methods and techniques to do that. Also, components are exists in different hierarchies in the system and could be applied in various domain of business.

Zhang et. al (2011) propose the method to measure the trustworthiness of software components using information entropy index as the parameter.

According to TCG, the system is trustworthy if its behavior and its results are always expected and controllable. Currently, various type of data are produced massively every day. Those data have important role in lot of business process in our daily life. Therefore, essentially we need to process data effectively and efficiently. In this context, there are four kinds of components in data processing area: data conversion components, data analysis components and data display components.

When we want to measure the trustworthiness of components we must consider about trustworthy proof. There are two kinds of trustworthy proofs need to be reported during the component development: process proofs and testing proofs. Testing proofs is reported in the testing environment. The component trustworthiness will be guaranteed only when both of process proofs and testing proofs meet the trustworthiness requirement. The component with guaranteed trustworthiness then can be stored in the component library for future use. The framework for trustworthiness measurement is shown in Fig. 6.

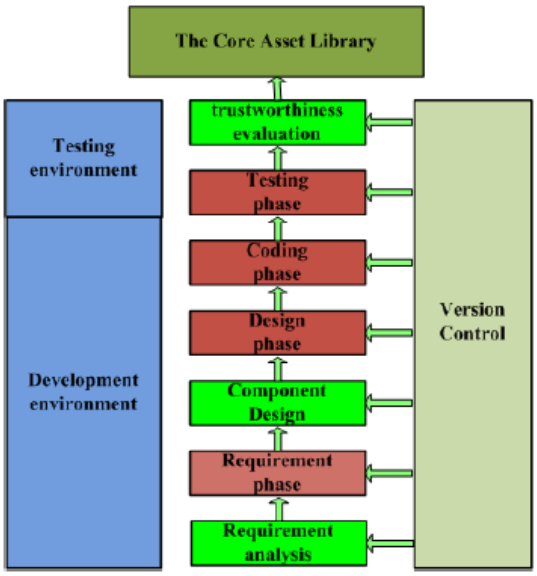

Figure 6. Trustworthiness measurement framework (Zhang et. al, 2011)

A system usually consists of some various factors and each factor in a system has an uncertainty. From this concept we can define the uncertainty of the system is the weighted average of factors' uncertainties. Claude E. Shannon in his 1948 paper "A Mathematical Theory of Communication" representing a measure of unpredictability of a system using the formula of entropy.

The formula: Entropy = $\sum \log pi$ (1)

Based on the definition of the trustworthiness we can ensure the system is trustworthy if its behavior and its result is always controllable and satisfy the expectation. In other word we can say that the trustworthiness level is equal with the match condition between the result from the system result and the user expectation. In the software development process we can consider the component as a function. We can ensure that there must be ascertained output data if input data have been ascertained. Therefore, the understanding level of the component can be verified based on the matching condition between the result and expectation.

This method suggest entropy for the criteria tomeasure the component trustworthiness. The correlation between trustworthiness and the entropy is negative, the component has high trustworthiness level if its entropy is small.

To measure the entropy of the component we must consider all of four stage In the component development stages: the component requirement stage, the component design stage, the component code implementation stage, and the component testing stage. We apply the entropy formula (1) to calculate the component entropy at every step. The trustworthiness tree in trustworthiness measurement is shown in Fig. 7.

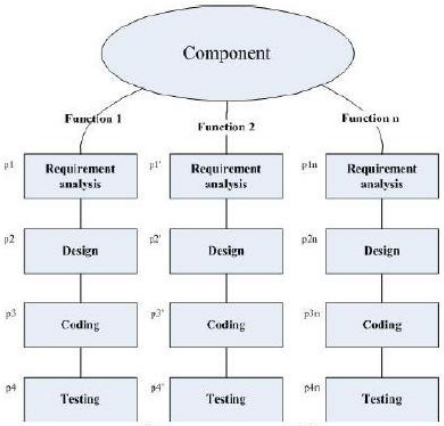

Figure 7. Trustworthiness tree(Zhang et. al, 2011)

3.4. Thrustworthiness Evaluation of Open Source Components

One of the main consideration of the integrators when developing software system using ready-made components is the quality of the component. When we develop system using Open Source Components (OCSs) we have to evaluate the reliability of OSCs. This is very important because if OSCs are not reliable they can cause some significant faults and reliability problems to the system. Evaluation of the reliability of OSCs is quite difficult because the only available artifact is the source code.

Immonen & Palviainen (2007) propose evaluation and testing method to validate the trustworthiness of OCSs. They define the software trustworthiness as "*the degree of confidence that exist that it meets a set of requirement*". To evaluate the trustworthiness in software development process, they suggest two type of evaluation: The technical and the non-technical evaluation.

The technical trustworthiness evaluation verify the software trustworthiness using quantitative reliability analysis in three level: the component level, the architecture level, and the system level. The non-technical trustworthiness evaluation combine some artifacts such as history and reputation of OSC, the evaluation of user communities, quality of OSC development process, and the property of OSC provider (see Table 1).

Table 1. The levels of trustworthiness evaluation method (Immonen & Palviainen, 2007)

Levels	Technical evaluation	Non-technical evaluation
Component	• Predicted reliability of new components • Reliability testing of OSCs • Analyzed reliability of OSC with the help of testing	• OS community and its reputation in a domain • Component development and certification processes • Component reputation and user communities • Understandability of a component • Component history, evolution and license
Architecture	• Predicted reliability of the system with OSCs	• Dependencies, constraints • Compliance with standards
System	• Reliability testing of the system with OSCs	• Component installation • Delivery

The RAP method is used as the basis for the technical part of the trustworthiness evaluation method. The RAP method is extended to support model-based reliability analysis and implementation-based reliability testing. Model-based reliability analysis is used to evaluate the level of reliability at two levels, the component and architecture levels. At the component level, the analysis use the probability of failure before component implementation to predict reliability. The probability of failure of components then will be combined with architectural models and system execution paths to simulate the system.

Implementation-based reliability testing use unit tests to evaluate reliability at the component level and tests the system when the component is integrated in the system.

New method and tool was developed by Immonen and Palviainen based on the RAP tool to support reliability testing of OSCs. Eclipse (http://www.eclipse. org/) was chosen for this method because Eclipse is able to promote interopability of tools. Also, Eclipse provide an extensible application framework which is very useful for software engineer when thy want to build a software system. The input for reliability evaluation is architectural model which using UML and the testing environtment is an open Eclipse Test and Performance Tools Platform (TPTP)

(http://www.eclipse.org/tptp/). Reliability analysis tool in Eclipse is shown in Fig. 8.

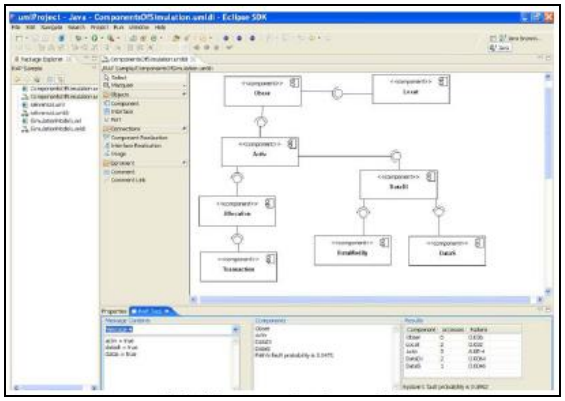

Figure 8. Reliability analysis tool in Eclipse(Immonen & Palviainen, 2007)

5. COMPONENT SELECTION IN INDUSTRIAL PRACTICES

At present, the software industry recognize the approach of reusing third-party software to build software system as an significant success factor. Torchiano & Morisio (2004) in Ayala et. al (2011) define an OTS component as "*a commercially available or open source piece of software that other software projects can reuse and integrate into their own products*". One kind of software components is Commercial-Off-The-Shelf (COTS) software which acquired by a fee. Companies use COTS to improve their software development process and achieve some great advantages for their business process: cost and time efficiencies, technology adoption acceleration, and better quality software. Nowadays, there are a lot of COTS available for various application areas.

One of the most important things in reusing COTS is the ability of the components integrators to evaluate which COTSs are appropriate for the system. Currently, software companies are still having problems about how to select appropriate COTSs for their system. The evidence shows that most of the proposed methods from the "research area" are rarely used in the industrial practice.

Ayala et. al (2011) investigate the common practices of the COTSs selection process done by 20 software companies in Spain, Norway, and Luxembourg. Form their investigation, it was found that the most popular process done by software companies to select COTSs in the soffware system development is informal evaluation . Common process used by companies to select COTSs listed in Table 2.

Most of the companies did not use any formal evaluation method to select COTSs for their system. Also, most of companies select the COTSs without using the documentation of the COTSs for their subsequent comparison. For most of companies, there are two main things that influence the evaluation process: their previous experience with the COTSs and the critically of the COTSs with in the system to be built. Sometimes, the companies just use the opinions about the COTSs from the experiences of people for the basis of the evaluation process.

Some companies hired consultants for their COTSs evaluation process, but most of them only hire consultants for critical projects. Some companies stated that they hired a consultant to minimize the potential risks in critical projects. Some companies follows specific procedures to ensure the quality of their system but some other companies did not have a specific procedure, they only use a spreadsheet tool to support the evaluation process.

In general, all companies consulting to the COTSs provider to search the COTSs information. Some methods used by the companies to evaluate the COTSs are listed in Table 3. From information in Table 3 we found that the most popular method used by the companies is testing of the basic functionalities of the COTSs.

Table 2. Processes to evaluate COTSs (Ayala et. al, 2011)

Code	Description
Eval-A	**Informal procedures**. Respondents neither used nor knew of any formal procedure or method for selecting components.
Eval-B	**No evaluation**. Respondents recognized that no evaluation of candidates was performed as they directly chose one component they had used before.
Eval-C	**Hire support to perform the task**. The respondents emphasized that a company was hired to lead the application of the Open Source Maturity Model (OSMM) for evaluating candidate components.
Eval-D	**Outsource the task**. A company was hired to perform the evaluation of candidate components (the procedure used was unknown to the respondent).
Eval-E	**Established evaluation procedures**. The respondent used established procedures based on the Kano model to evaluate candidate components.
Eval-F	**Tool supported evaluation**. The company developed a tool to partially support the candidate components' evaluation.

Table 3. Methods to evaluate COTSs (Ayala et. al, 2011)

Code	Description
Info-A	**Straightforward testing of trivial functionality**. Respondents just tested some basic functionality of candidate components to have an overview of whether the component might work as expected.
Info-B	**Building a prototype**. Respondents spent some time and effort building a prototype to check the behavior of the component in the expected environment.
Info-C	**Asking for people's experiences**. Respondents based their evaluation on previous experiences, mainly of people inside the company.
Info-D	**Just reading functional information of the component**. Respondent does not even test the component functionality but only checks the technical documentation available.
Info-E	**Hiring a consultant**. Respondent did not know how components were assessed as the task was hired and the process was unknown
	Total

6. CONCLUCIONS

The use of reusable software component have some great advantages and have a significant role in current software development practices. This paper discuss about some important points related with the concept of the trustworthiness of software component and we investigate some proposed methods to evaluate the trustworthiness of software components. In this paper we also discuss about the process to select components in industrial practices.

We found that there is still a gap between "research" area and "industry "area". The further research is still needed to minimize this gap. We are interested in applying the proposed methods in some real project so we can verify wether the proposed methods are appropriate to accommodate real industrial needs.

REFERENCES

Alvaro, A., Santana de Almeida, E., & Romero de Lemos Meira, S. (2010). A software component quality framework. *ACM SIGSOFT Software Engineering Notes*, *35*(1), 1-18.

Andreessen, M. (2011). Why Software Is Eating The World'. *Wall Street Journal*, *20*.

Bertolino, A. (2007, May). Software testing research: Achievements, challenges, dreams. In 2007 Future of Software Engineering (pp. 85-103). IEEE Computer Society.

Immonen, A., & Palviainen, M. (2007, October). Trustworthiness evaluation and testing of open source components. In Quality Software, 2007. QSIC'07. Seventh International Conference on (pp. 316-321). IEEE.

Ayala, C., Hauge, Ø., Conradi, R., Franch, X., & Li, J. (2011). Selection of third party software in Off-The-Shelf-based software development—An interview study with industrial practitioners. Journal of Systems and Software, 84(4), 620-637.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). The unified modeling language user guide. Reading, UK: Addison Wesley.

Dan, W., & Jing, Z. (2009, April). A Formal Verification Model for Trustworthiness of Component Interface. In Networks Security, Wireless Communications and Trusted Computing, 2009. NSWCTC'09. International Conference on (Vol. 2, pp. 643-646). IEEE.

Garlan, D., Allen, R., & Ockerbloom, J. (1995, April). Architectural mismatch or why it's hard to build systems out of existing parts. In Software Engineering, 1995. ICSE 1995. 17th International Conference on (pp. 179-179). IEEE.

JiuSong, H., Hong, H., QinBao, S., & KeGang, H. (2009, December). Reference Model of Trustworthy Proof for Trusted Components. In Future Information Technology and Management Engineering, 2009. FITME'09. Second International Conference on (pp. 136-139). IEEE.

Jones, G., & Prieto-Diaz, R. (1988, October). Building and managing software libraries. In *Computer Software and Applications Conference, 1988. COMPSAC 88. Proceedings., Twelfth International* (pp. 228-236). IEEE.

Kaindl, H. (2013, December). Software Reuse Based on Business Processes and Requirements. In Software Engineering Conference (APSEC, 2013 20th Asia-Pacific (pp. 85-86). IEEE.

Leach, R. J. (2012). Software Reuse: Methods, Models, Costs. AfterMath.

Le Lann, G. (1997, March). An analysis of the Ariane 5 flight 501 failure-a system engineering perspective. In Engineering of Computer-Based Systems, 1997. Proceedings., International Conference and Workshop on (pp. 339-346). IEEE.

Lions, J. L. (1996). Ariane 5 flight 501 failure.

Lyu, M. R. (2007, May). Software reliability engineering: A roadmap. In 2007 Future of Software

Engineering (pp. 153-170). IEEE Computer Society.

Meyer, B. (2003, May). The grand challenge of trusted components. In Software Engineering, 2003. Proceedings. 25th International Conference on (pp. 660-667). IEEE.

Morris, J., Lee, G., Parker, K., Bundell, G. A., & Lam, C. P. (2001). Software component certification. Computer, 34(9), 30-36.

Panunzio, M., & Vardanega, T. (2009, August). On component-based development and high-integrity real-time systems. In *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA'09. 15th IEEE International Conference on* (pp. 79-84). IEEE.

Snow, K. Z., Monrose, F., Davi, L., Dmitrienko, A., Liebchen, C., & Sadeghi, A. R. (2013, May). Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization. In Security and Privacy (SP), 2013 IEEE Symposium on (pp. 574-588). IEEE.

Zhang, X., Wu, H., & Lu, Y. (2011, April). The Exploration of the Component's Trustworthiness Measurement Method in the Data Processing Domain. In ITNG (pp. 186-190).

Kaur, A., & Mann, K. S. (2010). Component based software engineering.*International Journal of Computer Applications*, *2*(1), 105-108.

Szyperski, C. (2002). *Component software: beyond object-oriented programming*. Pearson Education.

Shareef, J. W., & Pandey, R. K. (2012). Component-Based Software Development with Component Technologies: An Overview.