**Available Online at www.ijcsmc.com**

**International Journal of Computer Science and Mobile Computing**

**A Monthly Journal of Computer Science and Information Technology**

**ISSN 2320–088X**

*IJCSMC, Vol. 3, Issue. 6, June 2014, pg.535 – 541*

**RESEARCH ARTICLE**

# Component Compatibility in Component Based Development

**Dr. Hardeep Singh[1], Anitpal Kaur[2]**

[1]Guru Nanak Dev University Amritsar

[2]Guru Nanak Dev University Amritsar

Hardeep_gndu@rediffmail.com; Anita.sodhi@yahoo.com

***Abstract- This paper presents a research on component compatibility in component based development. Component-based software engineering is a process that emphasizes the design and construction of computer-based systems using reusable software components. Commercial components repositories contain hundred thousand components that make component selection an extremely difficult and time expensive task. Often component selected by functional features are incompatible or the integration effort required is too high. Adding a selection of components based also on compatibility can simplify the integration task. This work focuses on the study of component compatibility using various metrics as parameters.***

***Keywords- Component Based Software Engineering, Components, Compatibility, Commercial Components, Software Reuse***

## I. INTRODUCTION

Primary role of component-based software development is to address the development of systems as assembly components. The developed components are reusable entities, and Component –based development facilitate the maintenance and upgrading of systems by customizing and replacing components. The main advantages of Component based development are reduced development time, cost and efforts along with several others. These advantages are mainly contributed by the reuse of already built-in software components. Generally a component can be defined as an independent and replaceable part of a system that fulfills a clear function. It works in the context of well-defined architecture and can communicate with other components through its interfaces. [2]

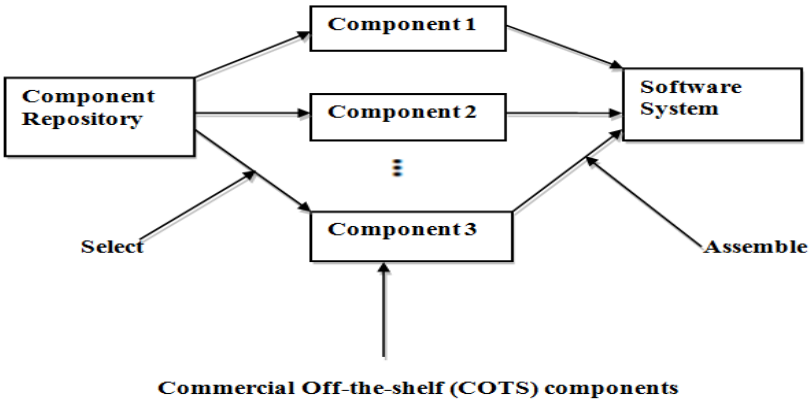



Fig.1 Component based development. [2]

A. Component

A component is an independent and replaceable part of a system that fulfills a clear function. Components are categories as in component based systems. Conceptual components are the components at the analysis and design phase. Implementation components such as source code files, data files etc. Deployment components involved in an executable system, such as dynamic libraries and executables. It was motivated by the frustration that objects oriented development had not led to extensive reuse as originally suggested. Components are more abstract than object classes and considered to be stand-alone service provider. [12]

*B.* Component Compatibility

Compatibility of two components is determined by three requirements firstly present the same operational interface to its environment, secondly component should be able to read and write the same data and conform to the semantics in all interactions in which they are engaged. The replacement component's provided features should obviously be at least the same as those of the current one, otherwise its clients will not be able to successfully communicate with it. Component compatibility defines two levels. Strict compatibility requires that replacement component should be a subtype of the current one. Relaxed level compatibility referred as contextual compatibility because it takes into account two important aspects of the environment in which the current component is deployed. One aspect is which of the current component's provided features are actually used by other components in the given application configuration. Secondly requirements of the current component are not considered. [11]

## II. RELATED WORK

Beibei, Xu. et al. (2012) [1] in paper "Research on software reuse methods based on the object-oriented components" has described component technology as the core technology to support software reuse. Software reuse is considered as an effective method of avoiding duplication of work, improving the efficiency and quality of the software development .The relationship between object-oriented methods and software reuse has been presented .The method based on the component in the object-oriented paradigm is very important to realize software reuse technology .The process of extracting components in order to realize software reuse has been discussed. Software reuse has a great importance in the generality, maintenance and flexibility of development system.

Ponomarenko, A. et al. (2011) [11] in paper "Automatic backward compatibility analysis of software component binary interfaces" has discussed a problem of ensuring backward compatibility when developing software components and software platforms. Breakage of the compatibility may result in crashing or incorrect binary level behavior or inability to build source level applications targeted at an old version of a dependent software component when the application are used with a new version of the component. The issues that cause compatibility problem have been considered. Eight different types of possible changes in library source code that can cause breakage of binary interface have been identified. A method based on combined

binary and header analysis and a corresponding tool for automatic detection of all the identified types of issues has been suggested .The tool can help library developers to automatically catch any unintended change in the library and thus maintain backward binary compatibility guarantee for the users.

Jha, M.et al. (2011) [5] in paper "A comparison of software reuse in software development communities" has discussed software reuse as one of the most important areas for improving software development productivity and the quality of software. Software reuse is limited to certain domains and not widespread across the software industry. A survey on issues and concerns in software reuse in the conventional software engineering community and in the software product line community to compare and contrast the results for similarities and differences in software reuse philosophy has been examined. Outlines some of the identified differences and similarities of issues and concerns in software reuse in both communities and what one community can gain from the other to overcome the identified software reuse problems. The availability of professional personnel with knowledge and experience in the area of software reuse has been considered very important asset for reuse.

Kebir,S. et al.(2012) [7] in paper "Quality-centric approach for software component identification from object-oriented code" has taken components and connectors as the main building blocks for software architectures .Components can either be created from scratch or reused in the design phase of a software system .When component reused they can either exist on component shelves or identified from exiting software systems so component identification is one of the primary challenge in component based software engineering .The two main limitation of Object oriented systems to identify a component based on high cohesion and loosely coupled set of classes has been discussed . A quality centric component identification approach to measure the quality of a identified component to overcome these two limitation has been proposed .Quality centric approach identify components based on explorative and requirement driven strategy. A method for identification of the internal structure of a component has been discussed.

Daniel, L. et al. (2012) [8] in paper "An investigation on the impact of MDE on software reuse" has investigated the impact of model-driven engineering on software reuse. Model-driven engineering can increase software reuse by reducing the amount of hand-written code. Model –driven engineering achieve reuse through generating programming, domain –specific languages and software transformations which not only save considerable effort in repetitive tasks. Model-driven engineering increases the abstraction level through modeling and code generations but new assets like models, languages, transformations and code generators must be taken into account. Model –driven engineering can reduce the semantic gap between the problem and the solution. Model –driven approach effectively increase the abstraction level where reuse takes place but it may also increase complexity and productivity loss.

Niranjan,P. et al.(2013)[10] in paper "Dynamic grading of software reusable components for effective retrieval of components" has presented classification and retrieval of software reusable components by using integrated classification and also the selection of best component by using rank of the component. The retrieval of an optimal relevant component has been considered one of the key issues. The selection of proper retrieval technique is very essential to use software reusable components from the reuse repository. In the absence of proper retrieval mechanisms the software reuse becomes ineffective. A grading algorithm has been proposed for the classification and retrieval of best software reusable component.

Kakarontzas,G. et al.(2012) [6] in paper "Extracting components from open source the component adaptation environment(COPE) approach" has represented open source software an extremely valuable resource that is reused systematically almost in every software project. The reuse of open software components is restricted to ready –made components and developers who want to reuse code that exists in open source software projects but is not offered as a black-box component often resort to copying existing code and adapting it in their projects. The component adaptation approach for component extraction seems to be effective and easy to use because it is not entirely automated it requires the participation of a knowledge expert.

Cai.et al. (2000) [2] in paper "Component based software engineering: technologies, development frameworks and quality assurance schemes" has addressed current component-based software technology, their advantages and disadvantages and the features they inherit. Quality assurance issues for component-based software has been addressed which covers component requirements analysis, component customization and system architecture design, integration, testing and maintenance.

Chengjion,W.et al.(2012)[9] in paper "The reusable software component development based on pattern –oriented" has thought software reuse as a key strategy for reducing development costs and improving quality. Pattern is a way of reuse abstract knowledge about a recurrent problem in a particular context and its solution. The simplicity of a pattern and its small size make it easy to understand, integrate and reuse. It is necessary to link every low level component to the higher level architecture, whereby developers can trace which components satisfy business goal or system requirements because developed software components give arise to the problem that developers do not know why and how they get there. Component model is defined in structured and formal way; the trace is expressed as composition of links. The semantics of links is helpful for developers to reason about the tasks performing with the linkage.

Duszynski,S. et al.(2011) [4] in paper "Analyzing the source code of multiple software Variants for reuse potential" has presented a scalable reverse engineering technique for assessing the reuse potential among multiple system variants. The technique provides an overview of commonality distribution in the whole analyzed system family and allows for detailed goal-driven refinement of the analysis results. Variant analysis, a scalable reverse engineering technique that aims at delivering exact information. It supports simultaneous analysis of multiple source code variants and enables easy interpretation of the analysis results.

Frankes.w and terry.c (1996) [14] in paper "Software reuse metrics and models" has suggested six types of metrics and models. As organizations implement systematic software reuse programs to improve productivity and quality, they must be able to measure their progress and identify the most effective reuse strategies. This is done with reuse metrics and models. [14]

Mahmood,A.K.et al.(2011) [9] in paper "A mixed method study to identify factors affecting software reusability in reuse intensive development" has considered the benefits of software reuse like reduction in cost, effort and time, better productivity and resource efficiency. Software reuse is employed in reuse intensive software development such as component based software development and software product line.

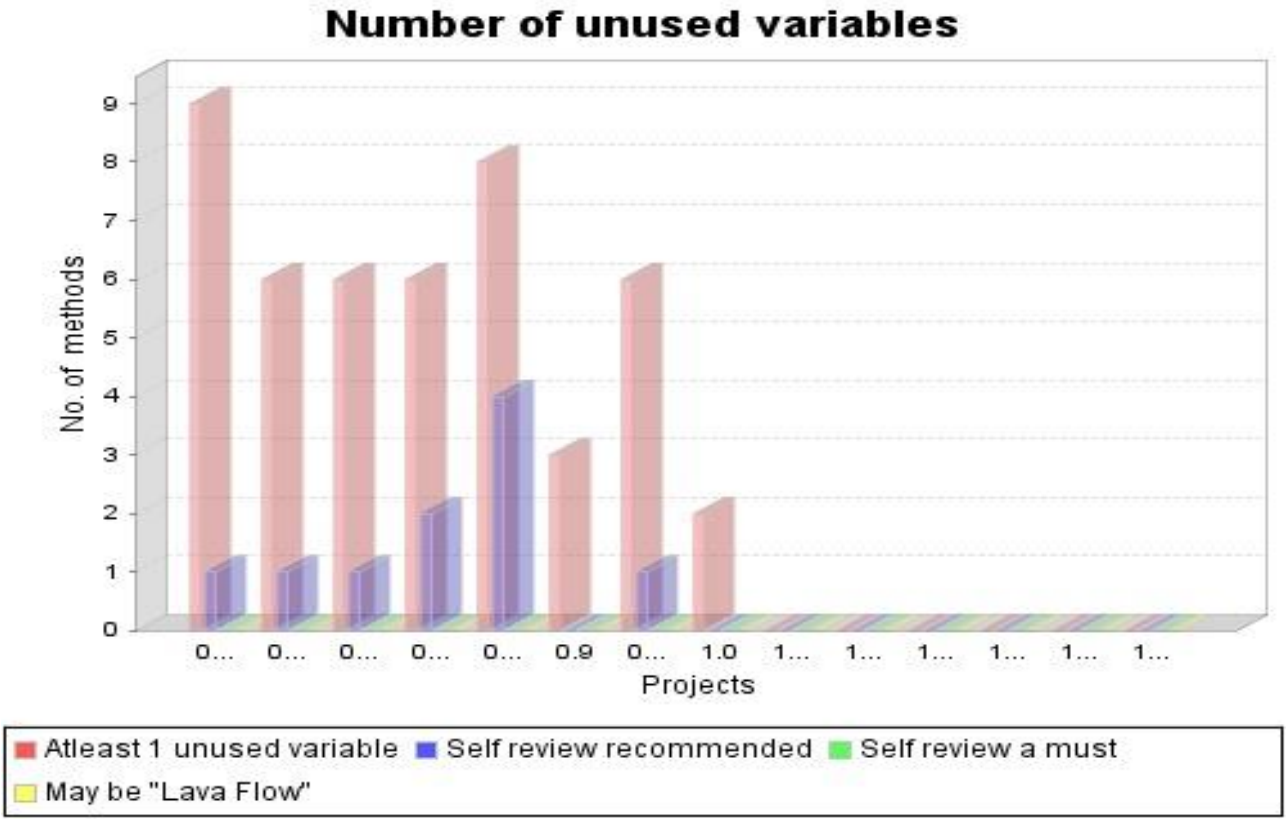
## III. RESULTS AND DISCUSSION

The following parameter table is used to measure compatibility.

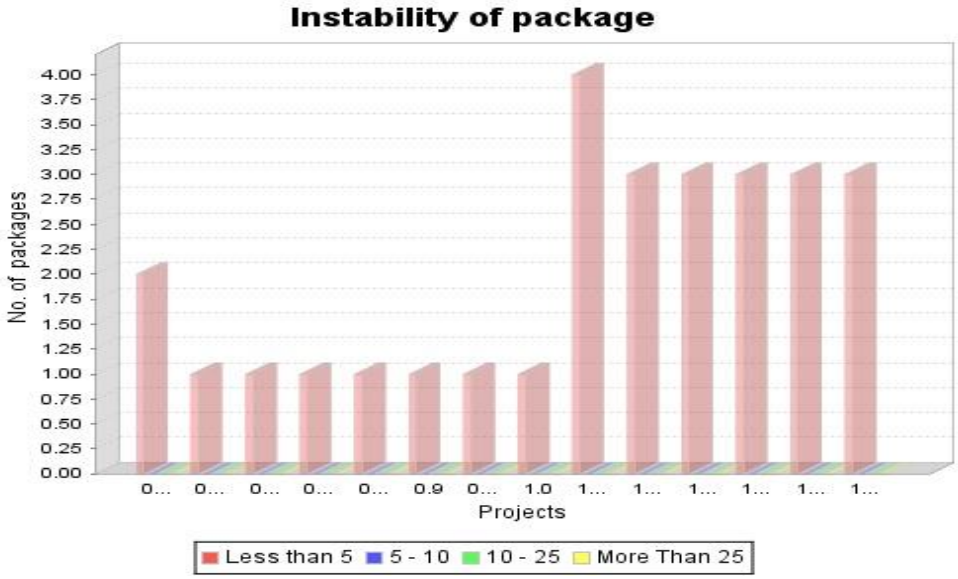
Fig.2 Parametric table

| Parameter Name |
|---|
| Number Of unused variable |
| Number of conditional statements |
| Number of recursive calls |
| Response of a class |
| Number of inner classes |
| Number Of non-private fields |
| Number of private methods |
| Instability of package |
| Dependency principle |
| Abstractness of package |

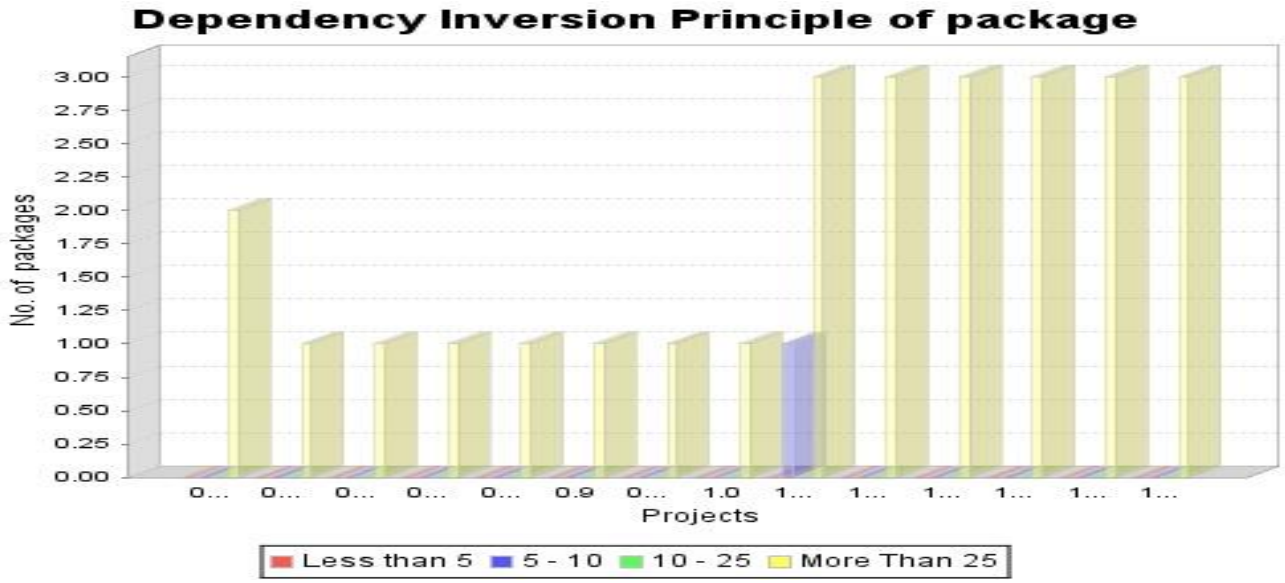



The above diagram depicts the number of projects in the X-direction and number of methods in the Y-direction. Metric value of unused variable should be minimized. Unused variables are a waste of space in the source. A decent compiler won't create them in the object file. Unused parameters when the functions have to meet an externally imposed interface are a different problem they can't be avoided as easily because to remove them would be to change the interface. The project at 0.4 has maximum number of unused variables and projects ranging from 1.1 to 1.6 have no unused variables so they are best for reuse.

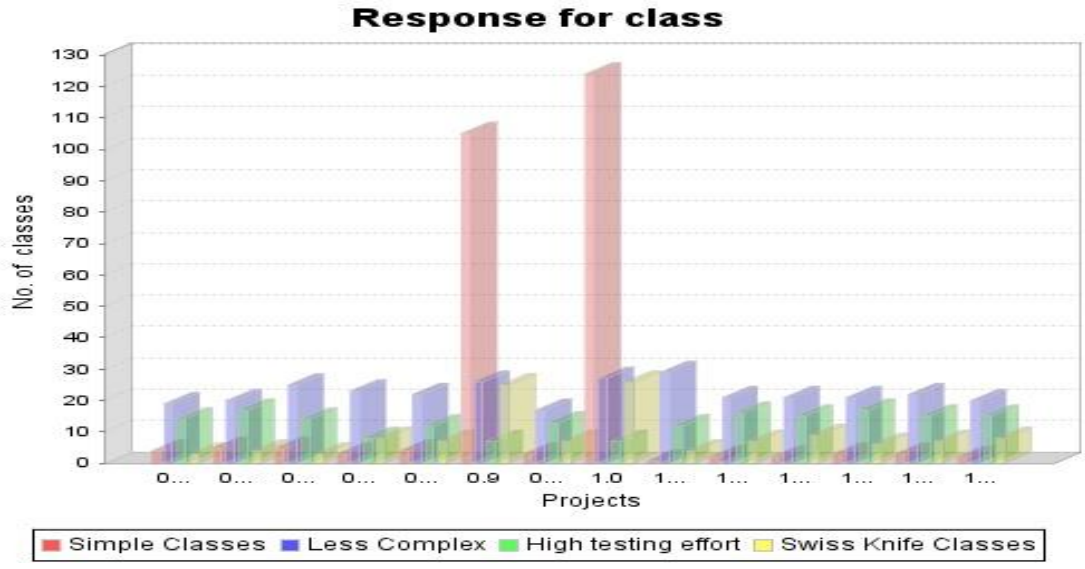


The above diagram depicts the number of projects in the X-direction and number of packages in the Y-direction. The project at point 1.1 shows the maximum instability of project and projects ranges from 0.5 to 1.0 show the constant instability. The ratio of efferent coupling (Ce) to total coupling (Ce + Ca) such that I = Ce / (Ce + Ca). This metric is an indicator of the package's resilience to change. The range for this metric is 0 to 1, with I=0 indicating completely stable package and I=1 indicating completely unstable package. The highly coupled system is considered as not good because it is hard to make changes in a system which is strongly coupled. So the projects ranging from 0.5 to 1.0 are most compatible.




The above diagram depicts the number of projects in the X-direction and number of packages in the Y-direction. The dependency inversion principle refers to a specific form of decoupling software modules. When following this principle, the conventional dependency relationships established from high-level, policy-setting modules to low-level, dependency modules are inverted thus rendering high-level modules independent of the low-level module implementation details. The principle states high-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions. The principle inverts the way some people may think about object-oriented design, dictating that both high- and low-level objects must depend on the same abstraction. Metric value of this parameter should be high. The projects ranging from 0.4 to 1.0 shows lowest dependency inversion principle and projects ranging from 1.1 to1.6 shows highest dependency inversion principle so they are most suitable for reuse.



The above diagram depicts the number of projects in the X-direction and number of classes in the Y-direction. Response for class is the count of the set of all methods that can be invoked in response to a message to an object of the class or by some method in the class. Methods in a class can be used as measures of communication. It is the number of methods of the class plus the number of methods called by any of those methods. If a large numbers of methods are invoked from a class (RFC is high) testing and maintenance of the class becomes more complex. Metric value should be low.

## IV. CONCLUSION

This research work focuses on the components compatibility when we reuse entire or some parts of the developed software. Software reuse is an important technology which can avoid repeated labor, improve quality and productivity. As component base development reuse components from different software's at a time but each software engineer has used its own naming conventions and methods so it becomes difficult to reuse the software directly. So it is required to convert the components into reusable form. Thus component compatibility becomes important part in component based development. Therefore this work focuses on the components compatibility which occurs during the software reusing time. The overall goal of this work is to select only those components which come up with optimal values when they are going to reuse.

## REFERENCES

[1] Beibei, Xu, Wang Haitao, and Zhang Fengwang. "Research on Software Reuse Methods Based on the Object-Oriented Components." In Computational Intelligence and Communication Networks (CICN), 2012 Second International Conference on, pp.1857-1860.IEEE, 2012.

[2] Cai, Xia, Michael R. Lyu, Kam-Fai Wong, and Roy Ko."Component-based software engineering: technologies, development frameworks, and quality assurance schemes." In Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific, pp. 372-379. IEEE, 2000.

[3] Chengjion, Wang. "The Reusable Software Component Development Based on Pattern-Oriented." In Computational Intelligence and Communication Networks (CICN), 2012 Fourth International Conference on, pp. 552-555. IEEE, 2012.

[4] Duszynski, Slawomir, Jens Knodel, and Martin Becker. "Analyzing the source code of multiple software variants for reuse potential." In Reverse Engineering (WCRE), 2011 18th Working Conference on, pp. 303-307. IEEE, 2011.

[5] Jha, Meena, and Liam O'Brien. "A comparison of software reuses in software development communities." In Software Engineering (MySEC), 2011 5th Malaysian Conference in, pp. 313-318. IEEE, 2011.

[6] Kakarontzas, George, Ioannis Stamelos, Stefanos Skalistis, and Athanasios Naskos. "Extracting Components from Open Source: The Component Adaptation Environment (COPE) Approach." In Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on, pp. 192-199. IEEE, 2012.

[7] Kebir, Selim, Abdelhak-Djamel Seriai, Sylvain Chardigny, and Allaoua Chaoui. "Quality-Centric Approach for Software Component Identification from Object-Oriented Code." In Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on, pp. 181-190. IEEE, 2012.

[8] Lucrédio, Daniel, Eduardo Santana de Almeida, and Renata PM Fortes. "An investigation on the impact of MDE on software reuse." In Software Components Architectures and Reuse (SBCARS), 2012 Sixth Brazilian Symposium on, pp. 101-110. IEEE, 2012.

[9] Mahmood, A. K., and Alan Oxley. "A mixed method study to identify factors affecting software reusability in reuse intensive development." In National Postgraduate Conference (NPC), 2011, pp. 1-6. IEEE, 2011.

[10] Niranjan, P., and CV Guru Rao. "Dynamic Grading of Software Reusable Components for Effective Retrieval of Components."

[11] Ponomarenko, Andrey, and Vladimir Rubanov. "Automatic backward compatibility analysis of software component binary interfaces." In Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on, vol. 3, pp. 167-173. IEEE, 2011.

[12] "Component Based Development Process" [Online available]:http://en.wikipedia.org.