Efficient Component Based Software Engineering using the TCEM Methodology and the TCET Tool

Santokh Singh, Cheung Ling Kelly Yu
Computer Science Dept, University of Auckland, Private Bag 92019, Auckland, New Zealand
{santokh@cs, cyu024@ec}.auckland.ac.nz

Abstract

Currently there are no comprehensive tools that can be used to harness the full potential of Component Based Software Engineering (CBSE). In this paper, we describe a novel component-based software development methodology, called the "Total Component Engineering Methodology" (TCEM) that we have conceived, formulated and tested that can be efficiently and effectively applied to every phase of the CBSE process. We also describe a novel and comprehensive visual tool called the "Total Component Engineering Tool" (TCET) which uses enhanced visual notations and features to efficiently support our new development methodology. This novel tool and methodology complement each other and can be used to produce high-quality component based software that is highly understandable, scalable, maintainable and reusable.

1. Introduction

Complex and compound software systems that carry out sophisticated tasks are currently becoming increasingly important, useful and popular. However the development of such software systems has become increasingly difficult to control, costly to manage and hard to understand and refactor [1]. Current Component Based Software Development (CBSD) techniques still cannot address all these problems though they have been advocated as possible solutions to the design, development, management and control of such systems [2]. No total CBSD methodology currently exists can be utilized to identify and reuse systemic components starting from the initial phase of Requirements Engineering itself and stretching beyond the delivery, deployment and maintenance stages. Moreover there is a clear lack of comprehensive visual tools to support this type of total component-oriented software development life cycle, i.e. one that totally, efficiently and effectively support the development in all phases of the SDLC using components.

2. Motivation

Software components are becoming more and more popular in software development because they modularize the code more effectively, are more understandable and allow for better reuse of code compared to the methodologies that do not use components in development [16]. However all the methodologies so far only focus on certain phases or specific areas in the component-oriented software development process. Also there is a clear lack of comprehensive visual tools to support a total component oriented software development life cycle, i.e. tools that totally, efficiently and effectively support the development in all phases of the SDLC using informative visual notations and components.

A number of Component-Based Software System (CBSS) development methodologies, including COMO [3], Catalysis [4], AECM [5] and KobrA [6], provide processes and notations to support CBSD. However, coding the software components without informative visual notations or comprehensive auto-code generation can be tedious and difficult. Using our tool, for instance, automatic code generation can be achieved because we use an informative Visual Language [7]. It is wasteful for software engineers/computer scientists to do things that machines can do automatically. The software engineers/computer scientists can focus more on refining the business logic and introducing new ideas and concepts rather than wasting time on the coding of syntax or reinventing the wheel like rewriting commonly used or existing code e.g. for persistency or security purposes. Also, if we were to use meaningful visual languages, including rich drag and drop features, users can design, develop and refactor software components and systems more easily, more accurately and faster.

Without proper development tools and visual notations to depict the software components in Component-Base Software Systems (CBSS), the designs can easily become disordered and disorganized, and this is more glaring if the system is very large like enterprise systems. Therefore, we are motivated and convinced that it is truly necessary and beneficial to have both a comprehensive methodology that can be used in all the phases of Component Based Software Development and an excellent Component-Based Software Engineering visual tool to support this methodology.

3. Background

Software components are blocks of code with business logic and have interfaces to define its functions. They can be encapsulated in different standards but the same type of software components should have the same architecture. The existing technology of creating software component include Microsoft® .NET or COM/COM+™ (Component Object Model) [8], J2EE (Java™ 2 Platform Enterprise Edition) or EJB (Enterprise JavaBeans™) [9], CORBA [10].

Component based software development (CBSD) approach is a concept that states that software systems should be developed with the use of software components [11]. Component based software systems (CBSS) are developed by selecting appropriate existing components [14] [15] or creating new ones to satisfy certain systemic requirements and assembling the software components into the architecture [17]. Current CBSD methodologies, e.g. COMO, Catalysis, KobrA and AECM cannot identify and use software components in all their SDLC phases, are not totally comprehensive and do not have sufficient tool support.

4. Total Component Engineering Methodology (TCEM)

The Total Component Engineering Methodology (TCEM) is a new CBSE methodology that we have conceived and developed to overcome the shortcomings and limitations encountered in current development methodologies. TCEM can be used to rapidly, efficiently and effectively produce component based software. This novel methodology can be used to guide software developers to constructively reason about how to identify, construct and deploy software components in software systems starting from the first phase in SDLC right to the end including deployment and maintenance of software systems.

4.1 System's Requirement Engineering with TCEM

For the TC (Total Component) Software Requirement Engineering phase (TCSRE) we introduce a new concept called "Early Components" (EC) in the requirement engineering phase. Currently the early aspects concept has been used successfully for the aspect-oriented software development. Rather than just decide on the type of aspect early in the development phase, we believe that it is a good idea to identify and determine (if possible) the component, during the early phases of TCSRE. The "Early Components" (EC) concept requires developers to identify any potentially useful software components e.g. authentication component, in the use case diagram of the CBSS during the requirement phase itself. By applying the "EC" concept the overall view of the software system become more visible and clearer even in the early development stages.

4.2 Analysis and Design using TCEM

We introduce the very useful concept of a Component Set (CS) here. A "component set" is a composite component that has its own interface but the functions of the component set is derived by calling methods from other smaller components (see Figure 1).

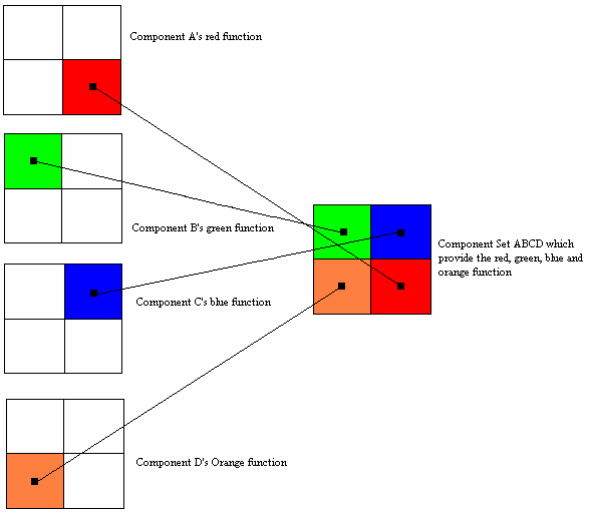

Figure 1 The concept of a component set

As shown in figure 1 above, the component set ABCD on the right provides four functions – colored red, green, blue and orange. The component set ABCD itself does not have the business logic of the four functions, it resides in the supporting components hidden behind the component set ABCD, and these four components are shown on the left.

A component set is used by developers to manage software components that are used within the component-based software system. With the use of component sets developers can manage the software components more easily and efficiently and have a clearer structure of the functions that they need while blocking away unwanted functions in the components but at the same time increasing component reusability.

4.3 Component based software system implementation with TCEM

Currently code generation has become increasingly popular in software development. A lot of software tools and IDEs are able to generate code to lower the implementation work load and reduce development time. We have developed a code generator in the TCET tool to generate both C# and Java code. Most software tools' code generators currently can only generate skeleton code of the software system. The aim of TCEM in its implementation phase is to generate more useful code for the developer instead of just skeleton code. We have setup a database for developers to retrieve or select a wide range of code, including business logic, that can be inserted/deployed into software components. We follow consistent and standardized conventions for all the namespaces of the components, interfaces and classes during the TCEM's implementation phase.

4.4 Component based software system testing with TCEM

A lot of isolated and individual testing is always carried out, as such we see it as more efficient if these testing results can be accessed and reused in TCEM. We can reuse the testing results based on trust models. Every TCEM component is to be accompanied with a document folder which contains complete documentations about that component, including security certificates and digital signatures of the component. Users have an option whether or not to trust the security and testing issues/results concerning any component based on the signature's signer. Testing duplication is not efficient. Therefore if there is a strong component authority who has established a trust model, other developers can save a lot of time and effort on testing by accepting the component and the test results. We have used the Public Key Infrastructure [12] to help us with trust issues.

According to the TCEM methodology each component should have its own certificate. The use of the certificate can help us identify and manage software components and solve some of the security issues of the software components. The component certificate has four elements, i.e. the "General", "Taxonomies", "TestCases" and "Securities" elements, as shown in figure 2.

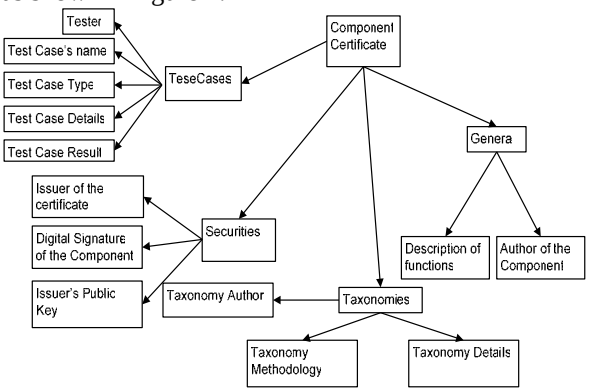

Figure 2 Component certificate's structure

The structure of a Component certificate is shown in Figure 2 above. The "General" element contains the identification of the author of the component and the description of the functions that the particular component provides. The "Taxonomies" element contains the taxonomy methodologies and the detailed descriptions of the component. Users can perform search functions by reading and understanding the component certificates. The "TestCases" element contains information about the person who had performed the tests, the specifications of the test cases as well as the results obtained from the tests. The "Securities" element contains the information of who has checked the particular component, the digital signature of the component and the issuer's public key.

```
- <General>
  - <GeneralInformation>
      <Author>Kelly Yu</Author>
      <ComponentName>LoginForm_Compo
    - <Functions>
      - <Function>
          <Name>showForm</Name>
          <ReturnType>void</ReturnType>
          <Description>This function is for c
        </Function>
      - <Function>
          <Name>setDataSet</Name>
          <ReturnType>void</ReturnType>
          <InputType>DataSet</InputType>
          <Description>This function is for i
            accounts</Description>
```

Figure 3 LoginForm_Component_Certificate's General section

Figure 3 above shows the general part of the LoginForm_Component_Certificate. It gives the name of the author, component name and also the comment/documentation about each function.

Figure 4 below illustrates the Taxonomy part of the LoginForm_Component_Certificate. It shows the classification of the component which is important for performing component selection dynamically.

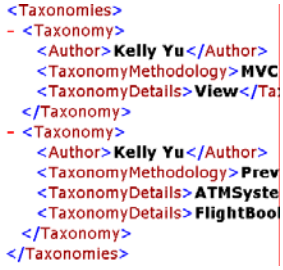

Figure 4 LoginForm_Component_Certificate's Taxonomy section

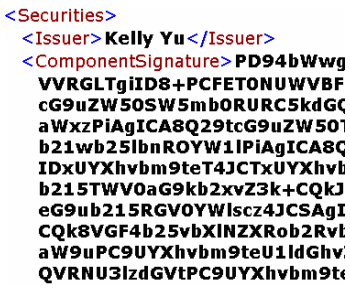

Figure 5 XML snippet from the LoginForm_Component_Certificate's Security section

The security part of a component certificate emphasizes the signer's details as regards the certificate. In figure 5 above, it can be seen that the security part is composed of four elements, i.e. the signature, issuer, issue date, and issuer's public key. The signature generated is based on the software component's code. If anyone tries to modify the code of a software component the signature will become invalid. The component's public key can be obtained from the signer. We have constructed a forum for the use of proof of concept purposes to allow signers to publish their public keys.

4.5 Component based software system maintenance with TCEM

Once the software system is developed, tested to the client's satisfaction and delivered there is a likelihood that it will need to be updated or maintained e.g. for debugging or functional enhancement. During the maintenance phase of TCEM, developers can reuse existing software components from our TCET's local repository and/or web-repository to maintain the software system. Developers are also able to reverse engineer the software back to the CBSS's design diagrams and further get back to the use case diagrams using the TCET tool.

4.6 Total Component delivery (TC- delivery)

It is vital in TCEM to also deliver the software component(s) after been completion. While some CBSD methodologies have not taken delivery of components into account, TCEM considers this to be of real importance. We have created a robust online platform that can be used to deliver the software component to the users who need the component. This assists in the Software component's reusability. However, the Software components security issues and communications between software component's developers and users are also important so that only trust-worthy and secure components are used. In TCEM we proposed the use of Component Certificate to handle the security issues. Complementing the multi taxonomy system in TCEM, this certificate can also assists during the component delivery phase since other developers of the software system can search for software components more accurately and efficiently. The delivery of the software systems comes also with full documentation and information about the software components used and their interrelationships so that subsequent maintenance, refactoring and code reuse will be easier and more efficient.

5. Total Component Engineering Tool (TCET)

The Total Component Engineering Tool (TCET) is a prototype application that is designed and implemented to support the TCEM methodologies requirements and techniques that we discussed in the previous sections. In this section we will discuss the designs of software structures, software components, precise functionalities and the user interfaces that can be visualised and supported by the TCET tool to complement our development methodology. We will also discuss the UML notational extensions that we created.

5.1 UML Extensions to support the TCEM methodology

Our Extended UML included early component use case diagrams, enhanced class diagrams, import-list diagrams, method diagrams and extended component diagrams.

5.1.1 Early Component Use Case diagram. As regards the Use Case diagrams for a software system,

the TCET tool supports identifying the components, called Early Components (or EC), of the software system during this initial phase of development. TCET provides an option to allow users to choose and store useful prefabricated software components or rapidly construct a prototype of a component that have not yet been fully implemented. This function of identifying and using Early Components can aid users to accumulate and consider of the use of software components even during the initial development stages so that they have more control during the rest of the SDLC phases and can rapidly and efficiently develop component based software through greater component reuse. No existing software development tool or methodology can handle Early Components.

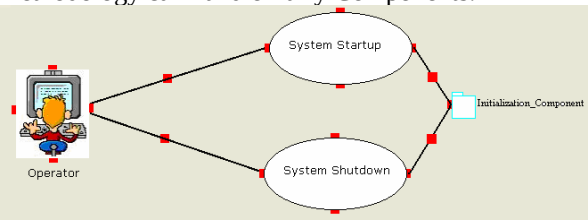

Figure 6 Visually enhanced Use Case diagrams in TCET

An example of a visually enhanced Use Case diagram is shown in Figure 6 above. Here it shows a component (called the initialization_component) that controls a system's startup and shutdown processes. The tool clearly captures the use cases and depicts that the initialization_component provides functions for the two use-cases' requirements. The tool also allows users to have options to choose their own type of visual notation for the actors so that developers can depict the actors in their own style. In this case the operator (actor) is better visualized in an enhanced cartoon format rather than with mere stick figures. More detailed information can be stored in pop-up frames for each and every component, use case or actor.

5.1.2 Enhanced Class Diagram. The class diagram shown below in figure 7 may look similar to the traditional UML class diagram. However there are many useful enhancements, for instance, there is one extra column that contains full information about the packages or other components a particular class uses. This column is called the "Import information panel" and can be used for importing information. The other panels for Class name, Variable panel and Method panel are used exactly as the name implies. Also any detailed instructions or code snippets can be stored in pop-up frames for each enhanced class diagram.

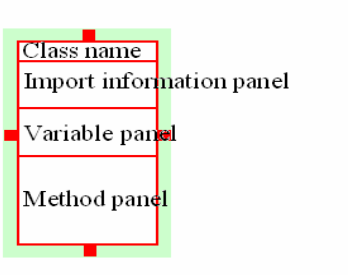

Figure 7 Enhanced class diagram drawn in TCET

5.1.3 Enhanced Method Diagram

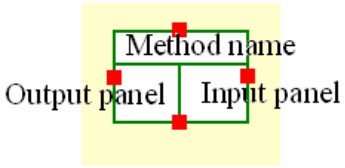

Figure 8 Enhanced Method diagram

The Enhanced Method diagram shown in figure 8 above is not one of the traditional UML diagrams. It also contains all the high level information about how it is to be implemented. The diagram consist of three parts, i.e. the top part is for the method's name, the left hand part is the "Output panel" that shows the types of value(s) that are returned by the method, and the right part is the "Input panel" that store the method's input. All detailed information, instructions or code snippets are again easily stored in pop-up frames for each method.

5.1.4 ImportList diagrams The visual notation of an ImportList diagram is shown in figure 9 below. It is not of a traditional UML type but our own extended version of our modeling language to support TCEM. Whenever there is a new connection with a new class diagram the information in the "Import information panel" from the importList diagram will be copied to the new class diagram's "Import information panel".

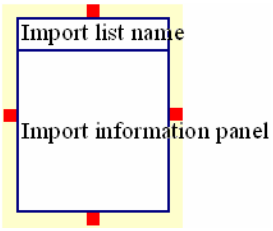

Figure 9 The ImportList diagram

The ImportList diagram has a direct connection to our enhanced class diagram. An importList diagram as such shows a class's or component's import details and relationships with other entities or components.

5.1.5 Component diagram. An example of a Component diagram in TCET is shown in figure 10 below. It consists of the component's name as its topmost tag and there are three very important parts within the diagram. On the left hand side is the Component's Function panel, the middle part is called the Component's Implementation details panel and the right hand side portion is the helping area for assisting with implementation. We can also store all detailed instructions, information, inter-relationship diagrams or code snippets in pop-up frames. This diagram comprehensively captures all the information about a component and its inter-relationships with other components and objects in the software system.

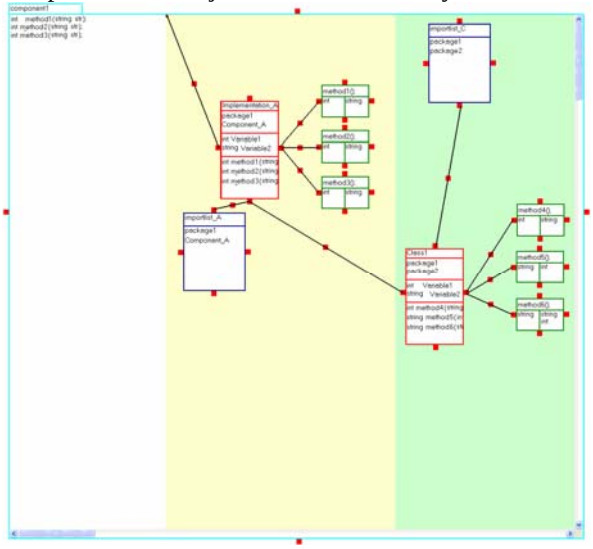

Figure 10 Component diagram in TCET

6. Aspect-Oriented and TCEM

AOSD is an approach that can be used to handle the cross cutting concerns and code interleaving issues involving aspects in software system [13]. We have to be able to address these cross-cutting issues if we are to have very robust systems. To achieve this in our methodology we have shown that we can also incorporate the techniques of AOSD into TCEM. We developed an On-line Banking prototype to illustrate that TCEM is able to integrate with other software development approaches. This is also to show that TCEM is very useful and flexible enough to be used to address other current issues like aspects in software development.

Figure 11 below shows the Total Components (TC) Use Case diagram for an online banking system. It depicts and defines the software components that will be used to satisfy the requirements based on the use cases. As can be seen in Figure 11 (on the right), there are nine Early Components that have been identified.

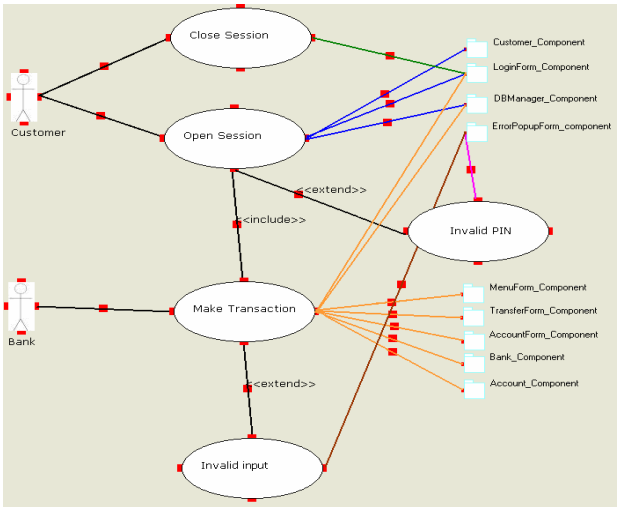

Figure 11 TC-Use case Diagram of an online banking system

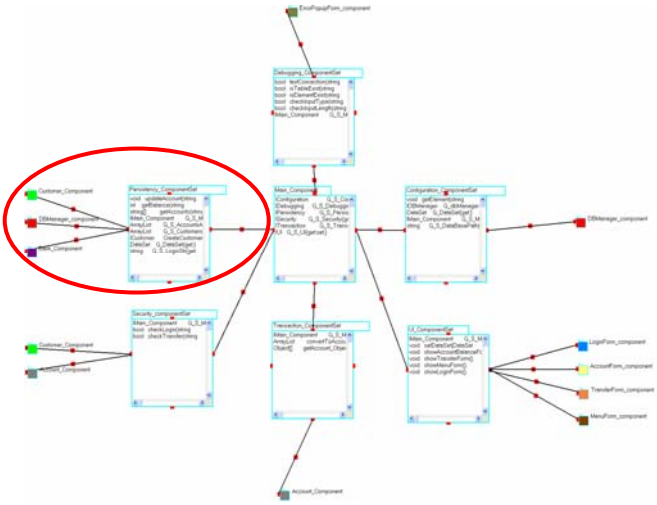

Figure 12 Total Component (TC) design diagram of the online banking system

Figure 12 above shows the TC design diagram that was generated in the TCET tool with the use of the TCEM methodology. The central component is the main component and the surrounding diagrams are component sets. In this example, we incorporated aspect-oriented software development techniques into TCEM. Each component set in this diagram belongs to an aspect type. We have therefore classified the software component's functions into different aspects. The collapsed diagrams (small colored boxes) are the actual software components. The functions in the

component sets are from the backend software components' logic. However, each component set may also introduce more business logic to increase its functionalities.

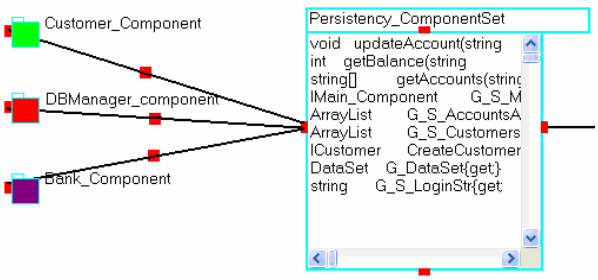

Figure 13 Example of a Component Set and its related components

Figure 13 above shows the Persistency_ComponentSet more clearly (shown circled red in figure 12) and its related components. All methods provided by the Persistency_ComponentSet are called from the components. We have demonstrated the efficient and effective use of TCET and TCEM through designing and developing an online banking prototype example in our research. We also discovered that the structure of the software system is clear, clean and can support plug-and-play features.

7. Evaluation

To show the usefulness and practical application of the TCEM methodology and TCET tool, we carried out an independent evaluation that was approved by the Ethics Committee of The University of Auckland. This evaluation was carried out by eight experienced software developers randomly selected who volunteered to do the evaluation. They were first given an explanation about the methodology and a demo on how to use the TCET tool. They were then asked to use the methodology and tool to develop a sub-system of a reasonably complex system. The rating given by the participants for the usefulness of the TCEM methodology is shown in the Pie Chart in Figure 14 below. (Rating 0 is lowest or least usable and 5 is the highest). 37.5% were of the opinion that is useful (rating level 4) while 62.5% thought that it is extremely useful (rating level 5).

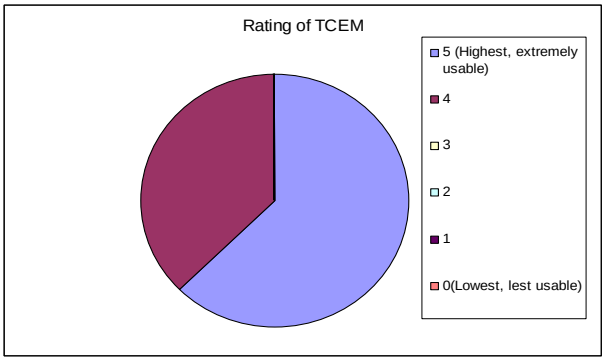

Figure 14 Usability ratings of TCEM

From the evaluation of TCEM, it can be seen that the TCEM methodology is indeed a useful and good way to create component based software systems as most of the users who evaluated it gave very good ratings due to its comprehensiveness and flexibility.

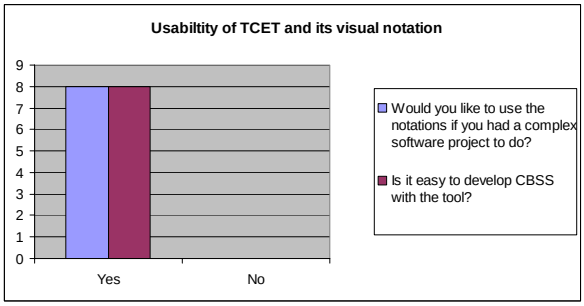

Figure 15 Usability of TCET and its visual notations

Figure 15 shows the results from the human participant usability evaluation on the TCET tool and the extended UML notations. As to the question on how easy it is to develop CBSS with the TCET tool, all the participants agreed that it is easy to use the tool as it is quite user-friendly and built on extending the UML language. The participants also all agreed that they would like to use the notations if they had complex software projects to work on because of the extensive notational support and Early Component identification and reuse.

This human participation evaluation has shown very promising and positive results as regards the use of the TCET tool and TCEM methodology. Based on the evaluations and comparisons with existing technology, we belief that the tool and the methodology is very useful and can be used to efficiently develop better quality Component Based Software Systems more rapidly and with greater control over the development processes involved.

8. Conclusions

We have successfully formulated a new Component based software development methodology called the Total Component engineering Methodology or TCEM. TCEM supports the development of component based software system starting from the early stages using Early Components right through all the development phases making it more efficient, effective and comprehensive when compared to the existing CBSD methodologies.
We have also successfully designed and developed a novel tool called the Total Component Engineering tool or TCET. The TCET tool fully supports and complements TCEM in developing comprehensive software systems and can also address other issues e.g. the problem of cross-cutting issues called aspects in software components and systems. The software developed using TCEM and TCET is more understandable, manageable, reusable and scalable and the development process is also more controllable, efficient, effective and comprehensive. Our next step is to use our research ideas, methodology and tool in other research labs and subsequently apply them in industry to increase productivity and develop efficient componentized software applications and systems.

9. Acknowledgements

We wish to thank the Ethics Committee of the University of Auckland for approving the human participant evaluation of our research ideas and the participants who volunteered to independently carry out the evaluation of our new methodology and tool.

10. References

[1] A. Bertolino and R. Mirandola, "Towards Component-Based Software Performance Engineering," presented at 6th ICSE Workshop on Component-Based 2003

[2] P. Gilda, "Component-Based Software Development: New Challenges and Opportunities," presented at Technology of Object-Oriented Languages and Systems, Santa Barbara, CA, 1998.

[3] S. D. Lee, Y. J. Yang, F. S. Cho, S. D. Kim, and S. Y. Rhew, "COMO: a UML-based component development methodology," presented at APSEC 1999.

[4] D. F. D'Souza and A. C. Wills, *Objects, Components, and Frameworks with UML : The Catalysis Approach*: Addison-Wesley Professional, 1998.

[5] S. Jalili, S. Malakuti, and K. O. Abadi, "AECM: an Aspect Enabled Component Model," presented at APSEC 2005.

[6] M. Matinlassi, "Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA," presented at ICSE, 2004.

[7] N. C. Shu, "A visual programming language designed for automatic programming," presented at Twenty-First Annual Hawaii International Conference, Hawaii, 1988.

[8] R. Sessions, *COM and DCOM: Microsoft's Vision for Distributed Objects* John Wiley & Sons, 1997.

[9] N. Kassem, *Designing Enterprise Application with the Java 2 Platform (Enterprise Edition)*: Addison-Wesley Professional, 2000.

[10] O. M. Group, "CORBA Components," vol. 2006: OMG, 1999.

[11] G. Pour, "Moving toward component-based software development approach," presented at Technology of Object-Oriented Languages, Santa Barbara, CA, 1998.

[12] A. Nash, W. Duane, C. Joseph, and D. Brink, *PKI Implementing and Managing E-Security*: RSA Press, 2001.

[13] J. Grundy, "Aspect-oriented requirements engineering for component-based software systems," In Proceedings of the 4th IEEE International Symposium on Requirements Engineering. IEEE Computer Society Press (1999): 84--91.

[14] T. Wanyama and B. H. Far, "Towards providing decision support for COTS selection," presented at Electrical and Computer Engineering 2005

[15] L. C. Briand, "COTS Evaluation and Selection," presented at International Conference on Software Maintenance, 1998.

[16] G. Pour, M. Griss, and J. Favaro, "Making the Transition to Component-Based Enterprise Software Development: Overcoming the Obstacles - Patterns for Success," presented at Technology of Object-Oriented Languages and Systems TOOLS, 1999.

[17] R.S. Moreira, G.S. Blair, and E. Carrapatoso, “A Reflective Component-Based and Architecture Aware Framework to Manage Architecture Composition,” Proc. of 3rd Int’l Symp. On Distributed Objects and Applications (DOA 2001), 2001, pp. 187- 196.