The Development of a Simplified Process Model for CBSD

Hazleen Aris[1] and Siti Salwah Salim[2]
[1]College of Information Technology, Universiti Tenaga Nasional, Malaysia
[2]Faculty of Computer Science and Information Technology, University of Malaya, Malaysia

Abstract: *This study introduces the MyCL process model, a simplified Component-Based Software Development (CBSD) process model that is clear and easy to be understood and hence, applied. It is motivated by the fact that even though many CBSD process models have been proposed, a clear and step-by-step guidance is still lacking. They vary from one another and some are even complicated and difficult to be applied. The MyCL process model is therefore derived by retaining the strengths and overcoming the impracticality causes of these existing CBSD process models. Evaluation of the model, which was carried out by interviewing the experts in this field has shown that this model has a good potential to be applied by software developers, especially those who are new to CBSD.*

Keywords: *Software methodology, component-based software development, software engineering, process model.*

Received April 6, 2005; accepted June 30, 2006

1. Introduction

As being practiced nowadays, most of the software applications in use are not developed from scratch [1, 14]. Designs and codes from previously developed software applications within the same domain are being reused, unsystematically however, with appropriate modifications done to suit their intended purpose. If systematic reuse of the previous designs and codes is practiced instead, the benefits gained can be greatly increased. Systematic software reuse can be accomplished by considering reuse from the very early stage of the software development process where the related *software units* are grouped together for later reuse [14]. These software units are called *components*, the fundamental ingredient for the Component-Based Software Development (CBSD).

CBSD brings together with it a range of benefits, from enhancing individual programmer's productivity to providing effective cost analysis on the software developed. These benefits of CBSD can be summarized as increased programmers' productivity [4], increased reliability [23], standards compliance [4], improved efficiency [4], effective use of specialists [6, 23] and effective cost analysis [4]. With these benefits, everybody will surely expect it to have taken off with a blast. However, in actuality, it is very much a work in progress and there are still many on going research being carried out on the various areas of CBSD. This non exhaustive list of research areas includes component definition and specifications [24, 25], CBSD process models [1, 2, 5, 7, 9, 13, 26], configuration management in CBSD [15, 16, 27], component repository [11, 22] and CBSD framework [20, 27].

The main objective of CBSD is to reduce the overall cost of a software development [14]. In other words, the software has to be less expensive to produce and maintain. Secondly, CBSD is also required for faster delivery of software product [12, 14]. The software has to meet the market window set by competing organisations. Finally, CBSD also aims at producing high quality software [8, 12]. This means that the software has to serve the requirements of the process that it is going to support and when serving the process, it has to be done with minimum failures.

2. Resistances

Resistances that have delayed the progress towards CBSD emerge from various aspects, ranging from the technical to social problems. They can be generally divided into three major categories; engineering, management and ethics [14]. This study focuses on finding the solutions to the problems in the engineering category. As far as the engineering perspective is concerned, the obstacles come from the deficient opportunities to encourage reuse in the current software development process, the lack of means to clearly identify the elements of the existing CBSD process model, and the differences that exist between these CBSD process models [4, 14].

Firstly, the conventional software development process is deficient in opportunities to encourage reuse in that there is no specific place in the development process where the developers can sit together and think about parts of the system that can be separated out and

substituted with the reusable components [14]. In other words, there is no specific place during the development where the developers can consider reusing existing components. Secondly, the lack of means to clearly identify the elements that constitute a CBSD process model that describe requirements, architecture, analysis, design, test and implementation along the development stream also makes the current models complicated to be used [14]. Furthermore, the differences that exist between these existing models are adding to the complexity even more. Finally, inherent complexities in the existing CBSD process models as well as the differences across them have therefore become a hindrance for the software developers to apply the CBSD.

Obviously, a clear and easy-to-follow CBSD process model is definitely required. This study will try to resolve the above mentioned confusion by coming out with such a process model. To achieve that, the following series of tasks will be performed:

1. Deriving a simplified CBSD process model that is clear and easy to be understood and hence, followed especially by the software developers who are new to CBSD. These developers may have heard of and understand CBSD, but have never developed a software application using the CBSD approach. Hence, the process model to be derived will be geared to suit small-scale system development, the kind of software development that most likely will be ventured by this group of developers. It is important to mention here that the main focus of this study will be on the processes that constitute the model and not on the components construction.
2. Evaluating the proposed process model. This evaluation will indicate to what extent that the model has managed to acquire its intended features. This should therefore be able to indicate whether or not the overall research objectives have been achieved.

The development of the model was also motivated by the fact that even though many CBSD process models have been developed, not many of them are currently being applied by the software developers [1]. Thus, the development of the MyCL process model began by reviewing the strengths and weaknesses of the existing CBSD process models, which was summarised in [3]. A study on the impracticalities of these process models were then performed and also presented in [3], and as a result, the MyCL process model was proposed.

3. Review of the Existing Process Models

The derivation process begins with a review on the existing CBSD process models. The aim of this review is to find out the strengths and weaknesses of each model, which serve as the basis for the development of the proposed CBSD process model. The summary of this comparative study is presented in Table 1 below. The details of this comparative study are presented in [3].

Table 1. Comparison between existing models with respect to the development processes.

Process Descriptions \ Models	Brown and Wallnau [5]	Enterprise Software [2]	Crnkovic [9]	COTS-Based [13]	Cheesman and Daniels [7]	CISD [26]	Twin Track-Based Pattern [1]
Analysing Requirements to Define System Vision			√		√	√	
Determining Component Specifications and Architecture					√		
Acquiring Qualified Components from the Component Market	√	√	√	√	√	√	√
Selecting Qualified Component Creation Processes	√	√	√	√	√	√	√
Understanding Known Bugs to be Avoided in Selected Components				√			
Adapting and Removing Mismatches Between Selected Components	√	√	√	√	√	√	√
Customising Application Design Based on Components		√					
Composing Adapted Components According to an Architectural Style	√	√					
Deploying the Components Using a Specified Framework			√		√		
Testing the Integrated Components		√		√	√		√
Testing the Developed System	√	√	√	√	√	√	√
Updating Components after System Development	√		√				√

As we have seen so far, a number of CBSD process models have been proposed. However, not many of these models are currently being practiced by the software developers. This is further supported by the fact that, throughout our review made on these models and their applications in the industries, there is only one software company encountered as clearly adopting one of these models in their software development process. This software company is the microTOOL GmbH [18], who is using the Cheesman and Daniels model in the software development with slight modifications. Amongst the reasons, at stated by Allen [1] is that most current processes are too overwhelmingly detailed to be applied in practical enterprise.

As such, a comparative analysis is performed on each model to find out the reasons for their low usability. As a result of the analysis, the following reasons have been identified as the possible causes for the low usability:

1. Models such as Aoyama, COTS-based and CISD were originated from the actual software project

developments undertaken by the company involving the researchers who are proposing the models. Hence, the processes and activities that form the process models are closely oriented to those projects and are not general enough to be applied to other software project development.

2. Reuse of the components from the previous development cycle, even though mentioned in all model descriptions, cannot be explicitly seen from the models. With the exception of Twin track-based model, the other models are open-ended (open-loop) and do not incorporate component repository in their models. For the CBSD to be successfully applied, the process of depositing components into the component repository has to be explicitly shown. In other words, there must be a dedicated link from the component updates process to the component repository. Models that have such a link are called the close-ended or close-loop models.
3. Models are too general to the extent that much customisation is required in order to put them into practice. This is especially obvious in Aoyama, Crnkovic and COTS-based models. In these models, the expected deliverables for each process are not described, let alone the activities or steps required in producing them.
4. On the contrary, some models are too detailed and complicated that developers become discouraged to apply. Models that bear this characteristic are Brown and Wallnau, Cheesman and Daniels and Twin track-based pattern models. In Cheesman and Daniels model for example, the diagram specifications to be produced are too detailed as if the application is to be developed from scratch. The aim of CBSD is to keep the components as general as possible [23] and detailed specifications are against this.
5. Strong emphasis is not given to the core development activities, but rather, to other aspects such as staffing and development environment. COTS-based and Twin track-based models exhibit this characteristic. In COTS-based model, the organisation of staff is given more attention while in Twin track-based model, different groups of people with differing interests will trigger the process model from different points. These will further complicate the models. In order to promote CBSD as the preferred approach in software development, the process model should be kept simpler by paying attention to the core development activities, rather than focusing on the non-critical ones.
6. Activities to be performed in each process are not clearly described and examples of implementing the process are not provided [3]. With the exception of Cheesman and Daniels model, all models reviewed in section 3 do not incorporate examples on how the processes in the models are performed. They simply describe what the processes are and the deliverables out of each process without explaining how these deliverables can be produced.
7. Supporting documentations that will guide the software developers in applying the models are not included. This characteristic is true for almost all models. Even if the documentations provided are considerably extensive, they did not explain how the activities in each process can be realised, let alone relating it to the actual implementation tasks. When the realisation of the implementation cannot be seen, the model will fail to catch the attention of the potential developers.

When the problems pertaining to the usability of the models have been listed, the next step is to come out with the possible solutions to each problem. For each problem, the corresponding solutions are suggested, as listed below:

- Reserved place for reuse, where a specific place in the model that will allow the developers to consider reuse in the process of developing the system will be included. This resolves the second impracticality cause.
- Unique process, where for each process included, detailed explanation and necessary examples will be provided that will guide the developers in applying this model. Unimportant processes will be left out to avoid confusion. This resolves the third, fourth and fifth impracticality causes.
- Step-by-step demonstration, where each activity to be performed in each process will be shown to further enhance developers' understanding. This resolves the sixth and seventh impracticality causes.
- Clear inputs and outputs, as the inputs expected for each process and the outputs generated from each process in terms of work products will be stated. This also resolves the third and fourth impracticality causes.
- Closed-Loop (CL) model, where components resulted from the previous development cycle are explicitly fed back to the model to populate the repository. Due to this closed-loop feature, the proposed model will be called the MyCL process model. This resolves the second impracticality cause as well as to emphasise the reuse of components produced from the previous development lifecycle, which is the main objective of CBSD.

Therefore, in this study, a simplified CBSD process model, which is clear and easy-to-follow, is proposed. This model is derived mostly from the existing CBSD process models studied before by retaining their strengths and improving their weaknesses. The process model which incorporates the features that will solve the low usability problems of the existing process models is what we mean by a process model that is clear and easy-to-follow, as stated at the very beginning of this article and referred to at several

places throughout. These features will be the guidelines for the development of the MyCL process model that will be explained in the next section.

4. MyCL Construction

The first step in deriving the model is to determine the processes to be included in the model. It involves two types of grouping done on each process in the existing models studied as described below:

- Firstly, processes in the existing models are grouped according to the fundamental software development phases. For this purpose, the five fundamental phases of software development are used as shown in Table 2 [19, 21, 23].
- Secondly, these processes, which have been grouped according to their respective phases, are further refined according to their descriptions. The differences and similarities are determined before a distinct set of processes can be identified. Processes that describe similar set of activities will be substituted with a name that reflects the activities that it supports. This summary will serve as the basis for determining the processes to be included in the MyCL process model.

The justifications on the groupings will be elaborated in subsection 4.1. The second step is to determine the deliverables from each process, which will act as the input to the process that comes next. This step will be discussed in subsection 4.2.

4.1. Process

As can be seen from Table 2, what is being done in the *requirements analysis* process for CBSD is the same as in conventional software development. Therefore, this process is retained. Next, it can be concluded based on their descriptions that processes which fall under *design* phase are actually part of the domain engineering process [10, 17, 23], which is a research topic on its own and will not be discussed in detail here. In the MyCL model, these processes are placed under the domain engineering process. A wide range of tasks from searching for appropriate components to composing the selected components to build a working system falls under the *implementation* phase.

Searching for components and adapting them for integration with other components are placed together under component development process. The process of composing the adapted components is the focus of the component composition process. Therefore, in the MyCL model, the implementation phase is replaced by the component development and component composition processes. *Testing* phase mainly covers integration testing and system testing, removing unit testing from the development lifecycle. This removal is obvious, as the system is no longer built from scratch, but from composed components. Component-based system testing is a broad research topic and will not be discussed in detail here. The process will be included in the model to indicate that testing is required before the application software is delivered to the customer.

Finally, the processes that fall under the *maintenance* phase are substituted by the component updates process. It concerns with fixing errors and adding new functionalities, replacing the old version of a component with an improved version [5, 9]. The tasks involved are the same as the maintenance tasks performed in conventional software development, just that they are made simpler as the component can be plugged in and out to accommodate changes. In addition to these processes, architectural design process, which is missing in almost all models, is included. It is placed right after the requirements analysis process and concerns with producing component specification architecture that enables component selection.

4.2. Deliverables

Determining the deliverables from each process is a more complicated task than determining the processes themselves due to their diversity across existing models. The main aim is that, the developer should not be overwhelmed with the production of documentations unimportant to the development process. To begin with, Cheesman and Daniels model [7] is closely followed as this model provides the most complete listing of deliverables from each process. Then, unneeded documentations are removed and new ones that are tailored to the component framework applied are added. One distinguishing feature of the MyCL model is that updated components are fed back to populate the component repository. It is indicated by an arrow connecting the component updates process to the repository. The whole of the MyCL process model, including the deliverables attached to each process is shown in Figure 1.

As can be seen below, the MyCL process model is triggered when requirements definition is received from the user. These requirements are then analysed using any existing requirement elicitation technique before an architectural design is established. Then, a group of domain engineers will perform a series of domain engineering activities based on the preliminary requirements analysis result. When the architectural design is established, each component will be implemented according to their specifications, which includes the development of the interfaces offered by each component. Next, these components will be individually tested prior to their composition using a selected framework to produce a working system. A series of tasks to test the system produced will follow before it is delivered to the user as application software. Finally, the repository will be updated with

the new components, which may include the new version of the reused components that have been improved or fixed. The whole process will be repeated when a change request or new requirements definition is received.

Table 2. Description of each process and its grouping.

Phase	Process	Description
Requirement Analysis	System Requirement, Requirements, Analysis	• Customer and developer agree on what the system should do • Understand system requirements and partition the requirements into various applications and domains
Design	Product Identification, Information Gathering	• Collect information on candidate COTS components and group them • Gather information on components from web or other similar projects
Implementation	Qualification, Find, Specification	• Search for appropriate components • Determine components to build and buy • Understand component descriptions, attributes, aspects of their performance, reliability, usability and so on
	Component Acquisition, Provisioning	• Acquire the appropriate components from COTS component market • Build and buy identified components
	COTS Understanding, Product Identification	• Review all candidate COTS to generate prioritised list for further evaluation • Understanding chosen COTS components in detail
	Business Process Improvement	• Used for software project reassessment based on previous experience
	Adaptation, Compositional Design, Select, COTS Evaluation, COTS Understanding, Product Evaluation, Solution Assembly, Component Provisioning	• Make the components work together by means of wrapping • Tailor and customise components • Select components that meet the requirements • Create prototype software for temporary integration and testing of candidate COTS components • Compare and identify optimum set of collaborative COTS components for the final integrated system • Searching for available components and where necessary, raising requirements for new components from the provisioning track
	Architecture Planning	• Used for component reassessment in a process of progressive refinement
	Composition, Component Integration, Adapt, Deploy, Integration, Deployment, Product Integration/ Enhancement, Integration, Assembly	• Compose components using visual composition environment • Integrate assembled components through some well defined infrastructure • Integrate with existing systems • Compose and deploy components using a framework for components • Integrate/interconnect different selected COTS products into a single integrated system • Integrate components together with existing assets and suitable user interface • Produce, package and distribute complete software
Testing	Integration Test	• Ensure that the composed components are working as expected • Inspect composed product for any overlooked bugs • Verify the proper integration of all components of the software
Maintenance	Evolution, Replace, Debug	• Fix errors or add new functionality in components • Replace earlier version of a component with a new one

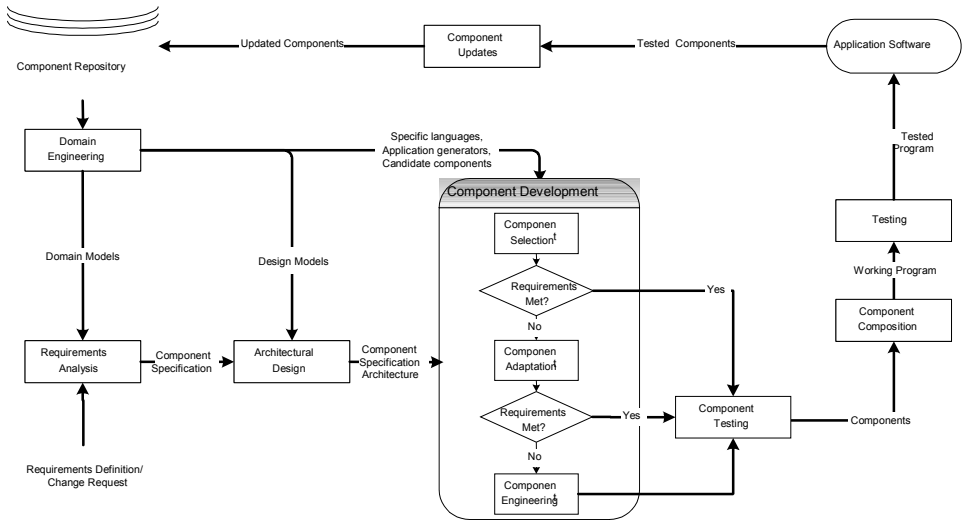

Figure 1. The MyCL process model.

5. Evaluation

Evaluating the MyCL process model is not a straightforward process. The best way to evaluate the feasibility of the model with respect to its desired features is to gather a group of people with a good understanding on the CBSD and ask them to develop an application system using the model. However, this approach is impractical due to the following constraints:

- Looking for suitable candidates to evaluate the model is difficult. This is due to the fact that the CBSD is still an ongoing research and its development and progress are currently constrained to a very limited group of people within this field.
- Even if suitable candidates can be found, asking them to apply the model in developing an application system is even more difficult. Not many of these candidates would agree to do this as they are also occupied with their own work.
- Furthermore, to understand the model before applying it will require the candidates to read the whole of this dissertation, which is not practical due to the time constraint.

In addition to this, assessing on the acceptability of the MyCL process model may require the model to be applied and tested in the actual software development environment industries, which is time consuming and beyond the scope of this study.

Unidirectional evaluation approach such as questionnaires distribution on the other hand, will not be suitable for this kind of evaluation. This is because, in the process of understanding the model, questions will need to be asked and examples need to be shown to provide proper understanding on the model. Even though supplementary information on the model can be placed on the web pages, it is found that these web pages are hardly referred to. Therefore, it has been decided that the most suitable evaluation approach is through interview and discussion sessions with the experts in this field. Five interview sessions have been conducted. The interviewees are chosen amongst those who have adequately high level of knowledge on CBSD.

As expected, all of the interviewees agree that the best way to evaluate this model is to apply it, but at the same time acknowledge the impracticality of implementing such evaluation. Therefore, they have agreed to provide feedbacks on the model based on the explanation given and the answers provided during the interview session. These comments are stated below followed by the action taken to avoid the possible problems addressed by them.

1. Notes need to be attached to the inputs and outputs of each process in the MyCL process model for better description.
2. Misleading step with respect to the link from domain engineering process to architectural design.
3. Domain engineering process shown is not very descriptive.
4. Explanation on how the link between requirements analysis and design and implementation is established for newly created components needs to be given.
5. Need to mention that the focus is on the processes and not on the components, to avoid confusion.

The first comment made on the model is referring to the inputs to and outputs from each model. It suggests that notes should be added to better describe each input and output. Since the description on how to produce each input and output will be too lengthy to be incorporated in the model, it has been decided at first that the purpose of each input and output will be attached as notes instead. Unfortunately, the addition of notes attached to each input and output cannot be made because they will clog up the model's figure and make it more complex. Complexity in the model is against the objective of this study.

The second and third responses comment on the domain engineering process that is part of the model. They state that the process is not descriptive enough and that the link from this process to the architectural design process is misleading. However, domain engineering is a very broad research discipline on its own and to extensively cover this topic is beyond the scope of this study. The fourth comment points out that the explanation on how the link between requirements analysis and design is established for newly created components is not given. Again, to engineer a new component will require a more detailed study on component characteristics and model, which is beyond the coverage as this study focuses only on the processes that constitutes the process model and not the components. This is also applicable in answering the last comment made.

6. Conclusion

In this study, a review has been made on a number of existing CBSD process models which are not really being applied. Reasons for them not being applied, together with their strengths and weaknesses are derived out of the review and used as a basis to come out with the MyCL process model that is clear and easy-to-follow. A group of experts have been interviewed to evaluate and give feedbacks on the simplicity and feasibility of the MyCL process model. This simplified process model can encourage software developers, especially those who are new to the CBSD to apply the CBSD in developing software. Research effort of this kind will be a significant contribution to fostering the transition towards software development based on components.

References

[1] Allen P., "Ebiz Components," *Objective View*, no. 6, pp. 12-20, 2003.

[2] Aoyama M., "Process and Economic Model of Component-Based Software Development: A Study from Software CALS Next Generation Software Engineering Program," *in Proceedings of the 5th International Symposium on Assessment of Software Tools and Technologies*, Pittsburgh, USA, pp. 100-113, June 1997.

[3] Aris H. and Salim S. S., "Component-Based Software Development (CBSD) for Web-Based Applications," *Technical Report*, University of Malaya, 2004.

[4] Brian W. H. B., "CBD: Is There a Point?," Surprise 2001: Component Based Development, Article 2, available at: http://infoeng.ee.ic.ac.uk/~malikz/surprise2001/hbw99e/article2/, Imperial College, London.

[5] Brown A. W. and Wallnau K. C., "Engineering of Component-Based Systems," *in Proceedings of 2nd IEEE International Conference on Engineering of Complex Computer Systems*, Canada, pp. 414-422, October 1996.

[6] Cann S., Rossi A., and Pilgrim P., "Frameworks for Building Component Based Applications," available at: http://www.jcorporate.com/econtent/content.do?state=template&template=2&resource=636&db=default, April 2003.

[7] Cheesman J. and Daniels J., *UML Components: A Simple Process for Specifying Component-Based Software*, Addison-Wesley, 2001.

[8] Cox P. T. and Song B., "A Formal Model for Component-Based Software," *in Proceedings of IEEE Symposia on Human-Centric Computing Languages and Environments*, Italy, pp. 304-311, 2001.

[9] Crnkovic I., *Component-Based Software Engineering: New Challenges in Software Development*, *Software Focus*, John Wiley and Sons, December 2001.

[10] Curfman B., Lewis S., Reddy J., Wallnau K., and Martin L., "Informal Technical Report for the Software Technology for Adaptable, Reliable Systems (STARS)," *STARS Informal Technical Report STARS-VC-B005/001/00*, Unisys Corporation, October 1993.

[11] Guo J. and Luqi, "A Survey of Software Reuse Repositories," *in Proceedings of the 7th IEEE International Conference and Workshop on the Engineering of Component Based Systems*, Scotland, UK, pp. 92-100, 2000.

[12] Haines G., Carney D., and Foreman J., "Component-Based Software Development/COTS Integration," *Software Technology Review*, available at: http://www.sei.cmu.edu/str/descriptions/cbsd_body.html, January 2003.

[13] Hirai C. and Nobuo S., "A Proposal of an Internet-Based Software Development Process Model for COTS-Based Systems Development," available at: http://sern.ucalgary.ca/~maurer/ICSE98WS/Submissions/Hirai/Hirai.html, April 2003.

[14] Jacobson I., Griss M., and Jonsson P., *Software Reuse Architecture, Process and Organization for Business Success*, Addison-Wesley, 1997.

[15] Larsson M. and Crnkovich I., "Component Configuration Management," *in Proceedings of the Workshop on Component Oriented Programming in ECOOP Conference*, France, June 2000.

[16] Larsson M. and Crnkovic I., "New Challenges for Configuration Management," *in Proceedings of the System Configuration Management SCM-9*, Toulouse, August 1999.

[17] Nilson R., Kogut P., and Jackelen G., "Component Provider's and Tool Developer's Handbook Central Archive for Reusable Defense Software (CARDS)," *STARS Informal Technical Report STARS-VC-B017/001/00*, Unisys Corporation, pp. 27-33, March 1994.

[18] ObjectiF, MicroTOOL GmbH, "Mastering the E-Business Challenge: A Process for Component-Based Development with ObjectiF® and the UML," ObjectiF® Special, available at: http://www.microtool.de/objectif/en/sp_cbd.html, April 2003.

[19] Pfleeger S. L., *Software Engineering Theory and Practice*, Prentice Hall, Inc., 2001.

[20] Praehofer H., Sametinger J., and Stritzinger A., "Component Frameworks: A Case Study," *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS'1999*), USA, pp. 148-157, August 1999.

[21] Pressman R. S., *Software Engineering a Practitioner's Approach*, McGraw-Hill International Edition, 2001.

[22] Seacord R. C., "Software Engineering Component Repositories," *in Proceedings of the International Workshop on Component-Based Software Engineering*, USA, 1999.

[23] Sommerville I., *Software Engineering*, Addison-Wesley, 2001.

[24] Szyperski C., *Component Software: Beyond Object-oriented Programming*, Addison-Wesley, 1998.

[25] Teschke T. and Ritter J., *Towards a Foundation of Component-Oriented Software Reference Models*, Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, pp. 70-84, 2001.

[26] Tran V., "Component-Based Integration Systems Development: A Model for the Emerging Procurement-Centric Approach to Software Development," *in Proceedings of the 22nd Annual International Computer Software and*

Application Conference, Austria, pp. 128-135, 1998.

[27] Wallnau K., Bachman F., Bass L., Buhman C., Comella-Dorda S., Long F. Robert J., and Seacord R., "Component Models and Frameworks," Technical Concepts of Component-Based Software Engineering, SEI Technical Report CMU/SEI-2000-TR-008, vol. 2, 2000.

Hazleen Aris obtained her Master degree in software engineering from the University of Malaya, Malaysia and Bachelor of computer engineering (Hons) from the University of Southampton, UK. Currently, she is a lecturer at the Department of Computer Science, College of Information Technology, Universiti Tenaga Nasional. Her research interests include the development of Component-Oriented Programming (COP) language and its compiler, and components composition at a higher level of abstraction.

Siti Salwah Salim obtained her PhD in computer science from the University of Manchester Institute of Science and Technology (UMIST), United Kingdom. Currently, she is an associate professor at the Faculty of Computer Science and Information Technology, University of Malaya. Her research interests include computer supported collaborative learning, human computer interaction, software requirements engineering, and animated pedagogical agents.