INTRODUCING INTEGRATED COMPONENT-BASED DEVELOPMENT (ICBD) LIFECYCLE AND MODEL

Amr Rekaby[1] and Ayat Osama[1]

[1]Egyptian Research and Scientific Innovation Lab (ERSIL), Cairo, Egypt
Rekaby@ersil.org
Ayat.mohamed@ersil.org

ABSTRACT

Component-based development methodology is one of the recent research windows in software engineering field. It investigates in how to build a reusable component to be used later in another application/context/domain. It targets the increase of software quality and the decrease of the production cost. Component-based development has many challenges to be applied in real world such as the impact on the running project schedule and budget.

In this paper, a survey of component-based development and reuse driven development life cycles is presented. We present our proposed model "Integrated Component-Based development" (ICBD) life cycle. ICBD contains all the needed activities towards a complete component-based development lifecycle. Comparison between ICBD, normal component-based development, and non-component based development is provided in this research. The presented case study proves that ICBD could decrease the effort of the projects by 40% after few months of applying.

KEYWORDS

Integrated Component-based Development (ICBD), Component-based Development, Component-based Development Lifecycle, Reusable Driven Development.

1. INTRODUCTION

Component-based development methodology is a way of software development that targets building reusable components to be probably ready for further reuse in other projects/products [1]. This reusability decreases the cost of the following projects and increases the quality and the stability of the product [11]. The reusability in software development is not a new term. It is existing starting from structure programming languages through developing a function or method and reusing its logic in many project areas by function calling [10]. The importance of the reusable component development is extremely demonstrated in the market survey [2]. This survey mentions that Toshiba reduced the defects by 20-30% per line of code when they have reusability by 60%. NASA reports said that reusability reduced the overall development effort and cost by 75%.

1.1. Software Reusability Levels

There are many levels of reusability in software world, samples of these levels are [2]:

- Code-level component reusability

  Source code constructs a component (fine-grained unit). This component is reusability-enabled to be injected in another context. The new projects may be within the same business domain or they may be in a different business domain. Technology disciplines are a restricting constraint on such reusability level, for example: Java component will not be reused in .Net project even if they are in the same domain and fulfil all the requirements' needs. Code-level components are always targeting technical functional reusability, not a business functional providing. Examples of technical functionality are: logging, XML manipulation, XSLT conversion, java objects mapping, etc. Business functionality examples are: student registration, employees' pay roll management. Reusability of object oriented classes affects the needed testing efforts positively [12].

- Functional module reusability

  Functional module level is the wider scope of code-level reusability. The module is a coarse-grained component that contains many code-level components. On this level, the module could be business functional provider, technical functional provider or both.

- Design pattern reusability

  Design pattern is a commonly known way of reusability. Design pattern is a design (and might include implementation) of already experienced solutions for common known problems. These solutions are reused in any context which faces the same problem.

- Library reusability

  Library is an encapsulated version of functional module reusability. It is a black box which provides concrete services. Library is also technical dependant. It could provide a technical/business services.

- Service oriented reusability

  From service oriented architecture (SOA) point of view, the system is built based on a set of independent services. These services are exposed to be used according to a consuming schema. Who have the privilege, and use the schema properly, could consume these services.

- Framework reusability

  Framework is a huge reusable unit that contains services and could manage his work through internal lifecycle. Frameworks always have a lot of configuration and customization towards more users' satisfaction acquiring. Examples of java web development frameworks are JSF, hibernate, spring, etc.

- Configurable application reusability

  Products in specific domain (clinical management for example) are high configurable applications. These products are highly customized enabled to be able to be adapted with customer needs to achieve the satisfaction. A big constraint in such level is the business domain. Almost all these products are very coupled with specific business domain and family of specific requirements.

- Commercial-O-The-Shelf reusability

  Commercial on the shelf integration is the highest reusable components level. It integrates already developed and running commercials from the shelf to get more complicated functionality. Huge organizations like HP, IBM perform these integrations between their products towards more business deals, sales and revenue generation.

This paper focuses on a fine-grained and medium-grained reusability. It focuses on levels from code-level to SOA reusability.

1.2. Component-based Development Model Introduction

The cost reduction comes from deleting part of the development effort in the project; uses already implemented stuff, and just integrate it into the new project code. Already done components are theoretically tested before, so it would increase the product quality. Detailed life cycle of the reusable component development is discussed in section 2. “White box” and “Black box” components are proposed in section 3 as part of the proposed integrated component-based development life cycle (ICBD). Initially, there are two roles in component-based development lifecycle [4]:

- Component producer:

  This is the team that creates the reusable components and tests it to be ready for future needs. This could be done in many ways like:

  - A separate team activity. This team searches for components that are needed, then design and develop them in advance.
  - As post project activities, after the project delivery, investigators work on the project’s artifacts to extract components which could be reused later in other projects.

- Component consumer:

  This is the new project teams who get the components and integrate them into their new source code. The reusable component always stored in specific repository for components. This repository is the source for consumer searching.

ICBD model proposed in this paper (presented in section3) will present a new vision in these roles.

Budget and cost is the main driver in the software projects. The projects decide either they consume already existing components or develop their new own according to cost calculations. More details of the cost calculations are presented in next sections.

The structure of the paper is as the following: section 2 presents a survey on the component based development life cycle and guidelines, section 3 contains the proposed “Integrated component based development model”, and evaluation and assessment are presented in section 4. At the end of the paper, the conclusion is provided.

2. COMPONENT-BASE DEVELOPMENT

The Component-Base development (CBD) is based on the integration of already developed reusable components that are picked from the organization reusable components repository [5]. CBD is divided into two parts. The first one is for developing/producing the reusable component, the other part is for using/consuming the reusable component. Most of the researches handle these two parts of lifecycle separately. In the next subsections, description of the component development and consuming lifecycles are presented. Afterwards guidelines of the component-based development overall will be mentioned [4].

2.1. CBD Lifecycle phases

This section is describing the Component-based development lifecycle [5] and a compare between it and the non-component based development lifecycle.

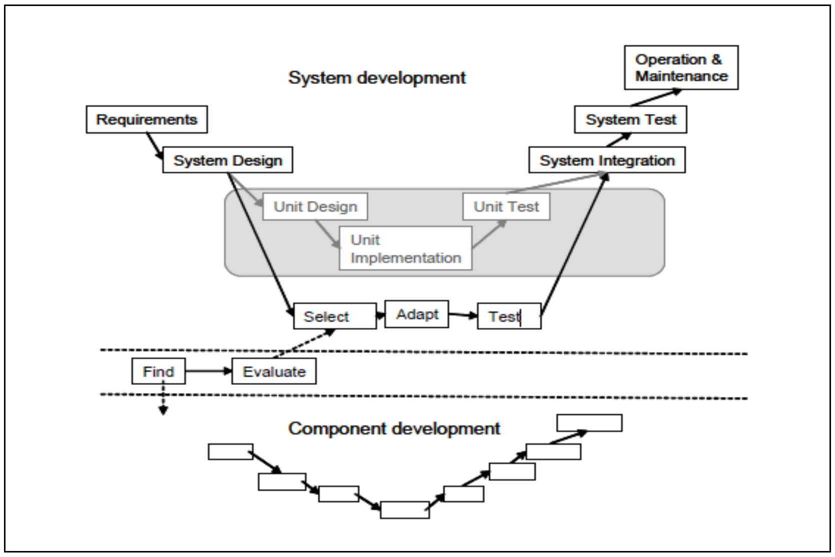


Figure 1. V development process for CBD [5]

As presented in the figure 1, the component production phases are shown in "component development" section. The producing life cycle is the same as any normal software development lifecycle (analysis, design......etc). The component development lifecycle is completely separated from the component-based development projects lifecycle (projects that consume the reusable components).

The component development lifecycle is an organizational scope activity, where the organization dedicates a development team to create these components.

The "system development" section as shown in figure 1, is related to the components consuming. It shows the difference between the developments lifecycle for component based project and non-component based project.

As commonly known, in the requirements phase the system/project requirements is collected.
In the system design phase the whole system design is proposed based on the requirements collected in the previous step. Here a question may be raised regarding whether the design should

be twinkled based on the components in the repository or should the designer ignore this point and subsequently design without looking at the repository.

The grey block in figure 1 shows the lifecycle phases related to non-component based lifecycle. These phases are the difference between component based life cycle and non-component based lifecycle. Here in the grey block, there will be unit design phase of the systems' components. Then unit implementation based on the unit design in the previous step. Afterwards, there will be a unit test for the implemented software pieces.

Figure 1 also shows the phases of select/adapt/test. These three phases are different from the non-component based development lifecycle phases.

In the select phase, the developer will look in the repository to find the component that will mostly match the system design. As in figure 1, the developer will look in the repository for all possible components to match the system design. Then the developer will evaluate the retrieved components to decide which components will be used. Many researches focus on the evaluation and component validation techniques [3].

In the "adapt" phase the developer will have to modify in the component if it doesn't completely fit the design. However if the component matches the design perfectly then this phase will be skipped along with the test phase as the component will be ready for system integration.

The test phase will take place after the component modification in the previous phase. The test will be unit testing for the modified functionalities of the component.

In the system integration phase the different components implemented or reused from the previous phase will be integrated together. Integration test will take place in this phase too.
In the system test phase the whole functionality of the system will be tested against the system requirement and design.

By the final phase operation & maintenance, the system will go live in production and maintenance on it will take place whenever needed.

2.2. Component-based Development Guidelines

In this section we will talk about the guidelines for component development and the selection criteria for the components from the repository [4]. There are some guidelines to help the producer while designing and developing the components, these guidelines will help him focusing on the abstraction path and verifying the components against the objective of reusability. There are many categories for the guidelines some of them are [7]:

- Language-oriented reuse guidelines

  Reusability is supported in most programming languages and specially the Object Oriented programming language. It's up to the developer to use these functionalities provided by the programming language to generate a reusable component. But the developer should be aware of the language capabilities which could help in the reusability development (packaging, encapsulation, information hiding.....etc).

- Domain-oriented reuse guidelines

  The reuse component development will be based on a given application domain. Each business domain has a common logic which makes the generalization inside this domain much easier than doing it within different domains.

- General high-level guidelines [8]

  - Conducting software reuse assessment.
  - Performing cost-benefit analysis for reuse.
  - Adoption of standards for components.
  - Selecting pilot project for wide deployment of reuse.
  - Identify reuse metrics.

On the other hand, selecting the components from the repository is not easy. Many techniques are used in classifying the components in the repository [9]. The developer has to select the most suitable component to fit with the new project's requirements. The selection decision will be very hard when there is a poor documentation of the components. The adaptation of the component might be too complex, that it will be easier to implement the component from scratch.

The decision of implementing the component from scratch or adapt an existing component depends on many factors. The most important factor is the cost, whether implementing from scratch cost more than adapting an existing component or not.

The cost difference can be calculated by the given formula [8]:
$\mathbf{C_{save} = C_s - C_r - C_d}$
Where C_s represents the cost for implementing the component from scratch, C_r represents cost of adopting a reusable component, and C_d represents the actual cost of delivering the component.

3. INTEGRATED COMPONENT-BASED DEVELOPMENT MODEL

In this section the proposed "integrated component-based development" (ICBD) model and lifecycle are presented. First, ICBD life cycle details are discussed. Then proposed guidelines and templates are provided.

3.1. ICBD Life Cycle

ICBD main objective is the integration of the two separated activities: reusable component producing and reusable component consuming. While other researchers studied either producing or consuming, ICBD model utilizes the daily projects activities in producing the reusable components. ICBD is constructed from normal project activities/life cycle phases plus new phases and responsibilities of already existing ones. As presented in figure 2, blue blocks present normal non-component development phases. Green blocks are the component producing proposed phases, while red are the proposed component consuming phases. The gray repository represents reusable component storage repository.

As presented in ICBD lifecycle, the project activities start normally according to iterative lifecycle (or Agile). After putting the high level design, the team goes into component searching activity, the team looks up the needs from the reusable component repository. Either the team could find a component and consume it (red consuming part would be described later), or the team would implement the requirements from scratch.

All the implementations that are done in the project go into component producing part (green blocks). Details of each block are presented later. ICBD is always targeting enriching the reusable component repository, so the repository is the centre point of the life cycle that the producer ends with and the consumer starts from.

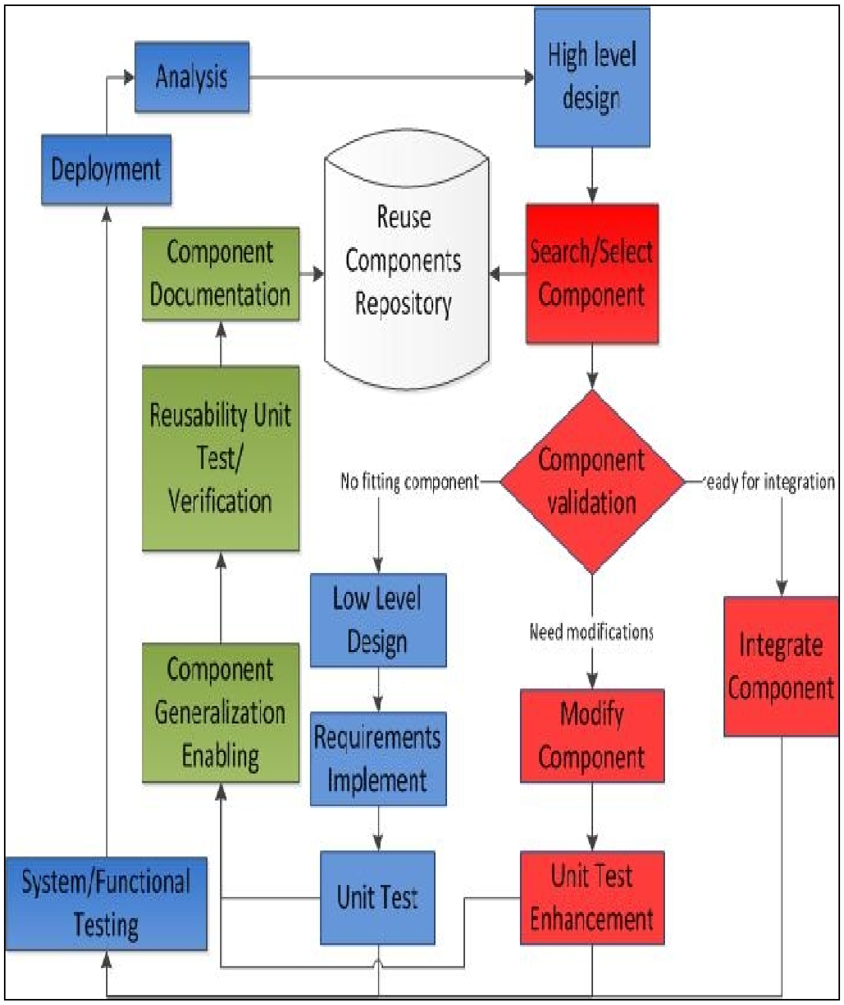


Figure 2. Proposed ICBD lifecycle

As presented earlier, the component development is either separate organizational activity or post project activity. In ICBD, the component producing is run through on the normal project activities. The normal project daily effort is used towards enriching the reusable components repository for the future needs.

The blue blocks are not discussed in details here due to their popularity.

The component consuming blocks (red) are:

- Search/Select Component:

  In this activity, a senior experienced technical person searches in the repository for components that fit the project needs and the high level design disciplines. Searching in the repository with a huge number of available components is very tough task. ICBD proposes documentation meta-data information of each reusable component in the repository. This documentation should be done as a part of component producing. The selection phase uses an automation search according to this documentation factors. The documentation template "ICBD component meta-data template" is presented in details in the next section.

- Component Validation Check:

- After retrieving the search results there are three scenarios applicable here:

  - No component available with needed requirements: the project life cycle continue as if it is normal non-component development project. The project starts in low level component design. ICBD requests that, the low level designers always do their best in making the design as much general as possible. This is only ICBD recommendation that should run without actual doing any project budget/plan violation. If the low level design, implementation and unit test is general, it would be perfect. If they are not as general as needed, it would be handled later in component producing phases as described later in this section.
  - Fetched component is 100% matching the requirements (black box component): the component is integrated directly into the project as described later.
  - Fetched component is NOT 100% matching the requirements (white box component): the component is adapted then integrated into the project as described later.

- Integrate Component:

  If the component is entirely matching the needs, it is considered as a black box component. The component is integrated with no extra unit test effort (the component is already tested during the component production phases). Meanwhile, functional testing will run after the integration to validate the functionality and the integration efficiency.

- Modify Component:

  If the component is not completely matching the needs, it is handled as a white box component. The component needs some enhancements or changes to fulfil the project needs. It is mandatory before the integration within the project.

  This enhanced version of the component would be a candidate for new reusable component or new version of already existing component.

- Unit Test Enhancement:

  Extra unit test effort is needed according to the component changes. Meanwhile, functional testing will run after the integration to validate the functionality and the integration efficiency.

Now we will discuss the component producing blocks (green):

- Component Generalization Enabling:

  Coming from scratch component development or from modified reusable component activity, the generalization enabling is the next phase. In ICBD, a team should be structured to be responsible for the components producing. This team is cross organization team, that contains a member of each project. In this phase, the design and implementation of the component is revisited to be very generic and configurable to be applicable for reuse in other projects. What we mentioned above is that, if the design and implementation exert an effort in doing the work in a proper reuse way, it would minimize the needed effort here. Absolutely, may be the team of the generalization decide that, this stuff is very coupled to a specific project and no use in enabling it for reuse. If so, the lifecycle of the producing is stopped here for this component.

  The level of the component size vary from very fine-grained component till functional module or complete service, so the component X could be generalized and after that it is generalized again beside component Y under a bigger component Z. This point is very tricky, when a component should be generalized and when not. The main rule of this point is “will this encapsulated item needed as it is in the future?”. If the answer is yes even if the subcomponents of it are already available but there is a need for the subcomponent and also for the complete bigger one.

- Reusability Unit Test / Verification:

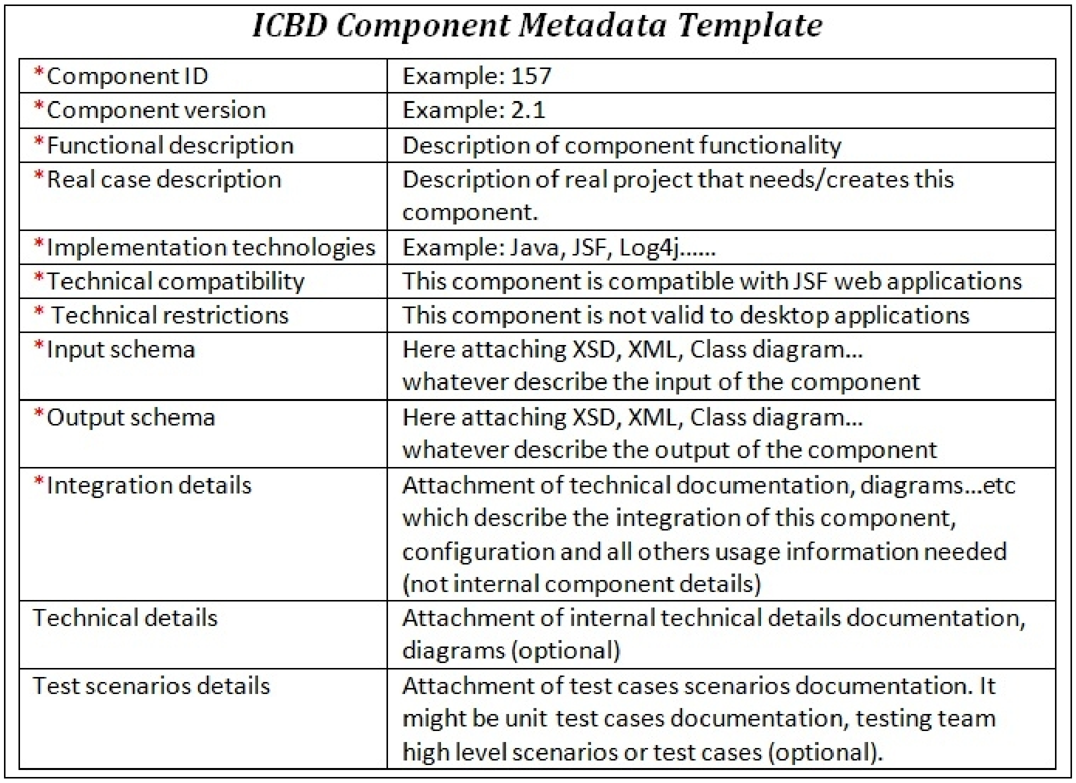
  Here a unit test for the component is done. In ICBD, unit testing is a mandated deliverable part in each reusable component. It would be enhanced in further projects if the component is selected and modified. The designer of the unit test should only care about the reusability purpose.

- Component Documentation:

  Documenting the component is mandatory in ICBD. The proposed template is described in the next section.

3.2. ICBD Documentation Templates

In this section the proposed component documentation template is provided. As presented in figure 3, each component in the repository should be tagged with these meta-data. The automated search for the component uses these meta-data fields as search criteria.



ICBD Component Metadata Template

*Component ID	Example: 157
*Component version	Example: 2.1
*Functional description	Description of component functionality
*Real case description	Description of real project that needs/creates this component.
*Implementation technologies	Example: Java, JSF, Log4j......
*Technical compatibility	This component is compatible with JSF web applications
* Technical restrictions	This component is not valid to desktop applications
*Input schema	Here attaching XSD, XML, Class diagram... whatever describe the input of the component
*Output schema	Here attaching XSD, XML, Class diagram... whatever describe the output of the component
*Integration details	Attachment of technical documentation, diagrams...etc which describe the integration of this component, configuration and all others usage information needed (not internal component details)
Technical details	Attachment of internal technical details documentation, diagrams (optional)
Test scenarios details	Attachment of test cases scenarios documentation. It might be unit test cases documentation, testing team high level scenarios or test cases (optional).

Figure 3. Proposed ICBD reusable component documentation template

As presented in figure 3, each component has unique identification. Version of each component is very critical during continues component modifications and improvements.

Functional description is very important for the consumer. Also ICBD requires a short description of the real project that creates this component to give the feel for the consumer where that component was needed to fulfil the needs.

Technical implementation details are listed in this template, also any technical compatibility or restrictions of using the component. Presenting technical high level information is very critical towards consumer time saving.

The input and the output schema is needed for the component integration. ICBD tries to be technology independent, so it gives the flexibility of documenting that in whatever the way. It could be XML, XML schema, UML class diagram.... etc.

Integration description document is mandatory item in the template. This documentation is from user (consumer) perspective, it describes how you could configure and use this component without going into any internal details of the component itself.

Technical description document is not mandatory item in the template, while we expect that it should be filled down in all the cases. It is not mandatory just to avoid making the template too complex. Missing this part makes the component drift to be black box component, it could be integrated as is, and otherwise it could not be adopted due to missing technical information. The proposed ICBD technical description document template is not presented in this paper, while any technical description document containing UML diagram fits here.

The last item in the template is the test cases scenarios. This field just describe the expected behaviour of this component. Without this document, the consumer only expects the behaviour from the functional description part. So it is needed for consumer decision either use this component or not. The unit test itself is part of the reusable component delivery, description of this unit test is expected to be allocated here.

4. ICBD ASSESSMENT AND EVALUATION

This section presents the evaluation of the ICBD and its pros and cons against component-based development and non-component based development

The pros of the ICBD model are: -

- ICBD increases the quality against the non-component based model due to the reusability of the components.
- The same project will act as a consumer and producer, this will increase number of reusable components added to the repository. The repository is expected to expand faster than what it does in the component based model
- ICBD increases the quality against the component based model. The count of the reusable component in the repository is more than what created from CBD model. This increase comes from injecting the reusability activities in daily project implementation activities which was not the case in CBD. More reusable components mean more reusability percentage, which impact the quality positively.
- The reuse will be a daily activity as it’s involved in many phases of the ICBD.
- The documentation process and templates proposed in ICBD model makes the selection phase easier than the CBD model and takes less time, automating the selection phase grantee better results accuracy.
- The structure of the team which is responsible for the generalization phase will minimize the duplication of components in the repository. This will decrease the time spent in the selection phase against the component based model and save any waste of unneeded generalization.
- The generalization of the components will be easier than in CBD, because it will be considered when designing the component, plus taking the reusability into consideration in all the projects normal activities (it is a subject of the real projects’ teams’ collaboration and understanding).

The cons of the ICBD model are: -

- The intense of the reusable component production depends on the willingness of the developers; however it could be encouraged by the organization according to the organization direction. The component’s production team is very critical part of the cycle, it could conduct a very rich generalization, or it could harm the process according to their performance, unit test, documentation....etc.
- Component selection and adaptation could be complex task in early organization transformation period from non-component to ICBD, but this complexity is decreased by time and becomes a part of the technical team culture.

Applying the model depends on the existence of qualified and experienced technical persons who are able to perform the generalization phase and sometimes complex modification of the selected components.

ICBD is currently under validation in real use case, a use case description is presented in the next section while it is still a running case study.

4.2. ICBD Case Study

Software development centre that contains around 200 technical (Architects, designer, and developers) was running on non-component based model. All this centre projects are in one domain with different customers (5 customers in same business domain). It tries to switch to component-based model to minimize the effort in the projects and utilized the common effort. It applied the post projects model. After the project delivery, the centre tries to find what could be extracted to be used in further projects. The really extracted components were around 5 fine-grained and medium-grained components (for example, reporting component for web applications) per quarter. The usage of these components was almost zero that comes from not encouraging the projects to select reusable components for using beside poor documentation of the component. Also the little number of components makes the options very poor, so the selection percentage is very low.

Using ICBD increases the number of reusable components to more than 70 components per quarter. This huge number of components beside the automation of selection indicates that, the reusability (consuming) in the next quarter would decrease the projects effort by 40%. We estimate that this percentage might increase to 70% after 2 quarters due to repository enrichment.

5. CONCLUSION

Component-based development is a trend for cost cutting and quality improvement in the software production world through reusing of already developed components. The component term is varying from very small scale component till a whole configurable project. The component sizes start from just a class in object oriented programming language (or even a method in a class) till a complete configurable product.

This paper discussed the reusability concept in component-based development (CBD). It mentioned the roles and responsibilities. It studies CBD model, its lifecycle, component based development guidelines.

Afterwards, it presents “Integrated Component-based Development” (ICBD) model. It is an integrated life cycle that conducts more productive software process by integrating the components producing and consuming phases into 1 complete lifecycle. ICBD is enriched by documentation templates, recommendations and mindset changes. This paper demonstrates ICBD phases and supplementary details and templates.

The theoretical comparison presents ICBD’s efficiency versus non-component and CBD models. Running case study shows very promising results that are mentioned in this paper. Using ICBD in application development centre that have around 200 technical guy and works in one business domain could decrease the effort of the new projects by 40% after 3 months of applying. This percentage estimated to be increased to 70% after 6 months as long as it is applied properly. This percentage of efforts’ cutting would decrease the cost of the new projects and obviously increase the quality of the new product, parts of this new product have been integrated and tested successfully in previous projects.

6. FUTURE WORKS

By time, more précised results of cost cutting are presented. The quality measurements and enhancements are not studied yet, it should be in the ICBD future verifications factors. More enrichment documentation templates (like integration and technical description documents) are under construction. Setting up these templates as standard within ICBD model would increase its productivity and make the component selection automation More efficient.

REFERENCES

[1] Alan Cameron Wills, (1999) "Component Based Development", TriReme International Ltd, http://www.trireme.com

[2] B.Jalender, Dr A.Govardhan & Dr P.Premchand, (2012) "Designing code level reusable software components", International Journal of Software Engineering & Applications (IJSEA), Vol. 3, No. 1, pp219-229

[3] Basem Y. Alkazemi, (2012) "On Verification of Software Components", International Journal of Software Engineering & Applications (IJSEA), Vol.3, No.5

[4] Carma McClure, (1997) "Software Reuse Techniques", Prentice Hall PTR, ISBN 0-13-6610000-5

[5] Ivica Crnkovic, Stig Larrson & Michel Chaudron, (2006) "Component-based Development Process and Component Lifecycle", International Conference on Software Engineering Advances (ICSEA'06), pp.44

[6] J Paul Gibson, (2009) "Software Reuse and Plagiarism: A Code of Practice", ITiCSE '09 Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education, pp 55-59

[7] Muthu Ramachandran, (2005) "Software Reuse Guidlines", ACM SIGSOFT Software Engineering Notes, Vol. 3, No. 3

[8] Nasib Singh Gill, (2003) "Reusability Issues in Component-based Development", ACM SIGSOFT SEN. Vol. 28 No. 6, pp. 30-33.

[9] P.Niranjan & Dr. C.V.Guru Rao, (2010) "A Mock-up Tool for Software Component Reuse Repository", International Journal of Software Engineering & Applications (IJSEA), Vol.1, No.2

[10] P.Shireesha & Dr.S.S.V.N.Sharma, (2010) "Building Reusable Software Component For Optimization Check in ABAP Coding", International Journal of Software Engineering & Applications (IJSEA), Vol.1, No.3

[11] Parastoo Mohagheghi, Reidar Conradi, Ole M. Killi & Henrik Schwarz, (2004) "An Empirical Study of Software Reuse vs. Defect-Density and Stability", Proceeding of the 26th International Conference on Software Engineering (ICSE'04), pp 282-292

[12] Sanjeev Patwa & Anil Kumar Malviya, (2012) "Reusability Metrics and Effect of Reusability on Testing of Object Oriented Systems", ACM SIGSOFT Software Engineering Notes, Vol. 37 Issue 5

Authors

Amr Rekaby is MSc in computer science (Grid Computing specialization) in 2012, worked as a technical coach / leader for 8 years in multinational companies like IBM and HP. He is now a founder of "Egyptian Research and Scientific Innovation Lab" (ERSIL), where he is working as a researcher in AI, software engineering and parallel computing fields.

Ayat graduated from the faculty of engineering Ain Shams University; worked in EDS/HP for 5 years as a software developer. She is part of the ERSIL research lab.