Reusable Software Component Life Cycle

Anas Bassam AL-Badareen, Mohd Hasan Selamat, Marzanah A. Jabar, Jamilah Din, Sherzod Turaev

***Abstract*—** In order to decrease the time and effort of the software development process and increase the quality of the software product significantly, software engineering required new technologies. Nowadays, most software engineering design is based on reuse of existing system or components. Also, it is become a main development approach for business and commercial systems. The concept of reusability is widely used in order to reduce cost, effort, and time of software development. Reusability also increases the productivity, maintainability, portability, and reliability of the software products. That is the reusable software components are evaluated several times in other systems before. The problems faced by software engineers is not lack of reuse, but lack of widespread, systematic reuse. They know how to do it, but they do it informally. Therefore, strong attention must be given to this concept.

This study aims to propose a systematic framework considers the reusability through software life cycle from two sides, build-for-reuse and build-by-reuse. Furthermore, the repository of reusable software components is considered, and the evaluation criteria from both sides are proposed. Finally, an empirical validation is conducted by apply the developed framework on a case study.

***Keywords*—** Software Reusability, Build for Reuse, Build by Reuse, Reusability Criteria, Software Quality, Quality Evaluation.

I. Introduction

In order to decrease the time and effort of the software development process and increase the quality of the software product significantly, software engineering required new technologies [2]. New horizons are opened, since the idea of software reuse appeared in 1968 [3]. Software reuse is an important and relatively new approach to software engineering [4]. It is become a main development approach for business and commercial systems [5]. Moreover, it is considered as one of the most important aspect used to improve the productivity of the software development [6].

Reusability is the degree to which a thing can be reused [7]. Software reusability represents the ability to use part or the whole system in other systems [8-11] which are related to the packaging and scope of the functions that programs perform [12]. Bitar [13] showed that, the reusability has to be considered as it is the most significant factor to improve the productivity and quality of the software development.

The concept of the software reuse is used to reduce the effort, cost, and time to develop a new system. Mohaghehi [14] and Philippow [15] state that the reusability is used to increase the productivity and reduce the developing time, which leads to complete the development faster and cheaper. According to Poulin [16], the US department of defense alone could save 300$ million annually by increasing its level of reuse as little as 1%. Moreover, software reusability aimed to improve productivity, maintainability, portability and therefore the overall quality of the end product [17].

Most software engineering design is based on reuse of existing system or components [5]. However, in order to achieve the benefits of the software reuse significantly it must be systematic [18]. Large organization need to introduce a systematic reuse in phases [19].

According to Ouyang [20], software reusability can be considered from two view points: design-by-reuse and design-for-reuse. Software-by-reuse is the use of existing application or its components to build new applications. Software-for-reuse is the ability of building applications that can be used all or part of it in other applications.

According to Prieto-Diaz [21], the concept of the software reuse is not only applied to source code fragment, but can also mean all of the information that are related to the product generating processes, including software requirements, analysis, design, and any information required by the developers to build a software. Moreover, Ramamoorthy [22] mentions that the reusability is not limited to the source code, but it has to include the requirements, design, documents, and test cases besides the source code.

Hence, the problem faced in software engineering is not lack of reuse, but lack of widespread, systematic reuse. Further, software engineers know how to adapt and reverse-engineer systems, but they do all of these processes informally. Sommerville [5]state that one of the main problems faced in software reuse is a lack of tools and reusable component library.

This study aims to propose a systematic framework considers the reusability through software life cycle from two sides, build-for-reuse and build-by-reuse. Furthermore, a repository of reusable software components is considered. Also, the evaluation criteria from both sides are proposed. Finally, an empirical validation is conducted by apply the

Anas Bassam AL-Badareen is with the University Putra Malaysia, 43400 UPM, Serdang, Selangor, Malaysia. Phone: 601-72301530; e-mail: anas_badareen@hotmail.com.
Mohd Hasan Selamat is a professor of software engineering with University Putra Malaysia, 43400 UPM, Serdang, Selangor, Malaysia. Phone: 603-89471720; e-mail: hasan@fsktm.upm.edu.my.
Marzanah A. Jabar PhD is with University Putra Malaysia, 43400 UPM Serdang Selangor, Malaysia; e-mail: marzanah@fsktm.upm.edu.my.
Jamilah Din PhD is with University Putra Malaysia, 43400 UPM Serdang Selangor, Malaysia; e-mail: jamilah@fsktm.upm.edu.my.
Sherzod Turaev is a Postdoctoral Researcher with University Putra Malaysia, 43400 UPM, Serdang Selangor, Malaysia; e-mail: sherzod@fsktm.upm.edu.my.

developed framework on a case study.

II. Reusable Components Extraction

Poor quality components may be unsuited for a reuse library [23]. Therefore, software product has to be designed and developed in certain method even it is requires an extra effort in order to be able to reuse it later [22]. The process of extraction of reusable component starts from project planning and match with all of the software development stages (see Figure 1). Thus it increases the quality of the reusable components and reduces the efforts required to build-by-reuse software.

In planning phase, the objective of the reusability is defined. The planning of component extraction identifies which component is needed to be extracted, how the process of extraction will be conducted, and when and where the extracted components can be used. Furthermore, the dependency and generality of the system goals are considered.

Requirement phase is the most important phase in build for reuse process, which makes the decision of build for reuse. According to the user requirements, software developers decide which subsystem can be built for reuse. Moreover, the reusable components library is checked to see whether these components exist. The decision is made when the required component does not exist in the library.

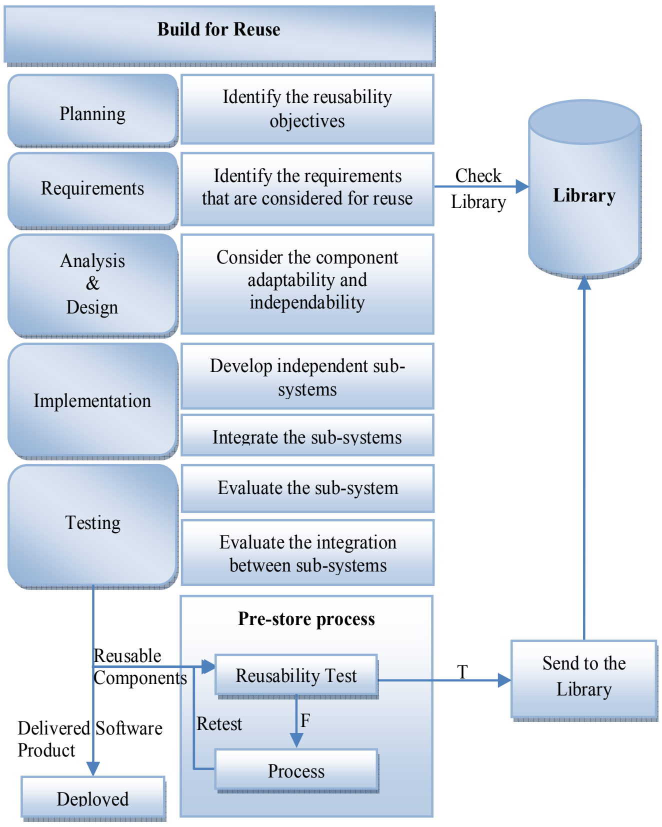
Build for Reuse
Planning
Identify the reusability objectives
Requirements
Identify the requirements that are considered for reuse
Check Library
Library
Analysis & Design
Consider the component adaptability and independability
Implementation
Develop independent sub-systems
Integrate the sub-systems
Testing
Evaluate the sub-system
Evaluate the integration between sub-systems
Pre-store process
Reusable Components
Reusability Test
T
Send to the Library
Retest
F
Process
Delivered Software Product
Deployed

Figure 1: Build-for-reuse Framework

In the system analysis and design, after the decision to build reusable components for specific requirements is made, the generality, portability, and coherence and coupling are the main issues of this phase. These characteristics are considered in designing reusable components and integrating these components to work together in order to perform the intended requirement.

Implementation phase, the independent sub-systems are developed, and the relationships among these sub systems are considered. Moreover, in the test phase, the evaluation of each sub system is independently conducted, and the ability of the sub-systems to work together and produce the expected result is also considered. Furthermore, in system documentation, more attention is given to the ability of the sub-system to work and its requirements to perform its goals.

As a result of the project, two software products are delivered instead of one. The first one is the required system, which is deployed in the market. The second product is the reusable components that are considered in the system. These components are sent to pre-store process in order to be processed and stored in the reusable software component library.

Pre-store process is a process of evaluate and enhance reusable components according to certain standard. Ramamoorthy [22] proposed a method of reusability-driven development. The concept of this method is adopted in the proposed framework (Figure 1), while the characteristics evaluated in this method are modified.

The reusability test consists of formal conditions that are required in the library. The process of reusable components is conducted till it passes the reusability test. Finally, when the component passes the reusability test, it is saved in the library to be used in other systems.

III. Reusable Components Adoption

The reusability is a concept of managing on how to use a system or some of it is components in the new system. As shows in figure 2, based on business objectives, software developers identify components that are able to be adopted in new system.

In requirement phase, system analyst checks the library whether it contains any reusable component suitable for each requirement. Moreover, the evaluation of the selected components is to check whether any modification is required before they are adapted to the system.

Reusability test in build-by-reuse is different than the test in build-for-reuse. Test for reuse considers the conditions required by reusable components library, whereas reusability test in build-by-reuse process considers the conditions required by a new system. Moreover, the conditions required by a library consider only the general characteristics of nonfunctional requirements, while in the test for reuse, functional and nonfunctional requirements are considered. These characteristics are evaluated specifically according to new system requirements.

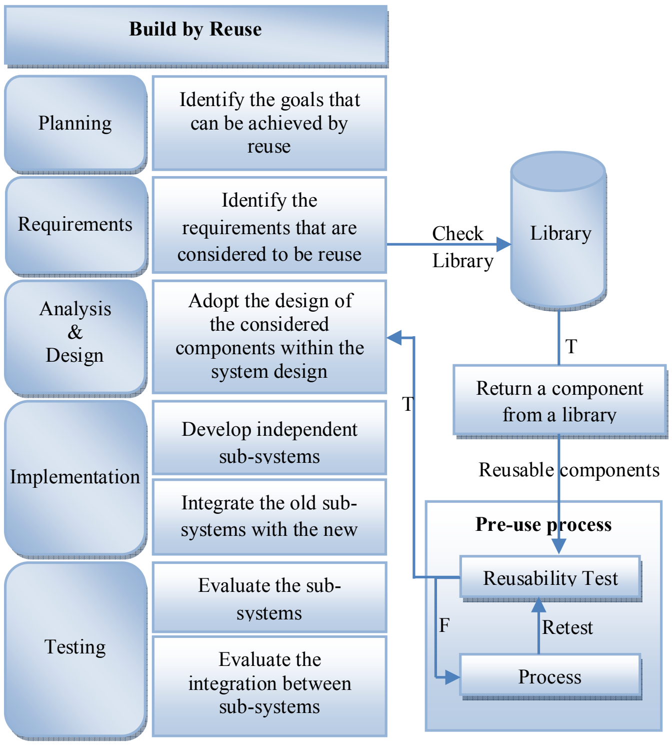
Build by Reuse
Planning
Identify the goals that can be achieved by reuse
Requirements
Identify the requirements that are considered to be reuse
Analysis & Design
Adopt the design of the considered components within the system design
Implementation
Develop independent sub-systems
Integrate the old sub-systems with the new
Testing
Evaluate the sub-systems
Evaluate the integration between sub-systems
Check Library
Library
T
Return a component from a library
Reusable components
Pre-use process
Reusability Test
T
F
Retest
Process

Figure 2: Build-by-reuse Framework

First step in system analysis and design phase is to consider their need in the new components. In the implementation phase, the sub systems are defined as a base of the system, in order to consider their requirements in the other sub-systems. Finally, the evaluation of the sub-system and the system integration test are conducted.

IV. Transfer From For Reuse to By Reuse

The link between develop for reuse and develop by reuse is very important in the success of the new project development. As shows in figure 3, the link between a build-for-reuse and build-by-reuse is a process consists of three phases, pre-store process, pre-use process, and reusable software component storage.

In order to decide whether the intended components able to achieve their functions properly within other system, project manager evaluates the ability of the components to work and communicate with other components within other systems in different platforms.

The reusability process, as mentioned above, considers the ability of the reusable components to achieve certain conditions that are required by reusable component library or new project.

Reusable component library is a repository of software components able to be used in different systems. Components exist in this storage have to follow certain standards and conditions. Hence, reusability test evaluates whether new component that needed to be stored in the library achieves these conditions. If the component passes this test, it will be send directly to the library. Otherwise, the process is required to modify this component in order to achieve a required standards and conditions in the library.

Component adoption, the required components returned from reusable component library have to be sent to reusability test. This test is different from reusability test in the extraction process. The components are returned from a library have been verified according to the library conditions. But in this test, it evaluates whether the component is suitable for a new system.

V. Reusability Evaluation

In order to transfer a reusable software component, it has to go through two main processes, pre-store process and pre-use process. Pre-store process is a process of modify and evaluate the extracted reusable components in order to satisfy the reusable component library. Pre-use process is a process of modify and evaluate the stored reusable components in order to be suitable and useable in a new system.

In these processes, the maintenance framework has been proposed in [24] is adopted. This framework, considered the characteristics of the modifiable software components, whereas the test phase intend to evaluate the reusability characteristics that discussed below. Figure 3, shows the reusable software components life cycle including pre-store and pre-use processes, which include a reusability test.

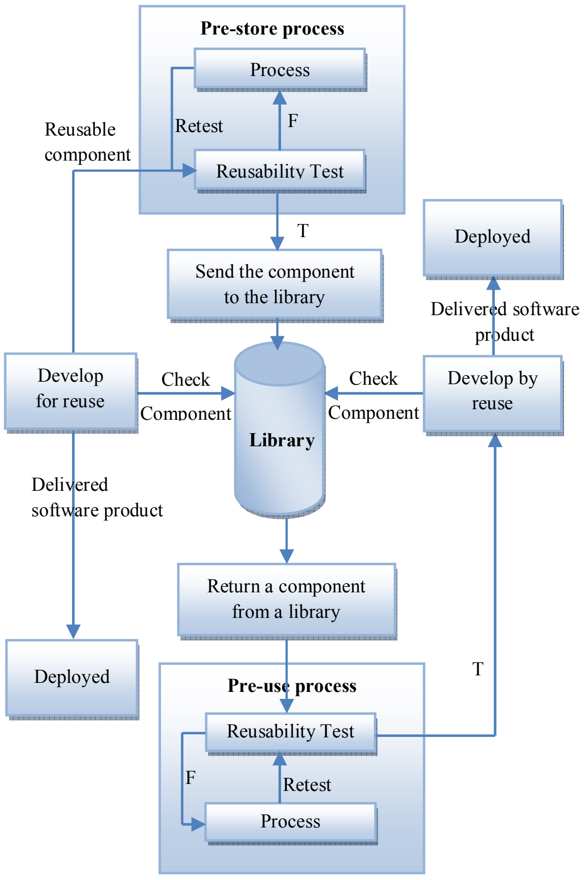

Figure 3: Reusable software component life cycle

A. Pre-store process

Reusability is concerned about how to use a system or its part in other systems. Several characteristics have to be considered. In this section the characteristics required to make the system qualify to be used in different system are considered. The first test which is required before store the extracted component in the library. It is intend to evaluate the general characteristics that required for any system as shown in figure 4.

Software coexistence considers the ability of the system or the sub-system to work in different platforms. This characteristic includes software system independence and machine independence. Software system independence represents the degree to which the program is independent of programming language features, operating system characteristics and other environmental constraints. Machine independence is the degree to which the software is de-coupled from its operating hardware.

The adaptability (interoperability) of the software is the degree of ability to communicate with other systems. It includes modularity, communication communality, and data communality. System modularity is a functional independence of program components that represents the degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

Communication commonality represents the degree to which standard interfaces, protocols and bandwidth are used. Data commonality is the explicit use of standard data structures and types throughout the program.

Software generality presents a level of abstraction to retrieve a result based on desired generality [25]. It is used to achieve a high level of abstraction that helps to solve a large class of problems even over different dimensions [26]. Therefore, it is defined as a degree to which a software product can achieve a wide range of functions [27].

The compliance verifies whether the software follows any standard or international certificates in order to build reusable software.

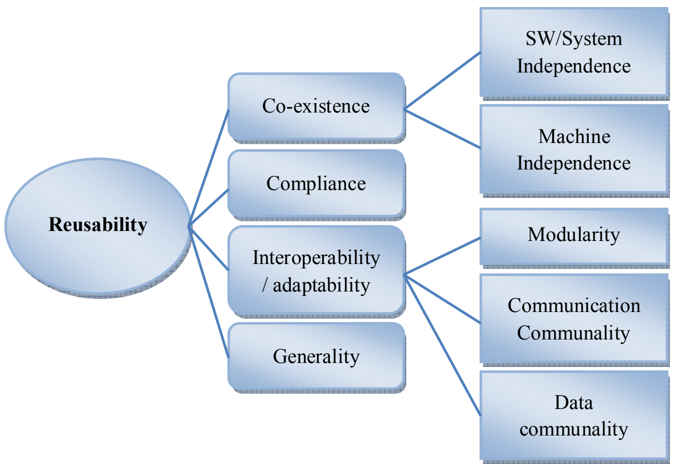

Figure 4: Pre-Store Characteristics

B. Pre-use process

The second process is required to retrieve and modify stored components from a reusable software component library in order to build new system. These components have to be satisfied with the new system requirements. Therefore, the functionality characteristic is the main factor has to be considered to present whether the retrieved component is suitable and able to achieve the requirements in the new system.

Software functionality consists of three main characteristics: suitability, accuracy, and compliance. Suitability presents the ability of the component to achieve the requirement of the new system and produce desired results. Accuracy presents the precision of the results required from the component. Compliance presents whether the component has been developed according to certain standard in order to achieve it is requirements.

In addition to the functionality of the software component, the ability of this component to work with the new system properly is considered. These characteristics have been measured previously in pre-store test. The evaluation has been conducted previously in pre-store process was intend to measure a general characteristics. While in pre-use process, the evaluation considers only the characteristics of the new system. Therefore, the measurement criteria consider only the characteristics of the new system. Table 1, shows the checklist of the evaluation that required for the new system.

These characteristics have been measured and modified previously in pre-store process. Therefore, this evaluation is started based on previous evaluation, which is required to make sure that the selected component is suitable for the new system. While, the other characteristics such as functionality are required to be measured after retrieve the stored component.

VI. REUSABILITY AND MAINTAINABILITY

Nowadays, several approaches are used in software component reuse, such as reusing components developed in-house, reuse of commercial-off-the-shelf (COTS) or open source software (OSS) components [14]. Sommerville [5], presents several ways to support software reuse. This study focused on in-house development approach. That the proposed framework considers the reusable component including it is design and implementation. Moreover, the reusable components are not considered as a main goal of the development project, but it is a secondary goal intended to be extracted during developing other system.

Therefore, the maintainability of the extracted components is considered as a one of the main issues needed to be discussed. This allow to modify the extracted components to meet a library conditions, and then to satisfy other systems.

Table 1: Pre-use evaluation checklist

Characteristic	Criteria	Question
Functionality	Suitability	Does the component able to achieve the user requirements
	Accuracy	Does the preciseness of the component result satisfy the user needs
	Compliance	Does the components followed any standard in order to perform the intended functions
Co-existence	SW/System Independence	Does the component able to work with the new system
	Machine Independence	Does the component able to work with the new machine
Compliance		Does the component follow any standard required in the new system
Adaptability	Modularity	Does the component an independent sub-system
	Communication communality	Does the component able to communicate with the new system
	Data communality	Does the component able to send and receive a data from the new system
Generality		Does the component able to be modified to be suitable for the new requirements

The framework proposed in [28] discussed the maintainability and the affects of the software quality on the maintenance process. As shows in figure 5, the maintenance process is classified into four main tasks, understanding, analyzing, modifying, and testing.

The ability of the reusable component to be modified in order to satisfy a reusable component library and requirements of new systems is an essential point in this framework. Therefore, the pre-store and pre-use processes are performed based on the maintenance framework.

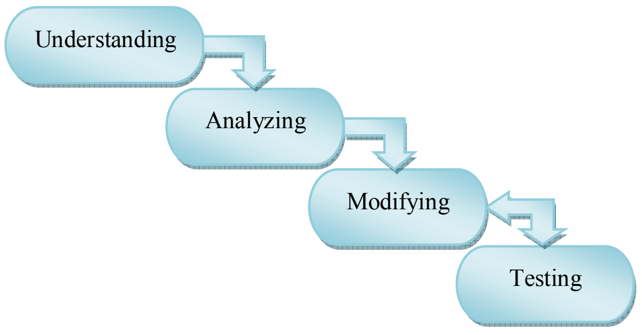

Figure 5: Software Maintenance Process

Component understanding represents the ability of the component to be understandable by software developers. In order to modify a software efficiently, it must be understandable [29].

The analyzability of the software component simplifies the task of identify a modification required in order to satisfy the new requirements and conditions[30].

The modification is a process of change a behavior and characteristics of component in order to satisfy new requirements.

Software test is an evaluation task required to evaluate whether the modification achieved it is goals. In software reuse, the test is conducted based on the requirements and condition required in both reusable component library and new software requirements.

The first modification process is required to satisfy general characteristics required in reusable software component library. The modification process intended to generalize a software component in order to be used in other software products.

The second modification is required to modify the stored software components and to change their characteristics from general to specific conditions required in the new system.

VII. RESULTS AND DISCUSSIONS

C-Registration System is documented in the Rational Unified Process (RUP Version 2003.06.00.65) document as an example of Web site project. This system enables students to register for courses on-line, professors to select their teaching courses and to maintain student grades, and registrar to maintain professors and students information.

The proposed framework is applied on Maintain Professor Information function within this system. Table 2 shows the general information required in Reusable Software Components Library. In addition to this information, more detailed description is required such as use case specification (Fig. 6), sequence diagram (Fig. 7), and source code.

The main form consists of general information about software components. The names of the components are modified to be more general. The sub-system maintaining professor information is changed to maintain user information, which is more general and suitable for many types of functions. In addition to create forms, the form of the component modification is also considered.

Component characteristics represent the ability of software component to achieve certain level of quality criteria. The values of these criteria are presented in details instead of specific value that are considered in order to calculate the reusability of the component. This description is clearer, usable, and specific in order to understand the real situation of the component, which is more meaningful for developing by

Table 2: Reusable Software Component Library Form

Component Name: *Maintain User Information*	**ID:** *1*	**Created Date:** *23-9-10*	**Responsible:** *Anas*
User Objective: *Allow the administrator to maintain the users' information (Add, Delete, and Update)*			
Actor: *Administrator*	**Trigger:** *The admin selects " Maintain User Information" activity from Admin menu*		

Component Characteristics		
Co-existence	SW/System Independence	The component passed the test of work with MS Windows XP, 2007, and Unix OS.
	Machine Independence	The components passed the test of 32 and 64 Bit PC
Compliance		The development process was conducted based on IEEE Standard for Developing Software Life Cycle Processes, 1998 [1]
Interoperability	Modularity	This component is developed as a one independent part, based on its use case
	Communication Communality	The ports of communication are defined in Process flow Part
	Data Communality	The ports of communication and data type are defined in Process flow and attributes Parts
Generality		The component is able to work with data described in Attribute Part

Process Flow			
Inputs		**Output**	
Description	**Source**	**Description**	**Destination**
User Info. Request	*Admin*	*User info*	*Admin*
New user info.	*Admin*	*UserID*	*User Info. DB*
Modified User Info.	*Admin*	*New User Info.*	*User Info. DB*
Deleted User Info.	*Admin*	*Updated User Info.*	*User Info. DB*
User Details	*User Info. DB*	*Deleted User Info.*	*User Info. DB*

Data Store		
Description: *This entity is used to store a general user information*		
Accessibility		
Source	**Type**	**Description**
Admin	*Create, Modify, Deleted, View*	
User	*View*	

Attributes			**Primary Key:** *UserID*	
Attribute	**Description**	**Acceptable Value**	**Format**	**Constraint**
UserID	*Identifier attribute*	*Number only*	*9 digits, no space*	*Unique, Not null*
Affiliation		*Text*	*30 characters*	*Not null*
Department		*Text*	*30 characters*	*Not null*
Occupation		*Text*	*30 characters*	*Not null*
Address		*Text*	*50 characters*	*Not null*
Phone		*Number*	*20 characters*	*Not null*
Mobile		*Number*	*20 characters*	*Not null*

reuse processes.

The process flow represents the ports of the components that used to communicate with other components or external systems. Moreover, the description of these ports shows the type of actions and data that are allowed to go through it.

The data store shows the storage required by the component in order to perform its tasks. The characteristics of the required storage and the information shared with other components or systems are included. They are used to check the information status, especially for security purposes.

The attributes of the storage show the information formats that the component can deal with and the special roles in the component. Moreover, this information shows the coupling between the component and the other components or systems, such as the primary keys and foreign keys.

The component interoperability is clearly detailed out in the process flow and data store. The process flow shows the functions required from other components. The attributes in data store shows the data that required from other components or external systems.

1. Use Case: *Maintain User Information*

1.1 Brief Description

- **Definition:** This use case allows the administrator to maintain user information in the system. This includes adding, modifying, and deleting user from the system.
- **Actors:** The actor of this use case is the administrator.

1.2 Pre-Conditions

- The administrator registrar has been logon to the system

1.3 Characteristic of Activation

- The use case begins when the administrator selects the "Maintain user information" activity from the Main Form.

1.4 Flow of Events

a. Basic Flow

- *Add a User*
  - ❖ The administrator selects "add a user".
  - ❖ The system displays a blank user form.
  - ❖ The administrator enters the following information for the user: name, affiliation, department, occupation, address, phone, and mobile.
  - ❖ The system validates the data to insure the proper data format *<<Invalid Data Format>>* and searches for an existing user with the specified name *<<User Already Exists>>*.
  - ❖ If the data is valid the system creates a new user and assigns a unique system-generated id number.
  - ❖ This number is displayed, so it can be used for subsequent uses of the system.
  - ❖ Steps 2-4 are repeated for each user added to the system.
  - ❖ When the administrator is finished adding users to the system the use case ends.

b. Alternative Flow

- *<<Cancel>>*
  - ❖ The user can cancel the operation at which point the use case.
- *<<Modify a User>>*
  - ❖ The administrator selects "Modify a user."
  - ❖ The system displays a blank user form.
  - ❖ The administrator types in the user id number he/she wishes to modify
  - ❖ The system retrieves the user information and displays it on the screen, if no data found the *<<User Not Found>>* sub-flow is executed.
  - ❖ The administrator modifies one or more of the user information fields.
  - ❖ When changes are completed, the administrator selects "save" button.
  - ❖ The system validates the data to insure the proper data format *<<Invalid Data Format>>* and then updates the user information.
  - ❖ Steps 2-7 are repeated for each user the administrator wants to modify.
  - ❖ When edits are complete, the use case ends.
- *<<Delete a User>>*
  - ❖ The administrator selects "Delete a User".
  - ❖ The system displays a blank user form.
  - ❖ The administrator types in the user's id number for the user that's being deleted.
  - ❖ The system retrieves the user and displays the information in the form, if no data found the *<<User Not Found>>* sub-flow is executed.
  - ❖ The administrator selects "delete".
  - ❖ The system displays a delete verification dialog confirming the deletion.
  - ❖ The registrar selects "yes".
  - ❖ The user is deleted from the system.
  - ❖ Steps 2-8 are repeated for each user the administrator wants to delete.
  - ❖ When the administrator finished deleting user from the system, the use case ends.

c. Exceptional Flow

- *<<User Already Exists>>*
  - ❖ In the *<<Add a User>>* sub-flow, if the user already exists with the specified name, an error message, "User Already Exists", is displayed.
  - ❖ The administrator can either change the name or choose to create another user with the same name.
- *<<User Not Found>>*
  - ❖ In the *<<Modify a User>>* or *<<Delete a Professor>>* sub-flows, if the user with the specified id number does not exist, the system displays an error message, "User Not found".
  - ❖ Then the administrator can type in a different id number.
- *<<Invalid Data Format>>*
  - ❖ In the *<<Modify a User>>* or *<<Add a User >>* sub-flows, if a typed data are not correct, the system displays an error message.
  - ❖ Then the administrator can type in a different data.

1.5 Post Condition

- *<<Add a User >>* the User is able to logon to the system and use its functions.
- *<<Modify a User>>* the user can use the new data updated.
- *<<Delete a User>>* the user is not able to use a system any more.

1.6 Roles

1.7 Constraints

Fig. 6: Use case description, Maintain User Information

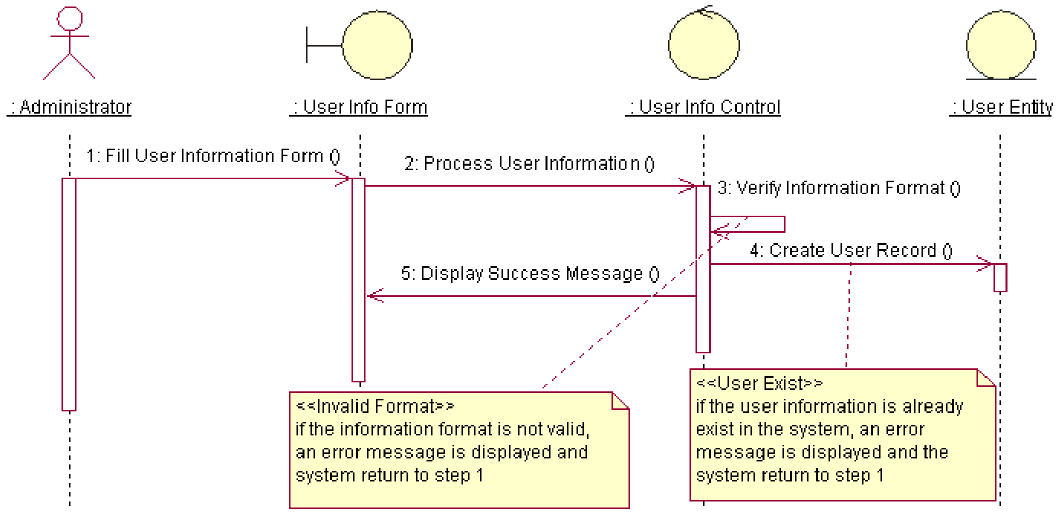
: Administrator
: User Info Form
: User Info Control
: User Entity
1: Fill User Information Form ()
2: Process User Information ()
3: Verify Information Format ()
4: Create User Record ()
5: Display Success Message ()
<<Invalid Format>> if the information format is not valid, an error message is displayed and system return to step 1
<<User Exist>> if the user information is already exist in the system, an error message is displayed and the system return to step 1

Figure 7: The Characteristics of the Software Reusability

VIII. Conclusion

In order to reduce the cost, effort, time, and to increase the quality and productivity of the software development process, the concept software reuse is used. Software reuse is used as a base of most of software design.

The reusability problem in software engineering is not a lack of use, but lack of systematic reuse. Software engineers know how to do it, but they do it informally. Moreover, Poor software components quality may be unsuited for a software reuse library or any other software system. Therefore, lack of software reuse tools and library is one of the main problems faced in software reuse.

This study proposed a reusable software component framework. The framework, considers the reusable software components through its life cycle, from both sides build-for-reuse and build-by-reuse development. The framework consists of reusable component extraction, adoption, storage, and pre-store and pre-use processes. The extraction process is considered during develop-for-reuse process, which focused on how to extract suitable information for a reusable component. The adoption process is a process of how to add a reusable component to new system during development stages.

The pre-store process is conducted to evaluate and modify the extracted components in order to satisfy reusable component library. The pre-use process is process of evaluate and modify the reusable components that retrieved from the library in order to satisfy new system requirements.

Whereas, the reusable framework required maintaining and modifying the reusable components, the maintenance process is adopted. The process of maintenance helps to modify the reusable components in both pre-store and pre-use processes efficiently.

Applying the framework on C-Registration system shows that the proposed framework simplifies the job of software developers, which required a little effort in order to build for reuse or build by reuse. The future work, we propose to apply the framework in different types of software systems.

References

[1] "IEEE Standard for Developing Software Life Cycle Processes," *IEEE Std 1074-1997,* p. i, 1998.

[2] D. C. Rine and N. Nada, "An empirical study of a software reuse reference model," *Information and Software Technology,* vol. 42, pp. 47-65, 2000.

[3] P. Gomes and C. Bento, "A case similarity metric for software reuse and design," *Artif. Intell. Eng. Des. Anal. Manuf.,* vol. 15, pp. 21-35, 2001.

[4] M. Burgin, *et al.*, "Software technological roles, usability, and reusability," in *Information Reuse and Integration, 2004. IRI 2004. Proceedings of the 2004 IEEE International Conference on*, 2004, pp. 210-214.

[5] I. Sommerville, *Software Engineering*, 8th ed. London: Addison-Wesley, 2007.

[6] I. PHILIPPOW, *et al.*, "Methodical Aspects for the Development of Product Lines," in *Information Science and Applications '02 (2nd ISA)*, Cancun, Mexico, 2002, pp. 1391-1396.

[7] W. Frakes and C. Terry, "Software reuse: metrics and models," *ACM Comput. Surv.,* vol. 28, pp. 415-435, 1996.

[8] J. A. McCall, *et al.*, "Factors in Software Quality," *Griffiths Air Force Base, N.Y. Rome Air Development Center Air Force Systems Command,* 1977.

[9] N. S. Gill, "Reusability issues in component-based development," *SIGSOFT Softw. Eng. Notes,* vol. 28, pp. 4-4, 2003.

[10] C. Luer, "Assessing Module Reusability," in *Assessment of Contemporary Modularization Techniques, 2007. ICSE Workshops ACoM '07. First International Workshop on*, 2007, pp. 7-7.

[11] F. Haiguang, "Modeling and Analysis for Educational Software Quality Hierarchy Triangle," in *Web-based Learning, 2008. ICWL 2008. Seventh International Conference on*, 2008, pp. 14-18.

[12] J. J. E. Gaffney, "Metrics in software quality assurance," presented at the Proceedings of the ACM '81 conference, 1981.

[13] I. Bitar*, et al.*, "Lessons learned in building the TRW software productivity system," 1985.

[14] P. Mohagheghi and R. Conradi, "An empirical investigation of software reuse benefits in a large telecom product," *ACM Trans. Softw. Eng. Methodol.,* vol. 17, pp. 1-31, 2008.

[15] I. PHILIPPOW, "Utilization of Object-Oriented Models," in *WSES International Conference on Multimedia, Internet, Video Technologies 2001*, Malta, 2001.

[16] J. S. Poulin, "Measuring software reusability," in *Software Reuse: Advances in Software Reusability, 1994. Proceedings., Third International Conference on*, 1994, pp. 126-138.

[17] A. Sharma*, et al.*, "Reusability assessment for software components," *SIGSOFT Softw. Eng. Notes,* vol. 34, pp. 1-6, 2009.

[18] W. B. Frakes and S. Isoda, "Success factors of systematic reuse," *Software, IEEE,* vol. 11, pp. 14-19, 1994.

[19] R. Prieto-Diaz and G. Arango, *Domain analysis and software systems modeling*: IEEE Computer Society Press Los Alamitos, CA, USA, 1991.

[20] Y. Ouyang and D. L. Carver, "Enhancing design reusability by clustering specifications," presented at the Proceedings of the 1996 ACM symposium on Applied Computing, Philadelphia, Pennsylvania, United States, 1996.

[21] R. Prieto-Diaz, "Status report: software reusability," *Software, IEEE,* vol. 10, pp. 61-66, 1993.

[22] C. V. Ramamoorthy*, et al.*, "Support for reusability in Genesis," *Software Engineering, IEEE Transactions on,* vol. 14, pp. 1145-1154, 1988.

[23] G. W. Hislop, "Analyzing existing software for software reuse," *Journal of Systems and Software,* vol. 41, pp. 33-40, 1998.

[24] A. B. AL-Badareen*, et al.*, "Software Quality Evaluation through Maintenance Processes," in *European Conference of Computer Science (ECCS '10)*, Puerto De La Cruz, Tenerife, 2010, pp. 131-134.

[25] S. Hyun Woong*, et al.*, "Measuring Generality of Documents," in *Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on*, 2006, pp. 62-62.

[26] S. Vey and A. Voigt, "AMDiS- Adaptive multidimensional simulations: object oriented software concepts for scientific computing," *WSEAS Transactions on Systems,* vol. 3, pp. 1564-1569, 2004.

[27] N. Ram and P. Rodrigues, "Intelligent risk prophecy using more quality attributes injected ATAM and design patterns," 2008, pp. 169-173.

[28] A. B. AL-Badareen*, et al.*, "The Impact of Software Quality on Maintenance Process," 2010.

[29] M. Feathers, "Before clarity [software design]," *Software, IEEE,* vol. 21, pp. 86-88, 2004.

[30] A. Kumar*, et al.*, "A quantitative evaluation of aspect-oriented software quality model (AOSQUAMO)," *SIGSOFT Softw. Eng. Notes,* vol. 34, pp. 1-9, 2009.