A STRUCTURAL APPROACH TO IMPROVE SOFTWARE DESIGN REUSABILITY

Tawfig M. Abdelaziz, Yasmeen.N.Zada and Mohamed A. Hagal

University of Benghazi, Faculty of Information Technology,
Department of Software Engineering
tawfig@cs.uni-essen.de, yasmeen.zada@hotmail.com
and Mohamed.hagal@benghazi.edu.ly

ABSTRACT

Software reuse become a very promising area that has many benefits such as reducing costs, time, and most importantly, increasing quality of software. However, the concept of reuse is not only related to implementation level, in fact, it can be included in the earlier stages of the software development life cycle such as design stage. Adopting reuse at this stage provides many benefits such as increasing productivity, saving time and reducing cost of software development.

Accordingly, this paper presents the concept of reuse at design level in more details. As well as, it proposes an approach to improve the reusability of software design using the directed graph concept. This leads to produce a design to be considered as reusable components which can be adapted in many software systems.

KEYWORDS

Software Reusability, Software Component, Unified Modeling Language (UML), Parameterization, Directed Graph.

1. INTRODUCTION

Software reuse is a fundamental aspect of high quality software. Effective reuse of software products is increasing productivity, saving time and reducing cost of software development. However, as the concept of reusing software components is very clear at the code level, while the same concept becomes more difficult to address when discussed in the context of reusing designs. The problem with design reuse in Software Engineering is the shortage of guidelines or approaches that support and guide the designers to be useful from previous design components.

In response to this, some researches related to reuse at design stage presented approaches that aim to improve design reusability. Gui and Scott in [1] worked on measuring software reusability by applying coupling and cohesion metrics on java components, and also focused on reflecting the complexity of those components to be used in reusability activities. Kang, Cohen and Holibaugh

David C. Wyld et al. (Eds) : CST, ITCS, JSE, SIP, ARIA, DMS - 2014
pp. 163–171, 2014. © CS & IT-CSCP 2014 DOI : 10.5121/csit.2014.4116

in [2] proposed the work based on the refinement of the software lifecycle to identify reuse activities. This work concentrated more on identification of reusable resources than constructing reusable resources. Price and Demorgian[3]measure object oriented design reusability focusing on abstraction concept using metrics to measure coupling and cohesion dependency relationships. Mishra's Misra's [4]used reverse engineering of legacy software to create reusable components as an attempt to understand the re-existing software by re-designing it. Johnson and Russo[5] described design techniques that support abstract classes and framework. It provided a way to express the design to customize it, developing frameworks and tools that facilitated the design reusability.

The work in this paper is motivated primarily by the possibility of improving and increasing the degree of reusability of design in any software system by discussing the concept of reuse associated with the level of design. It presented a structural approach that is mainly considered with the improvement of software design reusability, to result a design that is potentially reusable. The following section describes the steps of the proposed approach that is expected to improve the design reusability.

2. THE PROPOSED APPROACH

The work of this paper was motivated by the design reuse model represented and illustrated in [6], and the design for reuse process of that model was the base of the proposed approach introduced in this section.

The proposed approach consists of four activities as shown in Figure 1: Design classes, Refinement, Reusable packages and Documentation.

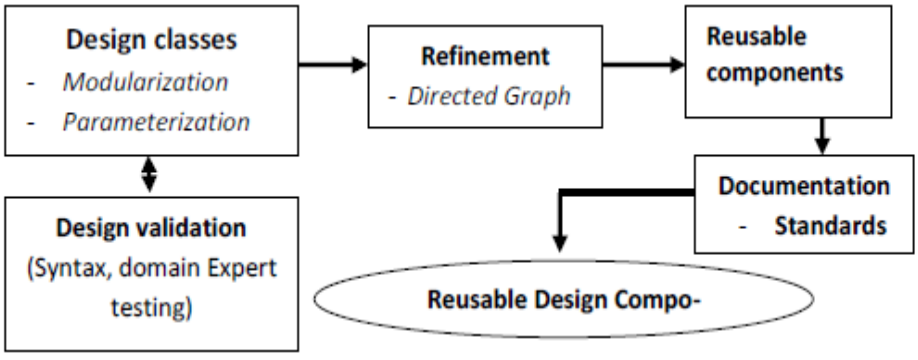

Fig. 1. Conceptual overview of the proposed approach

These activities will produce design components that are ready to be reused in many software systems. Figure 1 illustrates a conceptual overview of the proposed approach. The directions show the priority of the activities' steps. So, it must be considered that it cannot be move from Design classes step unless the design validation activity effectively achieved. This insures that mistakes are detected and handled early.

2.1 Design classes

The first activity in this approach is to decide major classes of the system design. The technique used to decide what classes are needed is to go through the software requirements and identify the nouns (entities) that can be considered as classes. After that, identify the characteristics (state and behavior) of each class, and then define the relationships between these classes. Finally, construct a class diagram as shown in figure 2.The structure of a system is represented using class diagrams. Therefore, modularization and the parameterization concepts must be taken into consideration in this process, due to their great impact on reusability of design. The use of modularity concept makes a system design to be considered as a set of smaller parts that should satisfy the quality concepts such as maintainability, testability and reusability. The effective modularity can be achieved by developing functional independence modules with single-minded function and refusal to excessive interaction with other modules.

Modularity and functional independence could be measured by two qualitative criteria: coupling and cohesion. Coupling is "a measure of inter-module connectivity, and is concerned with identifying the forms of connection that exist between modules" [7].Cohesion, in its turn, provides "a measure of the extent to which the components of a module can be considered to be ‘functionally related'. The ideal module is in which all the components can be considered as being solely present for one purpose"[7].Furthermore, parameterizing methods of classes is a very important activity to improve design reusability [3].Figure 2 illustrates an example of how a method (operation) can be parameterized in a class diagram at the design level. The Employee class on the left shows a duplication of method raise() to do same things with different values fivePercentRaise() and tenPercentRaise(). Where, the Employee class on the right shows the parameterization of the method by adding a parameterpercentage to the raise() method which later prevents duplication, which will not require much effort in creating the program, and most importantly, makes it easier to reuse the design with no much information to be included.

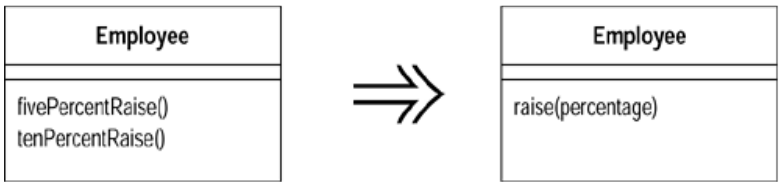

Fig. 2. Parameterizedmethod [11]

Another important note regarding with class attributes, a class with one or two attributes should be focused on, this may indicate that those attributes belong to another class related to the first class [9]. Therefore these attributes could be aggregated into one class. In this paper, Figure 3 illustrates an example of a small hospital system.

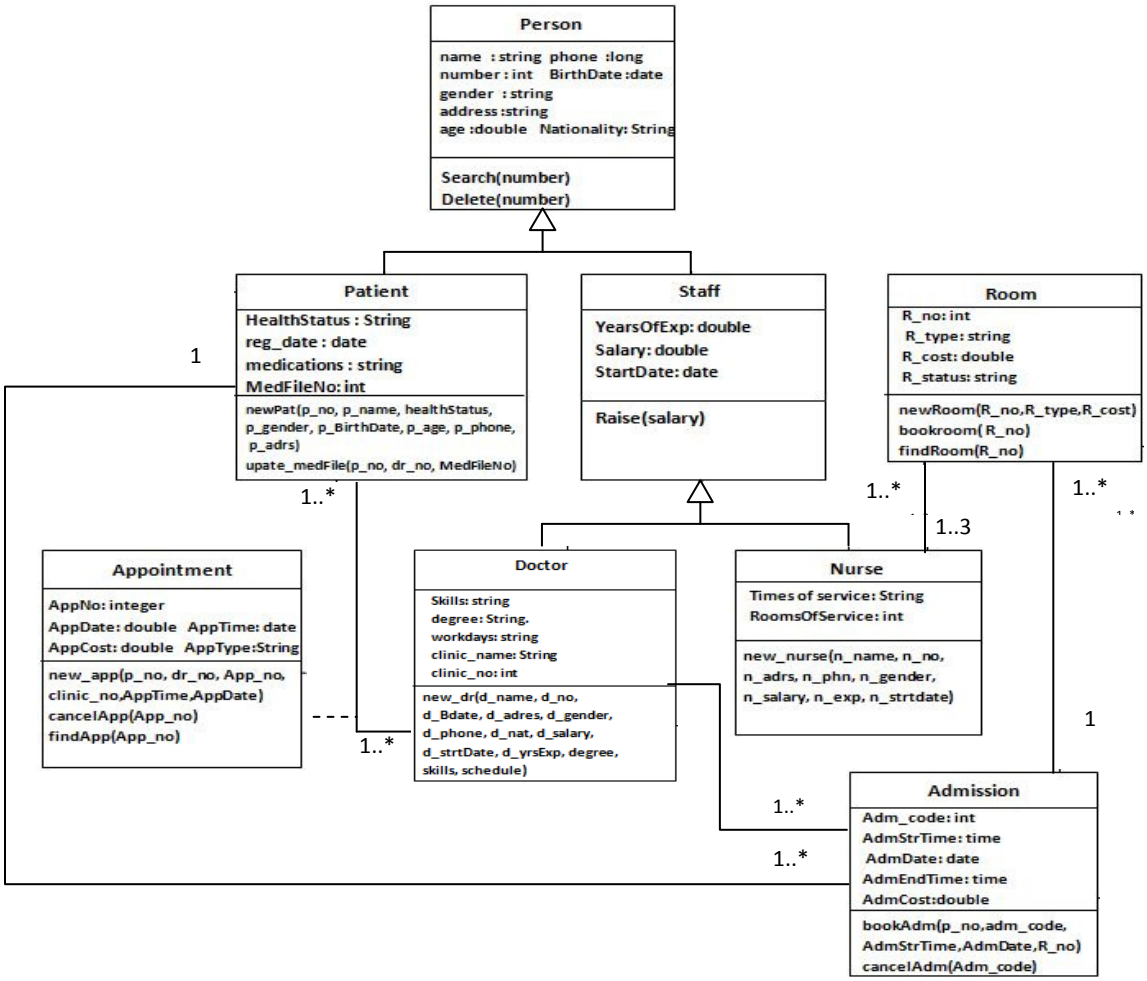

Fig. 3. Health- care UML Class diagram

It consists of the following classes: person, patient, staff, doctor, nurse, clinic, room, appointment, admission, surgery, and test. Modularity concept was represented by a class diagram and their methods are parameterized to increase reusability by providing more information.

2.2 Design Validation

To be able to reuse a component, and to insure that a component is reliable and ready to be reused, you have to make sure that this component does not contain any defects or errors.

Since the design validation work in this paper is based on the approach presented in [14] for class diagram testing. Therefore, UML class diagrams can be tested using some independent methods: Syntax Testing and Domain Expert Testing. Syntax testing is used to verify that the class diagram is correctly and properly constructed. Accordingly, three questions need to be answered: Is it complete? Is it correct? Is it consistent?. Then the domain expert testing is used to insure that the design is correct. Table 1and Table 2, illustrate the syntax and domain expert testing for the chosen health-care systems.

Table 1. Syntax Testing of Health care system class diagram

1.	Does each class define attributes, methods, relationships, and cardinality?	✓
2.	Is each associations' and aggregations' cardinality correct?	✓
3.	Are all parameters explicit rather than being embedded in method names?	✓
4.	Do all subclasses implement the "is-a-kind-of" relationship properly?	✓
5.	In inheritance structures, are all attributes and methods pushed as high in the inheritance structure as is proper?	✓
6.	Does each association reflect a relationship that exists over the lives of the related objects?	✓
7.	Are each 0..* and 1..* relationships implemented ?	✓

Table 2. Domain Expert Testing of Health care system class diagram

1	Is each class named with a strong noun?	✓
2	Is each attribute defined within the proper class? Is it of the correct type?	✓
3	Is each method in the correct class?	✓
4	Are all method names strong verbs?	✓
5	Does each method take the correct input parameters and return the correct output parameter?	✓
6	Does each method implement one and only one behavior?	✓

2.3 Refinement

The main goal of the refinement step is to produce reusable components. This will be achieved by convert the class diagram into a directed graph. Each class is represented as a node in the graph and the directions of the edges is the direction of dependencies between classes as shown in figure 4. The idea behind this transformation is to introduce a technique to help increasing cohesion of design and improve reusability as a result.

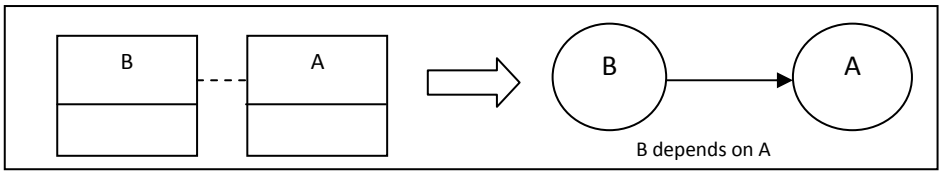

Fig. 4. Transforming classes to a graph

Attention should be paid to the inheritance relationships in a class diagram when transforming it into a directed graph. This relation is represented by a dotted edge between nodes in the graph. The nature of object oriented design with inheritance is to migrate more general information and operations up to the hierarchy where they can be reused by all descendants. Therefore when there is a need to reuse a child class it should be also reuse the parent class. However, when there is a need to reuse the parent class, it is not required to reuse its subclasses, since the parent class does not use members of its subclasses.

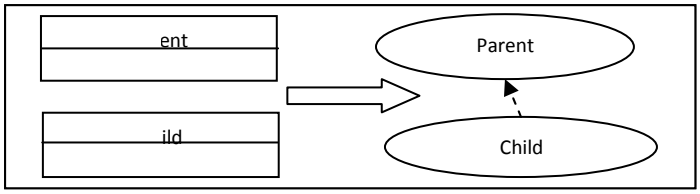

Fig. 5. Inheritance to directed graph

As a result, new systems can reuse the top portion of a hierarchy or the whole hierarchy, but they cannot reuse just a lower part of a hierarchy. Figure 5 shows an illustration of transforming inheritance relationship between classes to a directed graph.

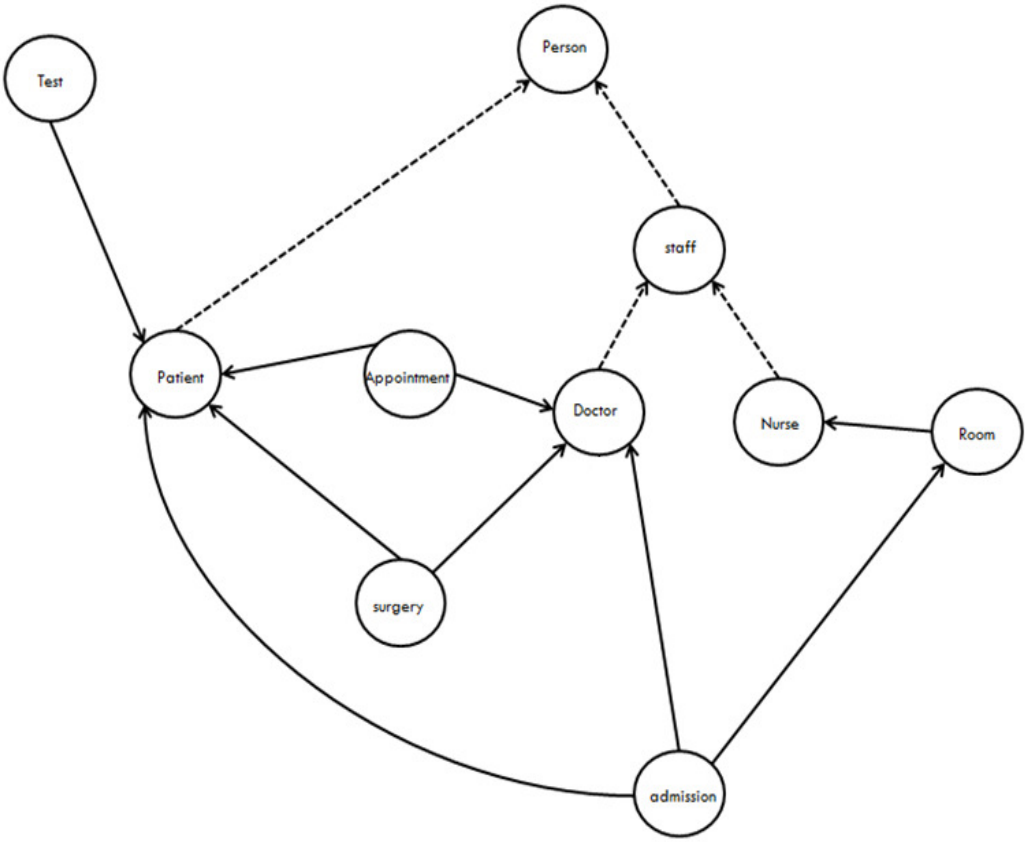

Fig. 6. Directed graph of a health-care system

Reusing just the top portion of a hierarchy is desirable in many cases, since the lower level classes are more specific classes, so with respect to their parents, they are less likely to be needed in other applications [3]. The following figure (Figure 6) illustrates the transformation of the figure 3 from class diagram into a full directed graph with taking into consideration the direction of dependencies and the inheritance relationships between classes.

2.4 Reusable Components

The purpose of this step is to extract the reusable components (packages) of the design. This will be achieved by grouping the nodes with the same directions, which was determined according to the relationships between classes of the system to deal with them as independent system components. So, every sub-graph (component) can be reused separately. Also, from another perspective, we can say that each component is a cohesive module (component), where every member is related with other members of the same module without the need to have dependencies on other modules. Figure 7 illustrates some examples of reusable components that can be generated from the directed graph illustrated in Figure 6.

Fig. 7. Reusable packages

2.5 Documentation

The importance of documentation in software component reuse is critical. It needs accurate information about a component in order to state the component with a requirement by referencing the Software Design Document (SDD) [12][13]. Although good documentation of components is essent enwhaaf

Some advice was provided in [10] about what should be included in a reuse document. Table 3 illustrates of documentation of two selected packages as an example from the given Health care system.

Table 3. Component documentation example

Component name	**Identification**	**Specification**	**Technical restrictions**	**Commercial or legal restrictions**	**Problems**	**Recommended enhancements**
R.Package1	Reusing this component is when the system needs to have patient information	Patient class that inherits from Person class, which contains all the properties and operations that need to be done for a patient	VB.Net programming language	This component is suitable only for health care systems	None (component is tested and bugs are detected and fixed)	This components can be integrated with other components to extend functionality
R.Package2	Reusing this component is when the system needs to have staff information	Staff class that inherits from Person class, which contains all the properties and operations that need to be done for a Staff member	VB.Net programming language	This component is suitable for health care or management systems	None (component is tested and bugs are detected and fixed)	This components can be integrated with other components to extend functionality

3. CONCLUSION

Reuse at design level is an important aspect that should be focused on during the software development life cycle. The proposed approach described how to improve design reusability in a structured way that can be applied on any software design in order to produce design of reusable components. The use of directed graphs helps the designers to understand how to extract the design components by putting them in a form of nodes and directed edges that can be grouped into packages (components) that could be reused in other systems.

As a future work, we are considering to develop a tool that performs the steps of the approach by providing the class diagram, and then the tool performs the refinement process of the class diagram to a directed graph and generates reusable packages of the produced.

REFERENCES

[1] Gui, G & Scott, P.D. (2009), "Measuring software component reusability by coupling and cohesion metrics", Journal of computers, Vol.4, No.9.

[2] Kang, K.C, Cohen, S &Holibaug H.R. (1992), "Reuse-Based Software Development Methodology", (Report No. SEI-92-SR-4). Application of Reusable Software Component Project.

[3] Price, M.W &Demorgian, S.A. (1997), "Analyzing and measuring reusability in object oriented designs", University of Connecticut, computer science and engineering department.

[4] Mishra, S.K, Kushwaha, D.S & Misra, A.K. (2012),"Creating reusable software components from object oriented legacy system through reverse engineering", Journal of object technology, pp 1-13.

[5] Johnson, R.E & Russo, V.F. (1991), "Reusing object oriented designs", Unpublished Manuscript, department of computer science, University of Illinois, West Lafayette.

[6] Duffy, S.M, Duffy, AHB &MacCallum, K.J. (1995),"A Design Reuse Model". International conference of engineering design (iced 95): Glasgow, Heurista, 490-495.

[7] Budgen, D. (2003), "Software design", England: Pearson education.

[8] Price, M.W &Demorgian, S.A.(1997) "Analyzing and measuring reusability in object oriented designs", University of Connecticut, computer science and engineering department.

[9] Chidamber, S.R &Kemerer, C.F.(1994),"A Metrics Suite for Object Oriented Design", IEEE Transaction on Software Engineering, Vol,20. No, 6.

[10] Sametinger,J.(1996) "Reuse Documentation and Documentation Reuse", A&M University, Texas, USA.

[11] Sourcemaking.com/refactoring, Access: (January,2013).

[12] Jones, M & Mortensen, U. (1995), "Guide to the software detailed design and production phase", ESA publications divisions,Vol.1: Paris, France.

[13] Kuhns, R.D. (1998) "Strategies for designing and building reusable GIS application components", Unpublished Manuscript, Convergent Group, Englewood, Colorado.

AUTHORS

Tawfig M. Abdelaziz

He is an assistant professor at the department of software engineering and a vice dean of faculty of Information Technology, University Of Benghazi, Libya. He is interesting in Agent systems, software Quality Assurance, software project management and Formal methods

Yasmeen.N.Zada

She is a graduate student at Department of Software Engineering, Faculty of Information Technology, University of Benghazi, Libya.

Mohamed Ali Hagal

He is a lecturer at the department of software engineering, faculty of Information Technology, Benghazi University-Libya. He is interesting in software requirements engineering, software design and software project management.