العنوان:	Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design
المؤلف الرئيسي:	الشندودي، أحمد بن سيف بن محمد
مؤلفين آخرين:	Al Khanjari, Zuhoor Abdullah Salim(Advisor)
التاريخ الميلادي:	2014
موقع:	مسقط
الصفحات:	1 - 95
رقم MD:	972147
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة السلطان قابوس
الكلية:	كلية العلوم
الدولة:	عمان
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، تصميم البرمجيات، المنهجية التسلسلية التقليدية، برمجة تحليل الشبكات الاجتماعية، إعادة التصميم، هندسة البرمجيات الموضوعية
رابط:	https://search.mandumah.com/Record/972147

Chapter 1: Introduction

Software, as a product, delivers the computing potential embodied by computer hardware. It is used to transform, produce, manage, acquire, modify, display, or transmit information. Information can be as simple as a single bit or as complex as a multimedia simulation (Pressman, 2014).

Software development or application development is the process of developing a single application or a full functional integrated system. In its basic form, it is the coding or implementation of the application at hand, but in a more broader professional sense it is all that is involved between the conception of the desired software's specifications through to the final manifestation of the software, ideally in a planned and structured process.

As the complexity of the desired solution or system increases, the need for a well planned and well executed process increases. The engineering aspect of the software development addresses this issue. Software Engineering is defined as the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; in other words it is the application of engineering techniques and strategies to software. It is based on three layers: process, methods, and tools. Software Engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Software Engineering methods provide the technical "how to's" for building software. Methods encompass a broad array of tasks that include requirements analysis, design, implementation, testing, and maintenance. Finally, Software Engineering tools provide automated or semi-automated support for the development process and the methods used in Software Engineering (Pressman, 2014).

Software Engineering defines an abstract representation of a process methodology, known as the process model. Each methodology constitutes a framework used to structure, plan, and control the process of developing a system. Waterfall and Agile represent process models. They don't specify how to do things, but they outline the types of activities, which are done. For example, Waterfall identifies the phases that a

project goes through without saying what artifacts to produce or what tools to use. This is also the case with Agile model, which defines core values in the form of the Agile manifesto, time-boxed iterations, and continuous response to change, but it doesn't say how long your iterations should be or how your response to changes should be.

On the other hand, a software process methodology is a specific way of conducting a software project, like the Rational Unified Process and Scrum. They define exactly what, when, and/or how various artifacts are produced. They might not be entirely explicit with all regards. For example, Scrum doesn't identify what documents to produce or not to produce, since its focus is on delivering value to the customer. But they define, in some way, the actions that members of the project team must follow.

Many models and methodologies of Software Engineering have been theorized. Early examples of the software development methodologies were the Waterfall model, Spiral, and Prototyping models. It is worth noting that these models have emerged out of need to control the development and carry it in a systematic manner. Some of them have emerged to help overcome the challenges that earlier models couldn't; such as, rapid delivery, or excess documentation overhead.

It is important to stress on a pivotal point. There is no such this as the "one for all" solution when it comes to software development. Each model can be refined and customized to meet the needs and standards of the development team. In some cases, a model can be a hybrid model of many, or a tailored cut of one model. It just needs to meet the standards of the management and the requirements of the client in order for it to work.

1.1 A General Overview for Software Engineering

According to Pressman in his book in 2014, Software Engineering moved into its fourth decade in the 90s. Also, throughout the industry, "software engineer" has replaced "programmer" as the job title of preference. Software process models, Software Engineering methods, and software tools have been adopted successfully across a broad spectrum of industry applications (Pressman, 2014). The history of

Software Engineering begins from a traditional procedural approach that is the first methodology in the software world. Then object-oriented and component based approaches came into this world and brought many new useful features.

Traditional methods for the analysis and design of computer-based systems have now been promoted for more than 40 years. Many of the organizations, which deal with the design and construction of computer-based systems apply traditional system development methods with varying degrees of success and there are still a great many system developers who do not use traditional methods at all, though some of these have attempted to introduce methods into their work practices. Where some organizations may have found increased benefits from the adoption of such methods, others have met only with dismay and failure (Kautz, 1999). Traditional methods relied on defining the system as two separate entities: Data and Function entities. These two entities would go on being traditional, planned, and controlled separately during the process of development. This would result in a solution that might access duplicates of data in different functions in a function oriented design, or a data that use duplicate functions in a data oriented design.

This dispersion of data and function in traditional design created the need for a new paradigm in Software Engineering resembling the new –back then- Object Oriented paradigm in implementation languages. This paradigm compresses the data and function in a single entity called objects. It defines the behavior of these objects internally, and in relation to other objects or users in the developed system.

1.2 The Aim of the Project

The main aim of this research is to focus on identifying the steps in shifting from a traditional procedural design to an Object Oriented (O-O) design. The steps are not meant only for Software Engineering professionals, but also for programmers with little Software Engineering principles knowledge. The goal is not to fully redesign the system, but to at least have the minimum requirements, which are the objects.

Also, this project aims to apply the proposed approach on a Social Network Analysis system in an Omani organization referred to as "Markaz". This could be achieved by

concerned with redesigning the application.The thesis at first provides background for O-O software engineering first and then it explains the implementation. The proposed detailed steps will serve into identifying the objects needed for redesigning the system into the O-O paradigm. The future work would take that a step forward to derive more of the current system in regards of diagrams and other needed documentation instead of creating it from scratch.

As a case study, only one module of Markaz's SNA tool will be redesigned using the O-O detailed steps. Since Markaz environment is a private and closed one, this research skips searching for ready-made objects. Therefore, each required object is developed by using .NET frame work 4.0 In addition; GUIs (Graphical User Interface) of each various system are developed using this approach and the MS Visual Studio 2010 IDE and MS Expressions Blend. Meanwhile, although hardware and software architectures of Markaz are described in the case study chapters such as Chapter 4 and 5, specific vendors, providing technology for Markaz, software codes and database schemas are not mentioned for security reasons.

The phases in the research methodology are shown in Table 1.1

Table 1.1 Research Methodology phases

Phase	Steps	Duration	Notes
Proposal	N/A	17-20 July 2012	
Literature Review	1.Search for literature related to Procedural Design 2.Search for literature related to O-O desing	21 July 2012 – 21 November 2012	
Literature Analysis	N/A	1 December 2012 -1 March 2013	
Problem Definition	1.Define problem scope. 2.Define problem parts.	2 March 2013 – 1 May 2013	
Develop Solution	1.Define Solution parameters. 2.Develop solution idea and approach	5 May 2013 – 1 September 2013	
Case Study	1.Apply solution on case study 2.Evaluate Findings	15 September 2013 – 5January 2014	
Write up and validation	N/A	10 January 2014 –15 May 2014	

1.5 Organization of the Thesis

Chapter 2 explains Software Engineering approaches in general. Chapter 2 also describes the various approaches and process models for all these approaches are described. Chapter 3 describes O-O development and its basic concepts, tools, modeling languages, technologies, and promises. Chapter 4 describes the current structure of Markaz SNA System in terms of both hardware and software architectures. Chapter 5 describes each step of implementation of the personal module in Markaz's SNA system. Chapter 6 is reserved for Conclusions and Suggestions in which has some suggestions for future of Markaz's SNA system and some inferences about Object Oriented software engineering studies.

العنوان:	Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design
المؤلف الرئيسي:	الشندودي، أحمد بن سيف بن محمد
مؤلفين آخرين:	Al Khanjari, Zuhoor Abdullah Salim(Advisor)
التاريخ الميلادي:	2014
موقع:	مسقط
الصفحات:	1 - 95
رقم MD:	972147
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة السلطان قابوس
الكلية:	كلية العلوم
الدولة:	عمان
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، تصميم البرمجيات، المنهجية التسلسلية التقليدية، برمجة تحليل الشبكات الاجتماعية، إعادة التصميم، هندسة البرمجيات الموضوعية
رابط:	https://search.mandumah.com/Record/972147

Chapter 2: Software Engineering Approaches

This Chapter describes in detail three major approaches in Software Engineering such as traditional, component-based, and object-oriented approaches. It describes these approaches in details and some of the process models used for development.

2.1 Various Approaches for Software Engineering

2.1.1 Traditional Approach

This approach contains basic steps of a software development process such as analysis, design, implementation, testing, and maintenance. This thesis focuses on only analysis and design phases and their detailed steps. Each of the steps of the analysis phase (Pressman, 2014) provides information that is required to create a design architecture. The flow of information during software design is illustrated in Figure 2.1. For specifying software, this approach offers some variety of elements such as a data dictionary, data flow diagrams, state transition diagrams, entity-relationship diagrams, process specifications, control specifications, and data object descriptions for analysis phase. The design phase produces a data design, an architectural design, an interface design, and a procedural design with the help of various methods and techniques such as transaction mapping and transform mapping for architectural design and structured programming, graphical design notation, tabular design notation, and program design language for procedural design. Brief descriptions of some favorite elements of analysis and design models are mentioned below (Brooks, 1987):

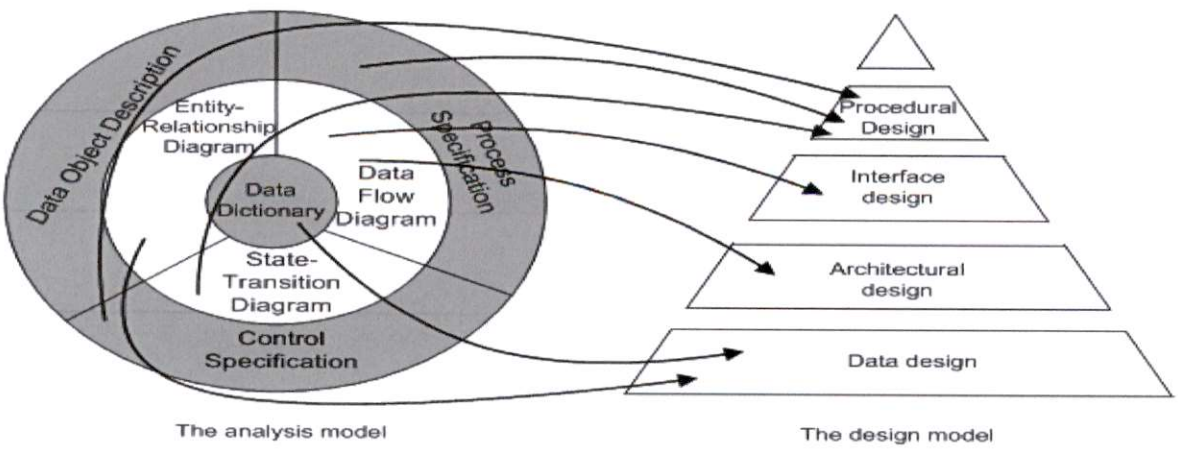

Figure 2.1 Mapping between design steps and related techniques

Data flow diagrams represent the transformations of data as it flows through a system and are the focus of SA/SD (Structured Analysis/ Structured Design). A data flow diagram consists of processes, data flows, actors, and data stores. Starting from the top-level data flow diagram, SA/SD recursively divides complex processes into sub diagrams, until many small processes are left that are easy to implement. When the resulting processes are simple enough, the decomposition stops, and a process specification is written for each lowest-level process. Process specifications may be expressed with decision tables, pseudo code, or other techniques.

The data dictionary contains details that cannot be included from data flow diagrams. The data dictionary defines data flows and data stores and meaning of various names. State transition diagrams illustrate time dependent behavior. Most state transition diagrams describe control processes or timing of function execution and data access triggered by events.

Entity-relationship (ER) diagrams highlight relationships between data stores that otherwise would only be seen in the process specifications. Each ER data element corresponds to one data flow diagram data store. In design phase, the most favorite technique is structured programming to produce procedural design. It is performed by languages such as Pascal, Ada and C. The broad definition of structured programming refers to any software development technique that includes structured design and results in the development of a structured program. Structured programming allows programs to be broken down into blocks or procedures, which can be written without detailed knowledge of the inner workings of other blocks. Thus allowing a top-down design approach or stepwise refinement (Brooks, 1987). Large-scale systems, built using this approach, are often deployed on only mainframes and minis. They feature as mainframe-based or other non-relational database systems. Therefore, both feeling the heat of competition, and simply looking for ways to improve software development can be the reason for moving into object-oriented approach in industry (Kautz, 1999).

2.1.2 Component-Based Approach

This approach is expected to revolutionize the development and maintenance of software systems. The Gartner Group, for example, estimates that "... by 2003, 70% of new applications will be deployed as a combination of pre-assembled and newly created components integrated to form complex business systems." The resulting increase in reuse should dramatically improve time-to-market, software lifecycle costs, and quality (Atkinson et al., 2005). In this section the emphasis is how this approach and its seed, CBD, are taken from previous approaches and intermediary approaches such as distributed objects and distributed systems. Figure 2.2 depicts the transformation that occurs after object-oriented approach. With distributed object approach extends the object-oriented approach with the ability to call objects across address space boundaries, typically using an "object request broker" capability. The distributed system approach is a development approach for building systems that are distributed, and are often represented as multi-tier.

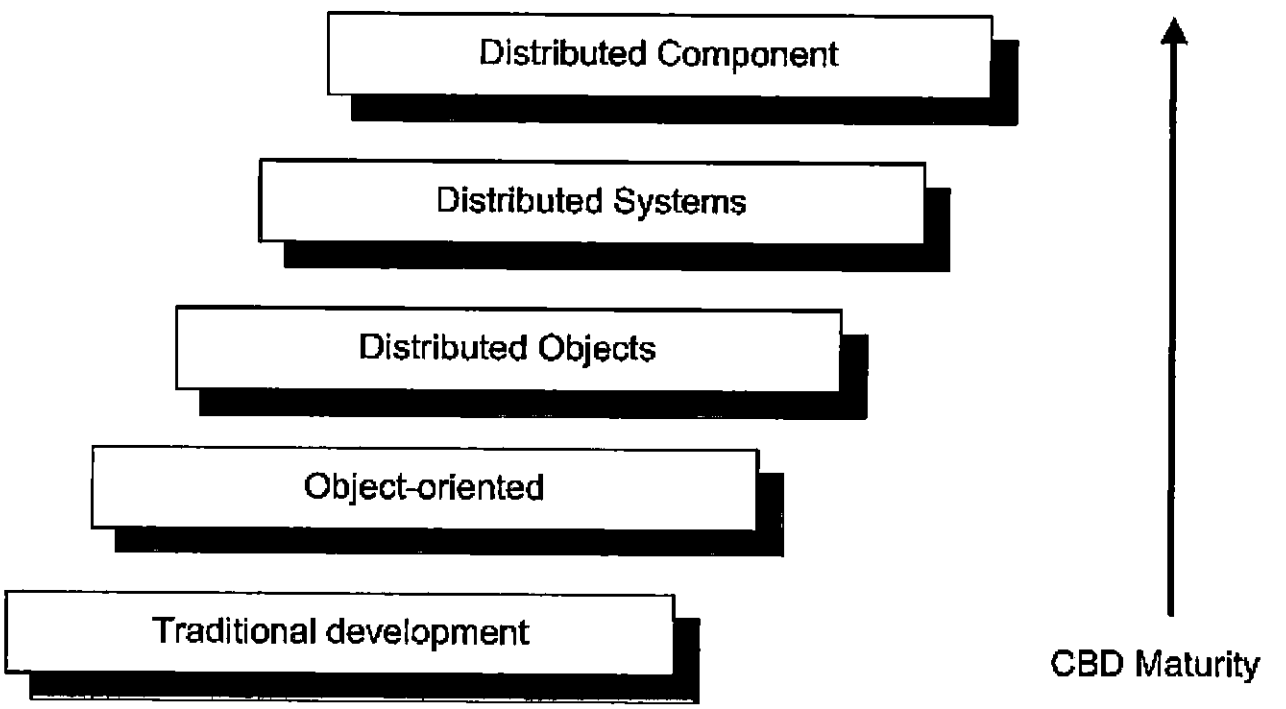

Figure 2.2 CBD maturity phases

Many companies today claim to be following component-based development when what they are really following is distributed system development, or using some kind of distributed object technology. While this can deliver significant benefits, such as allowing the technical bridging of heterogeneous systems, it does not decrease the cost of development. It is slightly addressed by using object-oriented techniques but not enough to make a big difference. At this point, distributed component approach is embraced in industry to reap the desired benefits, often looked for by a software development organization. This uses current build-time and run-time technologies such as Enterprise Java Beans that is attempted to reduce cost and development time for distributed systems. It becomes apparent that what is needed is something that addresses both the challenge of distributed systems interoperability and the challenge of how to build individual systems that can be treated as atomic units and can easily be made to cooperate with each other (Kautz, 1999).

Despite the industrial evolution, mentioned above, component-based technology introduces abstraction and lower-level mechanisms but has to be orchestrated into a comprehensive Software Engineering process (Dorgu and Tanik, 2003).

2.1.3 Object-Oriented Approach

The steps of software development mentioned above are common for all Software Engineering approaches. Therefore, analysis and design phases are inevitable for object-oriented approach, as well. In this approach, design is divided into four different steps as illustrated in Table 2.1.

Table 2.1 Analysis and design phases for Object-Oriented Approach

Phase	Techniques	Key Deliverables
Analysis	• Collaboration Diagrams • Class and Object models • Analysis Modeling	Analysis Models
System Design	• Deployment Modeling • Component Modeling • Package Modeling • Architectural Modeling	Overview design and Implementation architecture
Class Design	• Class and Object Modeling • Interaction Modeling • State Modeling • Design Patterns	Design Models

Object-oriented approach promises a way for implementing real-world problems to abstractions from which software can be developed effectively. It is a sensible strategy to transform the development of a large, complex super-system into the development of a set of less complicated sub-systems. Object-orientation offers conceptual structures that support this sub-division. Object-orientation also aims to provide a mechanism to support the reuse of program code, design, and analysis models (Hill and McRobb, 2002).

This approach uses classes and objects as the main constructs from analysis to implementation. It normally involves using an Object-Oriented language such as C++ or Java that provides (build-time) encapsulation, inheritance and polymorphism, and the ability for objects to invoke each other within the same address space. This last point is an important constraint when it comes to distributed systems.

UML (Unified Modeling Language) contains a number of concepts that are used to describe systems and the ways in which the systems can be broken down and modeled. The UML Specification defines the terms class and object as follows (Hill and McRobb, 2002):

- A class is a description of a set of objects that share the same attributes, operations, methods, relationships and semantics. Moreover, the purpose of a

class is to declare a collection of methods, operations and attributes that fully describe the structure and behavior of objects.

- An object is "an instance that originates from a class. It is structured and behaves according to its class.' Interface is another important construct defined as a group of externally visible operations. The interface contains no internal structure; it has no attributes, no associations, and only abstract operations."

In object-orientation, three main principles are important. Encapsulation, which is also known as information hiding, provides the internal implementation of the object without requiring any change to the application that uses it. The ability of one class of objects to inherit some of its properties or methods from an ancestor class is named inheritance in object technology. Polymorphism is producing various results for a generalized request based on the object that is sent to.

In Object-Oriented Approach objects of software can be (Brooks, 1987):

- External entities: printer, user, sensor
- Things: reports, displays
- Occurrences or events: alarm, interrupt
- Roles: manager, engineer, salesperson
- Organizational unit: team, division
- Places: manufacturing floor
- Structures: employee record

In this approach de facto standard notation (Hill and McRobb, 2002), UML, reveals analysis and design phases of software development.

Use cases specify the functionality that the system will offer from the users' perspective. They are used to document the scope of the system and the developer's understanding of what it is that the users require.

Classes might interact to deliver the functionality of the use case and the set of classes is known as collaboration. Collaborations can also be represented in various ways that

reveal their internal details. The collaboration diagram is probably the most useful one. In addition to collaboration diagram, class diagram also represents these collaborations in detail. The class diagram is fundamental to object-oriented analysis. Through successive iterations, it provides both a high-level basis for systems architecture; and a low-level basis for the allocation of data and behavior to individual classes object instances.

A sequence diagram shows an interaction between objects arranged in a time sequence. Sequence diagrams can be drawn at different levels of detail and to meet different purposes at several stages in the development life cycle.

In Object-Orientation, some algorithmic approaches are used, that is, Structured English, Pseudo-code, and Activity diagrams.

The UML specification defines the state as a condition during the life of an object or an interaction during which it satisfies some condition, performs some actions, or waits for some events. State charts describe that apparently.

Class Responsibility Collaboration (CRC) cards provide an effective technique for exploring the possible ways of allocating responsibilities to classes and the collaborations that are necessary to fulfill the responsibilities. CRC cards can be used at several different stages of a project different purpose.

The Object-Oriented Approach allows development-time reuse (Kautz, 1999), meaning that compared to previous approaches, it enhances developers' ability to build software that reuses pieces designed and coded by other developers. However, this level of reuse has clearly fallen short of addressing the needs of large-scale development.

The Object-Oriented Approach has facilitated development of large-scale projects, but it has been mainly limited to the use of one technology on one platform. It has not really developed technologies and models for interoperability, but rather has been mostly focused on the development of one single system. In the 80s, neither interoperability nor portability was a major issue. The need for open systems was

already there, but the technology to resolve the issues was not. This made it difficult to address portability and interoperability (Kautz, 1999).

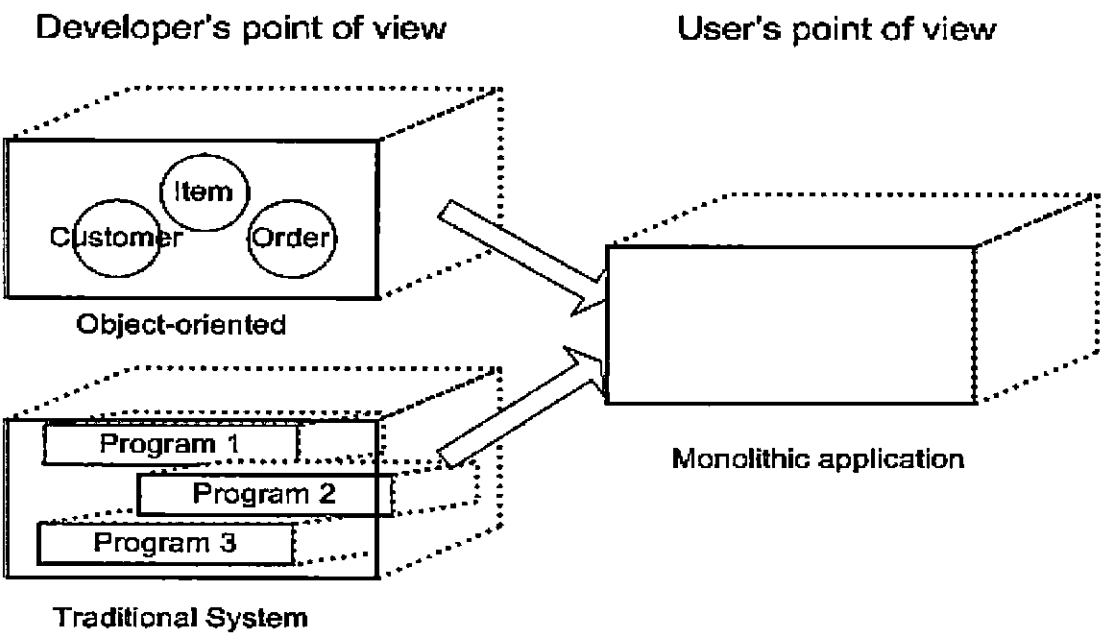

Figure 2.3 Object-Oriented Application

Besides the insufficiencies of two features of object-oriented, reusability and interoperability, their payloads are described above. The reason for these shortcomings can be explained as the Object-Oriented Approach changed the way, applications were built, but it did not change the nature of the applications themselves. This is illustrated in Figure 2.3. In structured approach, the end user would receive a monolithic application. When it is developed using Object-Oriented Approach, the end user would still receive a monolithic application. Shortly, the object-oriented approach is at the service of the functional developer, not the end user (Kautz, 1999).

2.2 Process Models

A software system, either small or large scale, is an uncertain concept at the beginning and therefore needs to be analyzed, designed and implemented. This completes the development but it is not over in terms of operation of software. Maintenance is required after all steps to keep software alive. All of these steps are called software

process model. There are various process models called life cycle (Schach, 1999), which are developed for different circumstances. Traditional process models are the most mature ones, and mostly suit structured approach. However, they are also used for recent approaches such as Object-Oriented and component-based approaches in various phases and in somehow modified form. Object-Oriented Approach does not offer any original process model, which provides for its needs completely. Therefore, this approach complies with traditional models and their appropriate combinations. However, it is different when it comes to Component-Based Approach because it changes the nature of software (Kautz, 1999).

2.2.1 Traditional Process Models

Since each product shows different characteristics at development stage, various project life cycles can be applied to computerize systems development. Some of them will spend years in the logical phase, current hardware may just not be fast 14 enough for the product, or current users may not be capable of this new system and computerized background. Some of them can also be quickly designed and implemented and then many years are spent for modifications to meet the users' changing needs in the maintenance phase. Table 2.2 shows the most used phases in process models and their deliverables. These output deliverables are useful and required for their next steps as their inputs.

Table 2.2 Life cycle deliverables

Phase	Output deliverables
Requirements Analysis	Requirements specification Functional specification Acceptance test specification
Design	Software Architecture specification System test specification Design specification Sub-system test specification Unit test specification
Construction	Program code
Testing	Unit test report Sub-system test report System test report Acceptance test report Completed system
Installation	Installed system
Maintenance	System changes Change Report

A number of different life-cycle models will be described and three of them are most widely used; Waterfall with iteration, Rapid Prototyping and the Spiral Model which has received considerable attention recently.

2.2.1.1 Waterfall Model

Until the early 1980s, the Waterfall model was the only widely accepted life-cycle model. This approach offers a sequential mechanism among the steps of development process. It seems difficult to return to an earlier phase once it is completed like a real Waterfall (see in Figure 2.4).

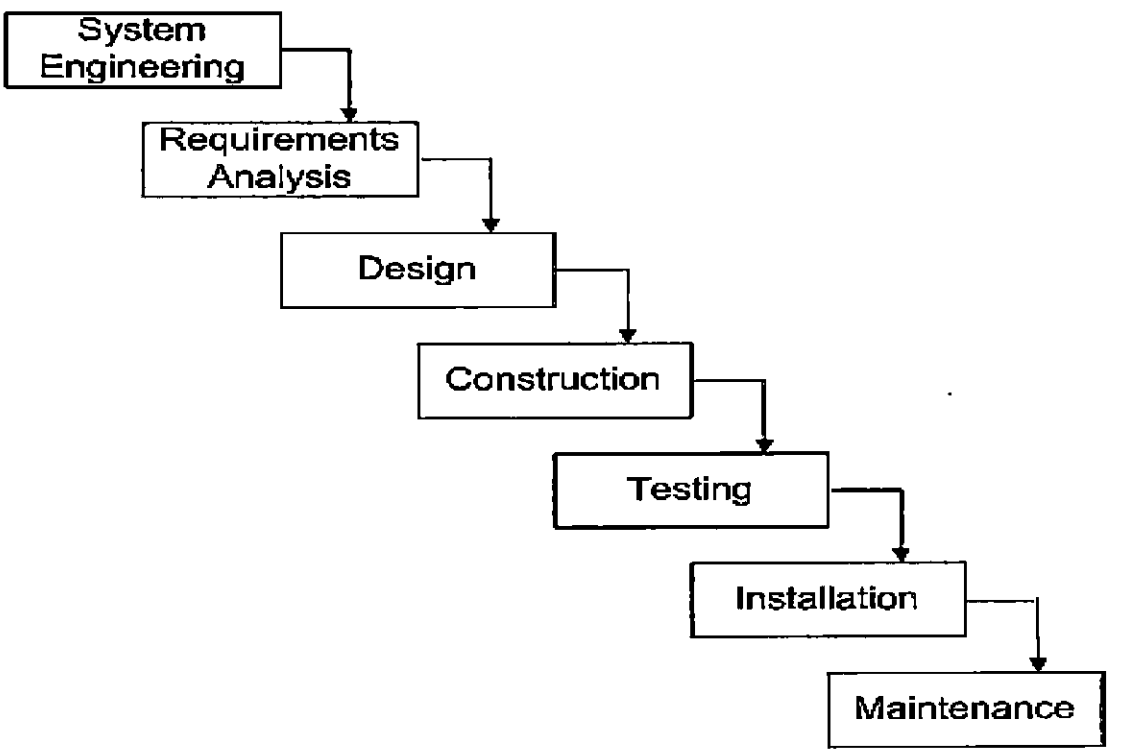

Figure 2.4 Waterfall Process Model

In the real world, there can be no high quality software that has been developed with this process model. Because when one of the phases of these process fails, that software is inevitable regarding the error prone in order not to go back with feedback. For that reason iteration is inevitable.

The Waterfall tends to be irresponsive to changes in client requirements or technology during the project. Once they have been made, architectural decisions are difficult to change.

In contrast to Waterfall, the diagram in Figure 2.5 shows possible paths for iteration within the Waterfall but these iterations can be very costly. Iterative Waterfall has the following advantages (Hill and McRobb, 2002):

- The tasks in a particular stage may be assigned to specialized teams. For example, some teams may specialize in analysis, others in design and yet others in testing.
- The progress of the project can be evaluated at the end of each phase and an assessment is made as to whether the project should proceed or not.
- The controlled approach can be effective for managing the risks on large projects with potentially high levels of risk.

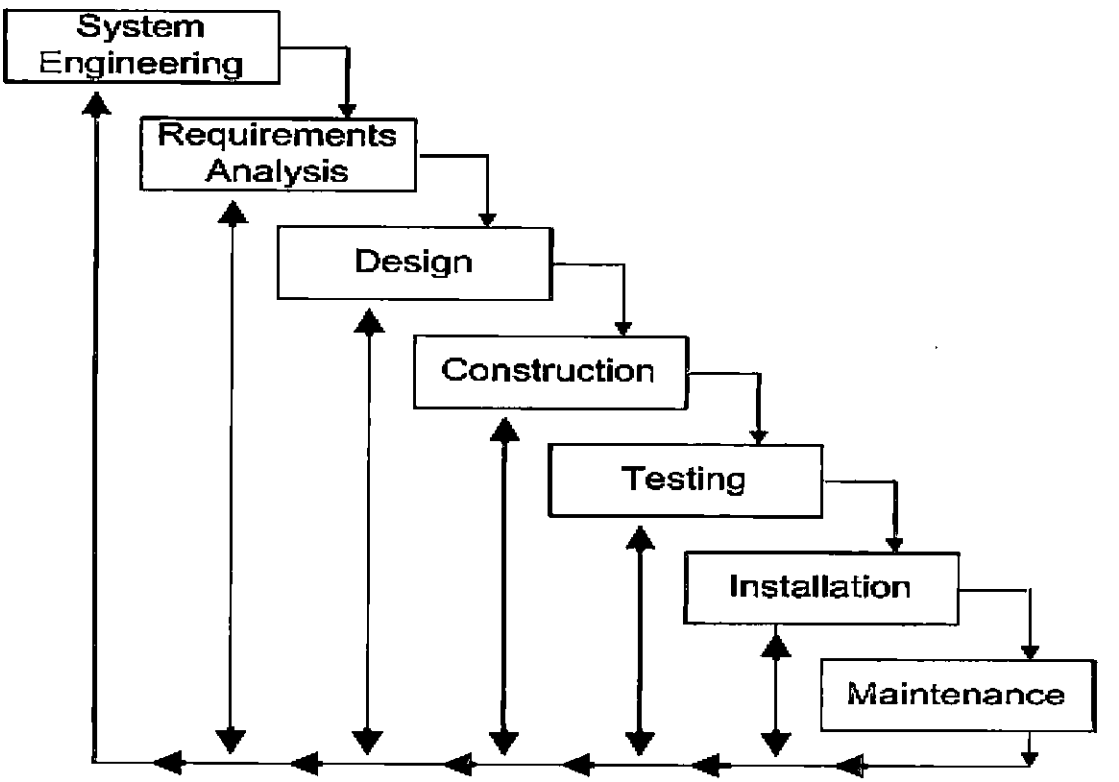

Figure 2.5 Waterfall with Post Installation check Process Model (Pressman, 2014)

Some authors (Schach, 1999) consider that testing is not a separate phase to be performed only after the product has been constructed; it is to be performed at the end of each phase.

2.2.1.2 Build-and-Fix Model

This approach may work well on short programming exercises of 100 and 200 lines long. In this approach the product is constructed without specifications or any attempt at design. The developers simply build a product that will be tested as many times as necessary to satisfy the client. Therefore, the build-and-fix model is very unsatisfactory for products of any reasonable size.

2.2.1.3 Rapid Prototyping Model

Since users operate the system and perform fixed processes, they cannot imagine the whole system most of the time and they need to be conveyed into a working system. Therefore, it is a problem not to entirely define the requirements in terms of development processing.

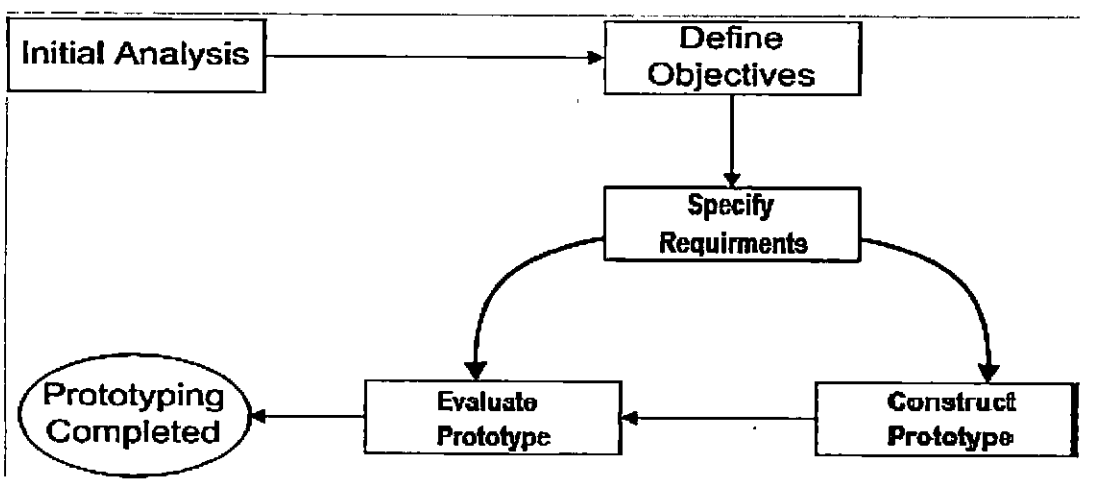

Figure 2.6 Rapid Prototyping Model (Pressman, 2014)

As well as being used to investigate the requirements, prototyping might also be used to discover the most suitable form of user interfaces. Some advantages and disadvantages are described below (Hill and McRobb, 2002):

- Early demonstrations of system functionality help identify any misunderstandings between developer and client;
- Client requirements that have been missed are identified;

- Difficulties in the user interface can be identified;
- The feasibility and usefulness of the system can be tested, even though, by its very nature, the prototype is incomplete.

Disadvantages:

- The client may perceive the prototype as part of the final system, or may not understand the effort that will be required to produce a working production system, and also may expect delivery soon;
- The prototype may divert attention from functional to solely interface issues;
- Prototyping requires significant user involvement;
- Managing the prototyping life cycle requires careful decision-making.

A solution (Schach, 1999) is offered that combining the two approaches, Waterfall and Prototyping. Rapid Prototyping can be used as a requirement analysis technique. In other words, the first step is to build a Rapid Prototype in order to determine the client's real needs and then to use that Rapid Prototype as the input to the Waterfall Model.

This approach also has a useful side effect (Schach, 1999). Some organizations are reluctant to use the Rapid Prototyping approach because of the risks involved in using any new technology. Introducing Rapid Prototyping into the organization as a front end to the Waterfall Model will give management opportunity to assess the technique while minimizing the associated risk.

2.2.1.4 Incremental Model

Models, described above, all produce a complete product, which satisfies the clients. If those models are used correctly, they result in such products that will have been entirely tested and the clients should be so confident that these products could be used for the purposes, which they wish.

In the incremental model, operational products are delivered at each stage. Software is not written, it is built (Schach, 1999). The complete product is divided into parts, and the developer delivers the product part by part. A typical product consists of 10 to 50

builds. At each stage, the client has an operational quality product that does a portion of what is required; from delivery of the first build, the client is able to do useful work. With the incremental model, portions of the total product might be available within weeks, whereas the client generally waits months or years to receive a product built using the Waterfall or Rapid Prototyping models. Another advantage of the Incremental Model is that it reduces the traumatic effect of imposing a completely new product on the client organization. From the client's financial viewpoint, phased delivery does not require a large capital. Figure 2.7 shows each phase of Incremental Process Model (Schach, 1999).

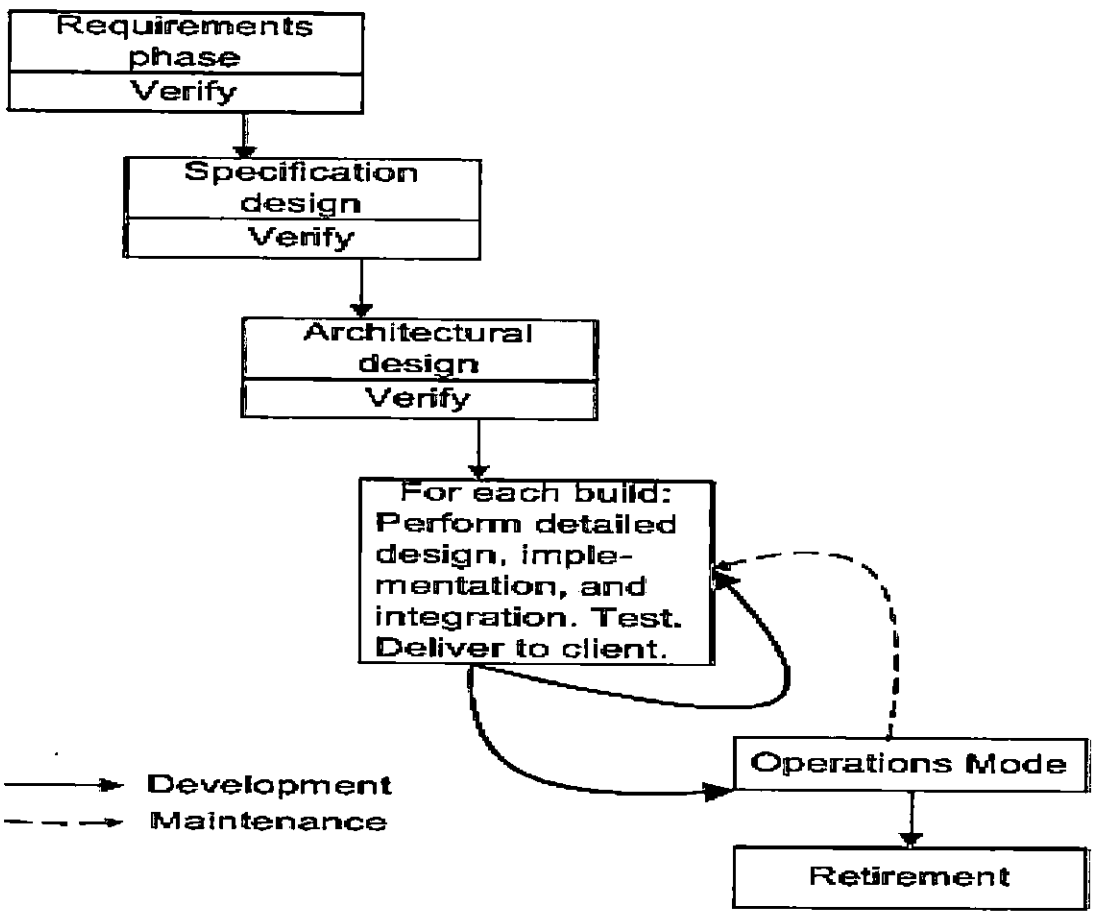

Figure 2.7 Incremental Model (Pressman, 2014)

A difficulty with the incremental model is that each additional build somehow has to be incorporated into the existing structure without destroying what has been built up to that date.

2.2.1.5 Spiral Model

Although the models, described above, involve reducing the impact of risk, they do not base their concepts on that risk factor. The idea of minimizing risk via the use of prototypes and other means is the concept underlying the Spiral Model. A simple way of looking at this life cycle model is a Waterfall Model with each phase preceded by risk analysis. Before commencing each phase an attempt is made to control (or resolve) the risks as shown in Figure 2.8. If it is impossible to resolve all the significant risks at that stage, then the project is immediately terminated.

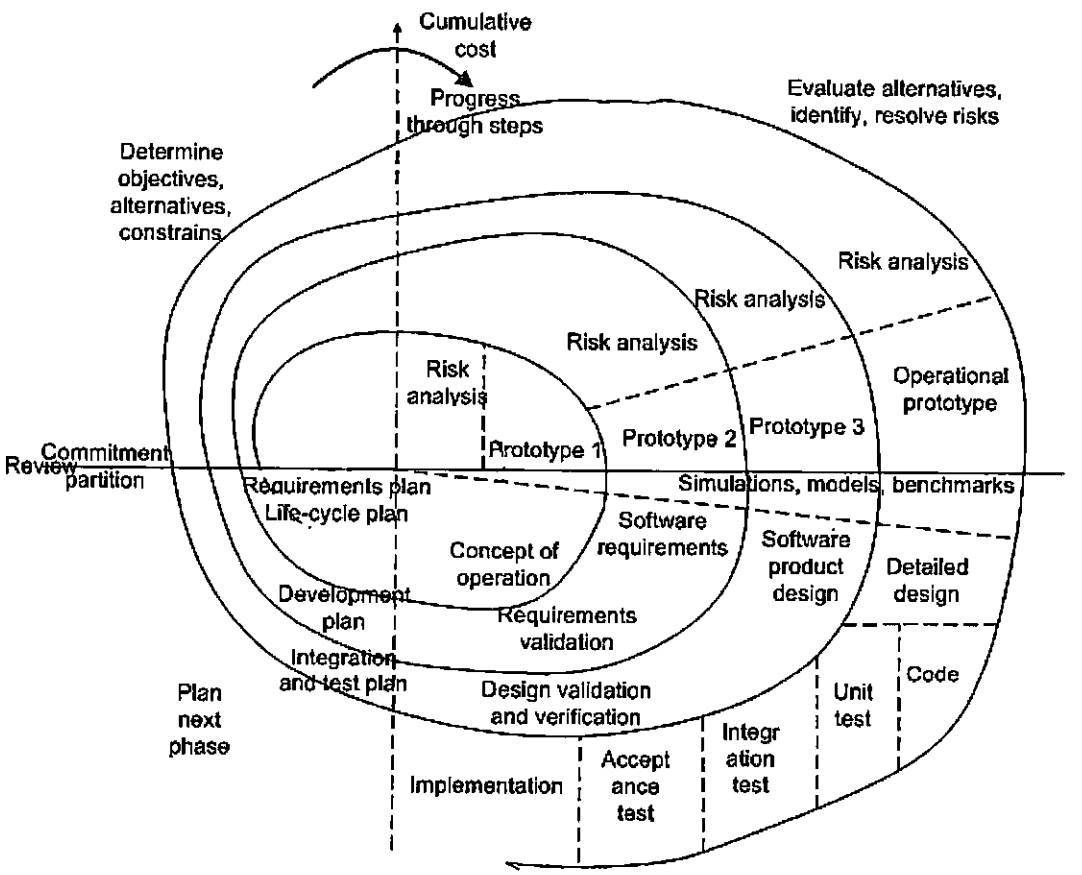

Figure 2.8 Spiral Model

The strength of this model comes from the emphasis on alternatives and constraints, supporting the reuse of existing software and the incorporation of software quality as a specific objective. In addition, a common problem in software development is to determine when the products of a specific phase have been adequately tested (Schach, 1999).

Sometimes it is not suitable to implement this model because this model is only used for large-scale projects. It makes sense when performing risk analysis in terms of cost of time and finance.

2.2.2 Component Based Models

As mentioned before Object-Oriented process models are not widely used since there is no change in nature of software for that approach. However, it is different for Component-Based Software Engineering Approach because it changes the nature of software, so there is a need for some original phases stem from the architecture of components.

One more definition must be added at this point to differentiate software development and the component life cycle. In a traditional software development, process model developers are often analysts, designers, and developers. A project has a well-defined beginning when requirements are elicited; and a well-defined ending when the final software system is delivered. However, component production is different. Considerably more time is devoted to business rules, business process modeling, analysis, and design. Much less time is spent in development, while testing occurs throughout the process. Following definition explains this software process type in general (Kroll and Kruchten, 2003):

The Component-Based Software Life Cycle (CSLC) is the life cycle process for a software component with an emphasis on business rules, business process modeling, design, construction, continuous testing, deployment, evolution, and subsequent reuse and maintenance.In general, analysis and design phases for Component-Based Process Models take more time than traditional ones take.

The questions of how to identify model and specify components, how to follow a Component-Based development process in a systematic and consistent manner, and how to assembly formally specified components into the Component-Based system architecture are not properly addressed yet. Though there are some studies in this era, in this study, only two of them are selected, one of them being from Stojanovic (Taft, 2002) and another one, which has been worked on in more detail, is from Ali H. Dogru and Murat M. (Dogru and Tanik, 2003).

2.2.2.1 Stojanovic Process Model

A Component-Oriented development Process Model, has been introduced by Stojanovic (Taft, 2002), focusing on the component concept from business requirements to implementation. This process will be called by its owner's name in this study. The phases of requirements, analysis, design and implementation in a traditional development process has been substituted by service requirements, component identification, component specification, component assembly and deployment. After the components of the system are fully specified, a decision can be made to build components, wrap existing assets, buy COTS (Commercial Off-the-Shelf) components or invoke web services over the Internet.

2.2.2.2 COSE Process Model

Dogru and Tanik emphasize that Component-Based Methodology is immature: "Software development methodologies began with traditional approaches that followed the waterfall process model. They then moved toward object-oriented abstractions, which were finally supported by object-oriented methodologies. Now, component-based technology introduces abstraction and lower-level mechanisms but has to be orchestrated into a comprehensive software engineering process." (Dogru and Tanik, 2003)

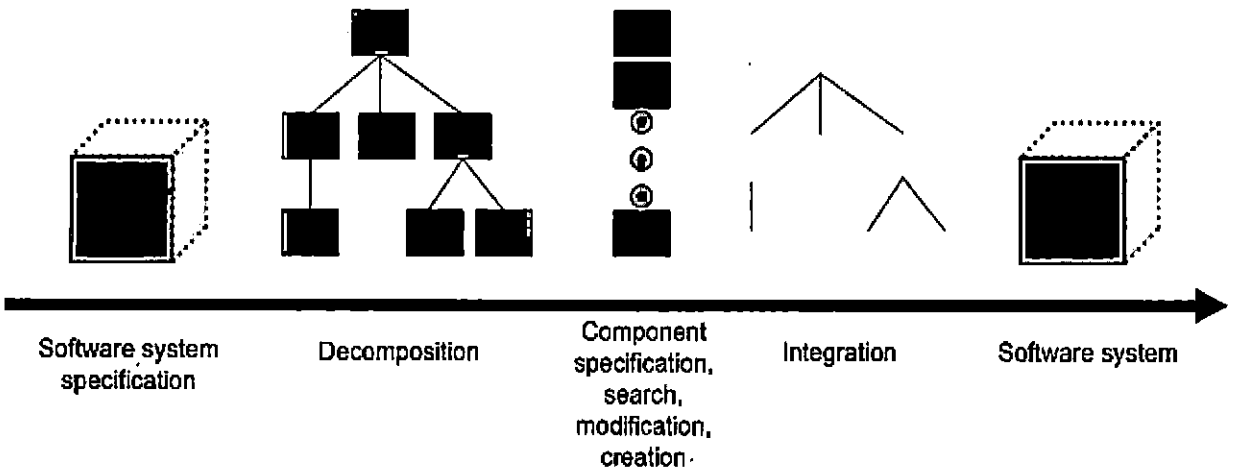

Figure 2.9 COSE Process Model (Dogru and Tanik, 2003)

Figure 2.9 illustrates steps of COSE Process Model in general. However, it is explained in more detail with a study of Vedat Bayar, which is summarized here to reveal COSE Process Model and its details; COSE Process Model building activity starts top-down to introduce the building blocks of the system. As the activity continues towards lower granularity blocks, interfaces between the blocks are also defined. At an arrived level where the module is expected to correspond to a component, a temporary bottom-up approach can be taken; if desired capability can only be achieved by a set of components, their integration into a super-component should be carried out.

COSE process model consists of four main phases and a system test phase:

- System specification
- System decomposition
- Component Specification, search, modification, creation
- Integration

COSE Process Model starts with system specification. Problem is specified and understood, then system high-level requirements are stated and a preliminary search for existing components is conducted in system specification phase. Problem and problem domain knowledge are input to this phase. System high level functional and

non-functional requirements and domain related existing component specification documents are the output of this phase.

System is analyzed and decomposed in the system decomposition phase. Functional requirements are detailed and required components and their specifications are stated.

Components that are going to be implemented in the system are specified and developed in component specification phase, either by using existing components or developing new components.

System decomposition and component specification phases are not independent phases. When the system decomposition comes to a stable level, component specification phase can begin. Component specification begins with searching for existing components and continues with the evaluation of located components. If the decomposition level is not low enough to find and evaluate needed components, then system is decomposed further until there is no further meaningful decomposition. The outcome of system decomposition and component specification phases is components and component specifications. The flow of the COSE Process Model continues with integration of the components and testing of the whole system.

2.2.3 Object Oriented Process Models

2.2.3.1 Fountain Model

The Fountain model outlines the general characteristics of the systems level perception of an object oriented development. There is a high degree of merging in the analysis, design, implementation, and unit testing phases. Moving through a number of steps, falling back one or more steps and performing repeatedly, is a far more flexible approach than the one proposed by Waterfall Model. It follows a bottom up approach, which starts from the solution. If there is an existing solution, that solution is studied first and the necessary details are identified and organized in a suitable manner. For a problem not having a solution, the domain experts (i.e., experts who are capable of providing useful information and future requirements) are consulted with the conventional solution to start with. Since the software is developed by analyzing the solution first, this approach is known as bottom-up approach (Buyya, 2009).

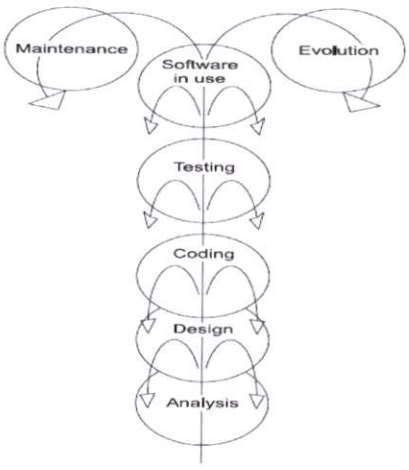

Figure 2.10 Fountain Model

The Fountain Model provided better solutions for complex problems compared to the top-down approach followed in the Waterfall Model. The procedural and structured programming languages were found unsuitable for the bottom-up approach because a change in requirement, analysis, or design phase can cause the programming to start from the beginning once again. They lack flexibility, modifiability, and software component reuse.

2.2.3.2 Unified Software Development Process

The Unified Process (UP) is a use-case-driven, architecture-centric, iterative and incremental development process framework that leverages the Object Management Group's (OMG) UML and is compliant with the OMG's SPEM. The UP is broadly applicable to different types of software systems, including small-scale and large-scale projects having various degrees of managerial and technical complexity, across different application domains and organizational cultures. (Kroll and Kruchten, 2003)

The unified process divides the project into four phases: Inception, Elaboration, Construction, and Transition.

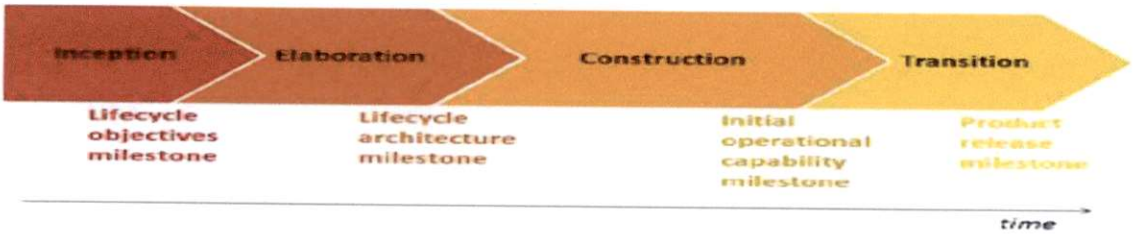

Figure 2.11 Unified Process phases (Kroll and Kruchten, 2003)

Inception is the smallest phase in the project, and ideally it should be quite short. If the Inception Phase is long then it may be an indication of excessive up-front specification, which is contrary to the spirit of the Unified Process. (Kroll and Kruchten, 2003)

In the Elaboration phase the project team is expected to capture a healthy majority of the system requirements. However, the primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture. Common processes undertaken in this phase include the creation of use case diagrams, conceptual diagrams (class diagrams with only basic notation) and package diagrams (architectural diagrams). (Kroll and Kruchten, 2003)

Construction is the largest phase in the project. In this phase the remainder of the system is built on the foundation laid in Elaboration. System features are implemented in a series of short, time boxed iterations. Each iteration results in an executable release of the software. It is customary to write full text use cases during the construction phase and each one becomes the start of a new iteration. (Kroll and Kruchten, 2003)

The final project phase is Transition. In this phase the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations. The Transition phase also includes system conversions and user training (Kroll and Kruchten, 2003).

2.2.3.3 Rational Unified Process

The Rational Unified Process (RUP) is an iterative software development process framework created by the Rational Software Corporation, a division of IBM since 2003. (Taft, 2002) it is based on the Unified Process. (Kroll and Kruchten, 2003)

In each iteration in RUP, the tasks are categorized into nine disciplines. First, there are six engineering disciplines: Business Modeling, requirements, analysis and design, implementation, test, and deploy. Then, there are three supporting disciplines: configuration and Change management, project Management, and environment. (Kroll and Kruchten, 2003).

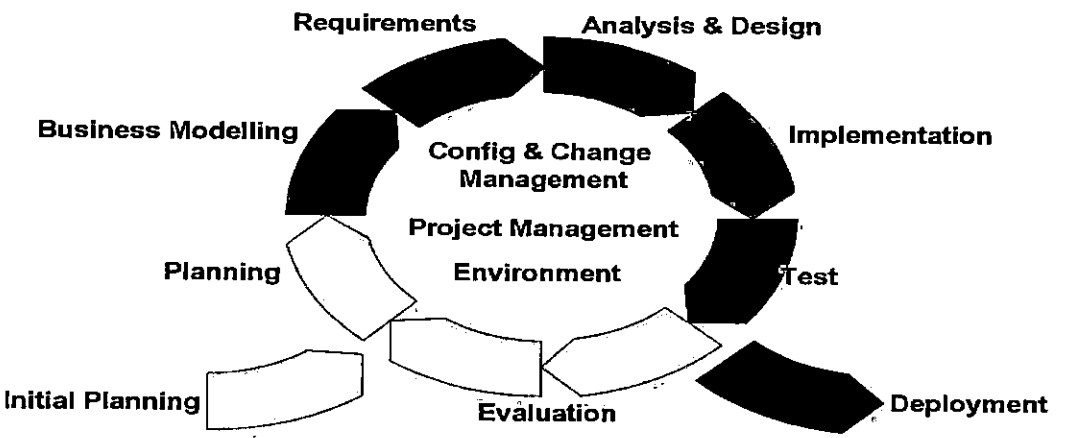

Figure 2.12 Rational Unified Process (Kroll and Kruchten, 2003)

العنوان:	Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design
المؤلف الرئيسي:	الشندودي، أحمد بن سيف بن محمد
مؤلفين آخرين:	Al Khanjari, Zuhoor Abdullah Salim(Advisor)
التاريخ الميلادي:	2014
موقع:	مسقط
الصفحات:	1 - 95
رقم MD:	972147
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة السلطان قابوس
الكلية:	كلية العلوم
الدولة:	عمان
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، تصميم البرمجيات، المنهجية التسلسلية التقليدية، برمجة تحليل الشبكات الاجتماعية، إعادة التصميم، هندسة البرمجيات الموضوعية
رابط:	https://search.mandumah.com/Record/972147

Chapter 3: Literature Review

Systems designed and implemented using the traditional procedural approach, the focus is mainly on the functions of the system in the current problem domain. This approach doesn't take into consideration the impact of the context in which this application or system will run (e.g. organization). This leads to a number of shortcomings of such a design.

In this chapter we will review the literature which discusses these shortcomings of the traditional procedural design approach, and how the Object Oriented approach has worked to solve such issues. This chapter answers through the literature the questions of:

- Where does Object Orientation have to start in a process?

- Why development teams hesitate to adopt object orientation in their development process?

- What quality attribute of the object oriented design is the most important to incorporate?

3.1 Issues with Traditional Procedural design

This section shows some issues with traditional procedural design which are inherited through its nature.

1. Linear view of the system that concentrates solely on functionality: Traditional procedural software design is concerned with capturing, realizing and testing one set of requirements, reflecting a snapshot of one field of application. The field of application may be a specific functionality with special requirements and interface features (Boehm, 1979). It focuses on the local problem domain resulting in a localized solution with a specific use and functionality. Procedural design is based on a linear worldview of the system development process and provides only inadequate techniques for data modeling (Floyd et al, 1986). Hence, data modeling is not the key factor in the design. It concentrates on the use and business logic of the system.

According to Simmons in 1994, Procedural design products are based on analysis of a single system only. They do not take into account the larger problem domain in which this current problem domain exists, which leads to a linear solution to not a solution that fits into a higher hierarchy of problem domains (Simons, 1994). Chen in 2004 gives a hint to the solution for this linear approach by concluding that Procedural design forces the implementation to be visualized as a series of steps where the problem is more of objects in relations with each other. (Chen, 2004)

2. Not adaptable to change: Traditional procedural design localizes the problem domain and as a result it doesn't take into account the changing nature of the larger context of the problem at hand. This leads to a false assumption of requirement stability (Boehm, 1988). In addition to such an assumption, this approach in design doesn't provide a facility to adapt to behavioral change in business logic or functionality (Kahler et al., 2000). This hinders the quality of the solution the traditional procedural design offers. This could be because they are localized and they lack scalability potential and adaptability to change (Mack, 2010). Another problem with procedural design is defects caused by change in data representation. This can propagate and become harder to detect in traditional procedural design since the whole focus is on the functional aspect of the system not the relations between data entities (objects) (Xiong, 2011).

3.2 The Object Oriented solution to these problems

The object Oriented design has solved the issues in traditional procedural design using its own native quality attributes, like Top level approach, reusable objects, and scalable solutions.

1. Top level view of system; hence better understanding of the context of the problem domain: The object oriented design starts at the top down design. Therefore, it takes a wider perspective of the problem domain which results in more general and abstract designs. These designs have more potential in covering a larger problem domain than its traditional procedural counterparts (Meyer, 1988). This approach of a top level design has an impact on how data is modeled and how the data-centered approach -as

opposed to a procedure centered approach- gives a better understanding of the nature of the problem (Barton, 1997).

2. Adaptability to change: The nature of object orientation which concentrates on objects and object behaviors rather than functionality. The strict coupling with the business logic of the system makes the system very change tolerable (Carlsen and Haaks, 1992). It makes object design very adaptable to change (Morch, 1997).

3. Standard and coherent representation of data: Object oriented design provides software with a unifying or coherent form or structure. It forces a standard representation of collection of data used in objects (Perry and Wolf, 1992). OO design uses techniques like inheritance which induces standards that are conceived through the base classes and the hierarchy of inheritance (Craig, 1999). The nature of object orientation and how versatile objects force a coherent standardized design through the system at hand (Eden, 2004).

4. Reusability: One of the quality attributes achieved by object oriented design is reusability. Reusability is evident in the reusable nature of objects, which results in more scalability in the extension of objects using inheritance and encapsulation. Reuse achieved by object orientation is said to dramatically increase productivity (Basili et al., 1996). Reusability of the object oriented design makes it more maintainable, and scalable (Lewallen, 2005). Reusability in the object oriented design approach simplifies the development and maintenance of the program (Mack, 2010).

3.3 Where does Object Orientation have to start in a process?

According to Michael Piefel, object orientation in the development process should start at the design level to fully take advantage of the Object Oriented Programming Languages (OOPL) at the implementation phase, since OOPL is the de-facto programming languages for software development. It is obvious that when the design does not produce an object-oriented structure then the full power of an OOPL will be wasted. It is important to identify classes at the design stage, and not take them just as an implementation issue (Piefel, 1996).

Another note regarding the design phase according to Barton is to start at the level of organization. If in this way you get people over the habit of writing code procedurally, then some very different views on information representations come to view. The design has to take into account the strategically important interests of the organization the software is being developed for (Barton, 1997).

3.4 Why development teams hesitate to adopt object orientation in their development process?

In spite of these potential benefits, many software development groups still hesitate to use Object Oriented Software Engineering (OOSE). Many of these groups are fully occupied by their current software development work and fear being overwhelmed if they introduce a new development paradigm with new knowledge base. Besides, development teams do not know in which phase of the development process to start the object orientation (Fayad et al., 1996).

3.5 What quality attribute of the object oriented approach is the most important to incorporate and when to establish it?

The most valuable quality attribute of the object oriented approach is reusability. Reusability of software is an important prerequisite for cost and time-optimized software development. It influences directly the effort, necessary to build new applications upon existing ones (Narzt et al., 1998). But according to Al-Ahmed to most common reuse in the object oriented is code reuse. This happens in the implementation phase not in the design phase (Al-Ahmed, 2006). Reusability benefits from establishing it in a higher level in the development process especially in the design phase (Batory and Smaragdakis, 1998). Design reusability is more advantageous over code reusability in the object oriented approach because it can be applied in more contexts and so is more common. Also, it is applied earlier in the development process, and so can have a larger impact on a project (Johnson, 2000).

3.6 Related Work

There are approaches for deriving the objects needed for an O-O design from a procedural system. But they do not meet some of the criteria the solution set for it to work for development teams. The following are the two main approaches:

1. Refactoring Anti-Pattern by Alexander Shvets (Shvets, 2013).
2. MetaObject Facility's (MOF) Model Driven Architecture (MDA) by the Object Management Group (OMG, 2014).

3.6.1 Refactoring (BLOB) Anti-Pattern

This anti pattern is used to decompose what is called the "BLOB", "Singleton", or "The God Class". In general, the BLOB is a procedural design even though it may be represented using object notations and implemented in object-oriented languages. The BLOB contains the majority of the processes, and the other objects contain the data. Architectures with the BLOB have separated processes from data. In other words, they are procedural-style rather than object-oriented architectures.

The solution involves a form of refactoring. The key is to move behavior away from the BLOB. It may be appropriate to reallocate behavior to some of the encapsulated data objects in a way that makes these objects more capable and the BLOB less complex.

Although the Refactoring Anti-Pattern is a good solution for decomposing BLOBs, it will not be suitable for the required solution need in our case. This is because of two main reasons:

1. The actual work is done after implementation not during the design. Where the refactoring needs to be on a higher level than implementation to use the underlying Object Oriented Programming Languages to their full Potential.
2. The anti-pattern still localizes the solution to the problem domain. It doesn't encompass an organizational view of the system. Hence, the result is only reusable in similar problem domains not on a larger scale.

3.6.1.1 General Form

The BLOB is found in designs where one class monopolizes the processing, and other classes primarily encapsulate data. This Anti-Pattern is characterized by a class diagram composed of a single complex controller class surrounded by simple data classes. The key problem here is that the majority of the responsibilities are allocated to a single class.

In general, the BLOB is a procedural design even though it may be represented using object notations and implemented in object-oriented languages. A procedural design separates process from data, whereas an object-oriented design merges process and data models, along with partitions.

The BLOB contains the majority of the process, and the other objects contain the data. Architectures with the BLOB have separated process from data; in other words, they are procedural-style rather than object-oriented architectures.

The BLOB can be the result of inappropriate requirements allocation. For example, the BLOB may be a software module that is given responsibilities that overlap most other parts of the system for system control or system management.

The BLOB is also frequently a result of iterative development where proof-of-concept code evolves over time into a prototype, and eventually, a production system. This is often exacerbated by the use of primarily GUI-centric programming languages, such as Visual Basic, that allows a simple form to evolve its functionality, and therefore purpose, during incremental development or prototyping.

The allocation of responsibilities is not repartitioned during system evolution, so that one module becomes predominant. The BLOB is often accompanied by unnecessary code, making it hard to differentiate between the useful functionality of the BLOB Class and no-longer-used code.

3.6.1.2 Symptoms and Consequences

- Single class with a large number of attributes, operations, or both. A class with 60 or more attributes and operations usually indicates the presence of the BLOB.
- A disparate collection of unrelated attributes and operations encapsulated in a single class. An overall lack of cohesiveness of the attributes and operations is typical of the BLOB.
- A single controller class with associated simple, data-object classes.
- An absence of object-oriented design. A program main loop inside the BLOB class associated with relatively passive data objects. The single controller class often nearly encapsulates the applications entire functionality, much like a procedural main program.
- A migrated legacy design that has not been properly refactored into an object-oriented architecture.
- The BLOB compromises the inherent advantages of an object-oriented design. For example, The BLOB limits the ability to modify the system without affecting the functionality of other encapsulated objects. Modifications to the BLOB affect the extensive software within the BLOB's encapsulation. Modifications to other objects in the system are also likely to have impact on the BLOB's software.
- The BLOB Class is typically too complex for reuse and testing. It may be inefficient, or introduce excessive complexity to reuse the BLOB for subsets of its functionality.
- The BLOB Class may be expensive to load into memory, using excessive resources, even for simple operations.

3.6.1.3 Typical Causes

- Lack of an Object-Oriented Architecture. The designers may not have an adequate understanding of object-oriented principles. Alternatively, the team may lack appropriate abstraction skills.
- Lack of (any) architecture. The absence of definition of the system components, their interactions, and the specific use of the selected

programming languages. This allows programs to evolve in an ad hoc fashion because the programming languages are used for other than their intended purposes.

- Lack of architecture enforcement. Sometimes this Anti-Pattern grows accidentally, even after a reasonable architecture was planned. This may be the result of inadequate architectural review as development takes place. This is especially prevalent with development teams new to object orientation.
- Too limited intervention. In iterative projects, developers tend to add little pieces of functionality to existing working classes, rather than add new classes, or revise the class hierarchy for more effective allocation of responsibilities.
- Specified disaster. Sometimes the BLOB results from the way requirements are specified. If the requirements dictate a procedural solution, then architectural commitments may be made during requirements analysis that is difficult to change. Defining system architecture as part of requirements analysis is usually inappropriate, and often leads to the BLOB Anti-Pattern, or worse.

3.6.1.4 Refactoring Solution

As with most of the Anti-Patterns in this section, the solution involves a form of refactoring. The key is to move behavior away from the BLOB. It may be appropriate to reallocate behavior to some of the encapsulated data objects in a way that makes these objects more capable and the BLOB less complex. The method for refactoring responsibilities is described as follows:

1. Identify or categorize related attributes and operations according to contracts. These contracts should be cohesive in that they all directly relate to a common focus, behavior, or function within the overall system. For example, a library system architecture diagram is represented with a potential BLOB class called LIBRARY.

In the example shown in Figure 3.1, the LIBRARY class encapsulates the sum total of all the system's functionality. Therefore, the first step is to identify cohesive sets of

operations and attributes that represent contracts. In this case, we could gather operations related to catalog management, like Sort_Catalog and Search_Catalog. We could also identify all operations and attributes related to individual items, such as Print_Item, Delete_Item, and so on.

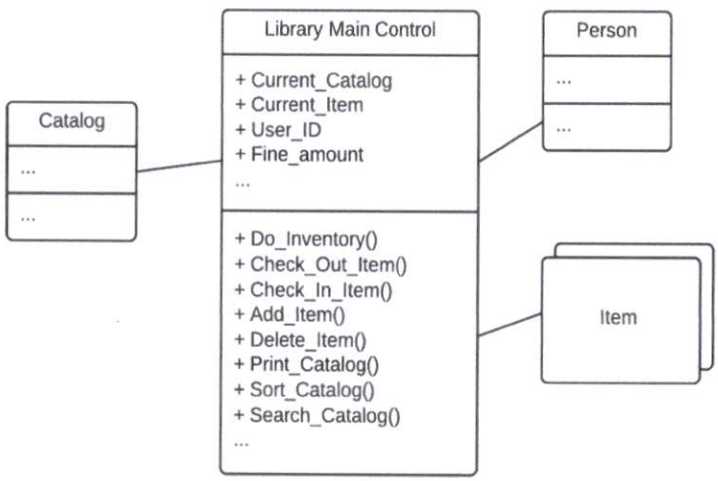

Figure 3.1 LIBRARY Class

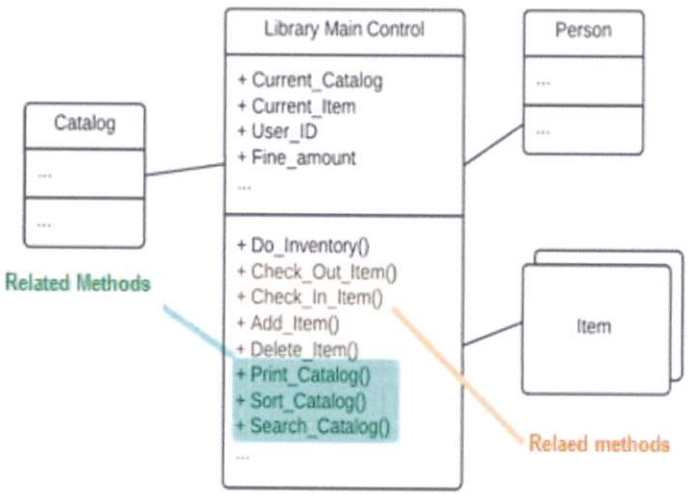

Figure 3.2 Identify LIBRARY Class Methods

2. The second step is to look for "natural homes" for these contract-based collections of functionality and then migrate them there. In this example, we

gather operations related to catalogs and migrate them from the LIBRARY class and move them to the CATALOG class.

We do the same with operations and attributes related to items, moving them to the ITEM class. This both simplifies the LIBRARY class and makes the ITEM and CATALOG classes more than simple encapsulated data tables. The result is a better object-oriented design.

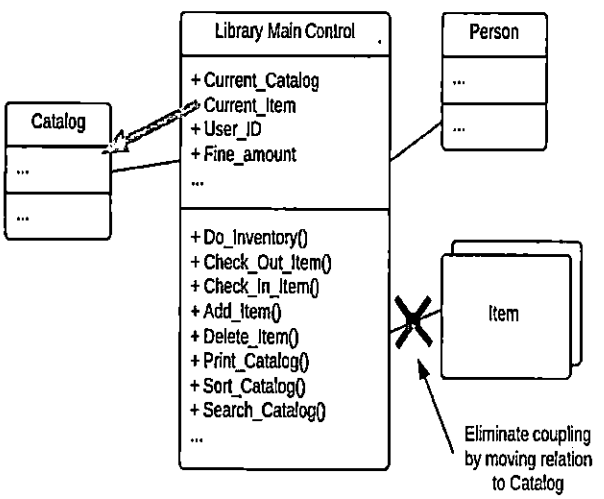

Figure 3.3 Creating the CATALOG Class

3. The third step is to remove all "far-coupled," or redundant, indirect associations. In the example, the ITEM class is initially far-coupled to the LIBRARY class in that each item really belongs to a CATALOG, which in turn belongs to a LIBRARY.

4. Next, where appropriate, we migrate associates to derived classes to a common base class. In the example, once the far-coupling has been removed between the LIBRARY and ITEM classes, we need to migrate ITEMs to CATALOGs, as shown in Figure 3.4.

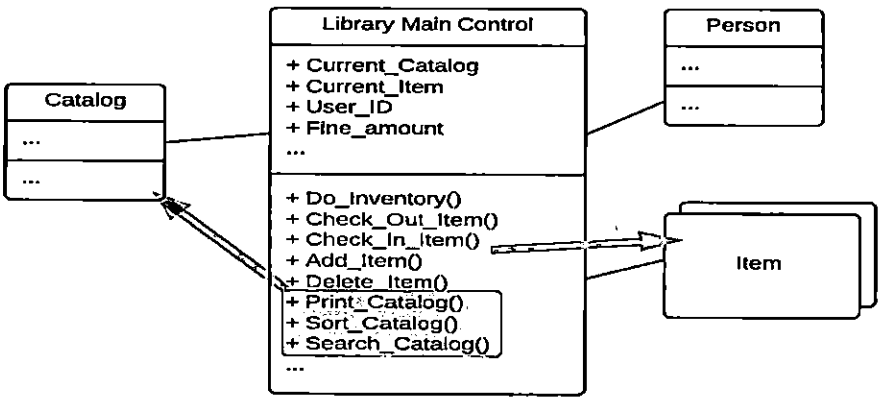

Figure 3.4 Migrating ITEMs to CATALOG

5. Finally, we remove all transient associations, replacing them as appropriate with type specifiers to attributes and operations arguments.

In our example, a Check_Out_Item or a Search_For_Item would be a transient process, and could be moved into a separate transient class with local attributes that establish the specific location or search criteria for a specific instance of a check-out or search.

3.6.1.5 Variations

Sometimes, with a system composed of the BLOB class and its supporting data objects, too much work has been invested to enable a refactoring of the class architecture. An alternative approach may be available that provides an "80%" solution.

Instead of a bottom-up refactoring of the entire class hierarchy, it may be possible to reduce the BLOB class from a controller to a coordinator class. The original BLOB class manages the system's functionality; the data classes are extended with some of their own processing.

The data classes operate at the direction of the modified coordinator class. This process may allow the retention of the original class hierarchy, except for the

migrations of processing functionality from the BLOB class to some of the encapsulated data classes.

3.6.2 MetaObject Facility's (MOF) Model Driven Architecture (MDA)

MDA is a way of developing applications and writing specifications, based on a Platform-Independent Model (PIM) of the application or specification's business functionality and behavior. A complete MDA specification consists of a definitive platform-independent base model, plus one or more Platform-Specific Models (PSM) and sets of interface definitions. Each interface definition describes how the base model is implemented on a different middleware platform. A complete MDA application consists of a definitive PIM, plus one or more PSMs and complete implementations, one on each platform that the application developer decides to support.

MDA is MOF compliant. This means the models – or Meta-models- produced by MDA are designed using MOF language and terms. MDA uses Meta-models to describe the PIMs. It is a very good tool, but it has some disadvantage including:

1. The employment of Meta-Models will raise the level of abstraction which might leave less detail for an operational purpose.
2. According to the MDA principles there are multiple representations of artifacts inherent in a software development process, representing different views of or levels of abstraction on the same concepts. This includes:
    a. Add to the complexity of MDA
    b. Change propagates through the different views: In complex systems, a lot of models, artifacts, and several different levels of abstraction are required. This also increases the complexity of the relationships between them. So when a change needs to be done in an artifact that affects other artifacts and relationships in some cases it is impossible to automate the entire process. So a manual intervention is needed. This is especially difficult if a change is done in the lower levels given the fact that the most of the models at the lower levels are automatically generated. "Some MDA proponents respond that they

generate the code from the model and then let the developers deal with the remaining specifics of platforms, libraries, and legacy interfaces. This is a nightmare because now the poor developer, misled by the "all you need is UML" hype, is stuck having to debug and develop code that a tool generated." (Thomas, 2013)

3. MDA also requires more training and expertise. A developer must learn additional languages used in the modeling. Since artifacts resulting from any stage in the life cycle can impact those produced at any other stage, knowledge of different model technologies and terminologies must exist. Considering the sorts of transformation technologies that are a big part of MDA, developers may also have to be fluent in various transformation notations which can extremely be complex. This adds to the complexity of using MOF's MDA by development teams and makes it less appealing.

In the MDA, models are first-class artifacts, integrated into the development process through the chain of transformations from PIM through PSM to coded application. To enable this, the MDA requires models to be expressed in a MOF-based language. This guarantees that the models can be stored in a MOF-compliant repository, parsed and transformed by MOF-compliant tools, and rendered into XMI for transport over a network. This does not constrain the types of models you can use - MOF-based languages today model application structure, behavior (in many different ways), and data; OMG's UML and CWM are good examples of MOF-based modeling languages but are not the only ones. (OMG, 2014)

OMG members voted to establish the MDA as the base architecture for our organization's standards in late 2001. Software development in the MDA starts with a Platform-Independent Model (PIM) of an application's business functionality and behavior, constructed using a modeling language based on OMG's MetaObject Facility (MOF). This model remains stable as technology evolves, extending and thereby maximizing software ROI. MDA development tools, available now from many vendors, convert the PIM first to a Platform-Specific Model (PSM) and then to a working implementation on virtually any middleware platform: Web Services, XML/SOAP, EJB, C#/.Net, OMG's own CORBA, or others. Portability and

interoperability are built into the architecture. OMG's industry-standard modeling specifications support the MDA, the MOF; UML, now at Version 2.0; the Common Warehouse Metamodel (CWM); and XML Metadata Interchange (XMI). OMG Task Forces organized around industries including Finance, Manufacturing, Biotechnology, Space technology, and others use the MDA to standardize facilities in their domains.

The most recent OMG statement of MDA architecture and direction is this MDA Foundation Model. It states that "Models in the context of the MDA Foundation Model are instances of MOF metamodels and therefore consist of model elements and links between them." This required MOF compliance enables the automated transformations on which MDA is built. UML compliance, although common, is not a requirement for MDA models. (This means, for example, that a suitable development process based on OMG's Common Warehouse Metamodel can be MDA-compliant, since CWM is based on MOF.)

The MDA Guide, Version 1.0.1, just mentioned, defines the MDA. OMG members expect to replace this interim version with an update, based on the Foundation Model also just mentioned, around mid-2005.

Additional OMG specifications populate the architecture. Development tools, provided by vendors, implement these supporting standards. Working together, these tools constitute the working MDA modeling and development environment where architects and developers create MDA applications.

Applications and Frameworks (that is, parts of applications that perform a particular function) can all be defined in the MDA as a base PIM that maps to one or more PSMs and implementations. Standards written this way enjoy two advantages:

The base PIM is truly a business specification, defining business functionality and behavior in a technology-independent way. Technological considerations do not intrude at this stage, making it easy for business experts to model exactly the business rules they want into the PIM.

Once business experts have completed the PIM, it can be implemented on virtually any platform or on multiple platforms with interoperability among them, in order to meet the needs of the industry and companies that use it.

OMG Domain Task Forces, after years of writing specifications in only CORBA, are now writing their base specifications in the MDA to take advantage of these considerations.

OMG recognizes (based on analogy to the CORBA-based Object Management Architecture) three levels of MDA-based specifications:

- The Pervasive Services, including Enterprise necessities such as Directory Services, Transactions, Security, and Event handling (Notification).
- The Domain Facilities, in industries such as Healthcare, Manufacturing, Telecommunications, Biotechnology, and others; and
- Applications themselves, perhaps created and maintained by a software vendor or end user company or enterprise using MDA tools to run an MDA-based approach, but not standardized by OMG.

العنوان:	Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design
المؤلف الرئيسي:	الشندودي، أحمد بن سيف بن محمد
مؤلفين آخرين:	Al Khanjari, Zuhoor Abdullah Salim(Advisor)
التاريخ الميلادي:	2014
موقع:	مسقط
الصفحات:	1 - 95
رقم MD:	972147
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة السلطان قابوس
الكلية:	كلية العلوم
الدولة:	عمان
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، تصميم البرمجيات، المنهجية التسلسلية التقليدية، برمجة تحليل الشبكات الاجتماعية، إعادة التصميم، هندسة البرمجيات الموضوعية
رابط:	https://search.mandumah.com/Record/972147

Chapter 4: Simplified O-O Redesign Approach

The proposed approach is a series of steps. It aims to form a basis for an object oriented design in the form of an object repository from which a full object oriented design can be formed. The nature of the approach has four main characteristics:

1. Provides design level solution
2. Incorporates an organizational point of view in the design
3. Provides reusability
4. Makes it easy to use by development teams

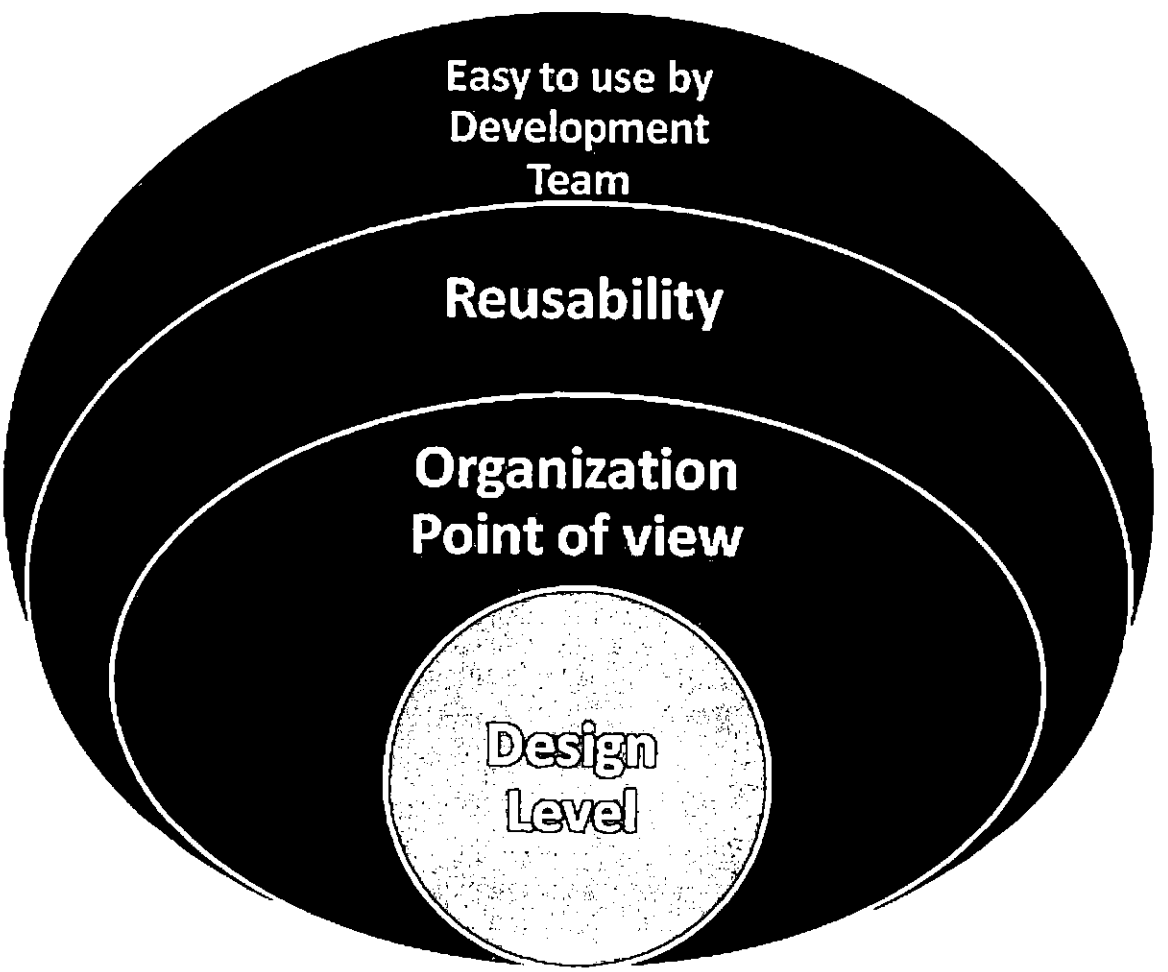

Figure 4.1 Characteristics of the proposed solution

4.1 Provides Design Level Solution

Systems which have been used for a long period of time have a solid business logic which has been obtained through stable requirements and thorough analysis. The requirements define what the stakeholders expect the system to do. The Analysis represents the developers' interpretation of the requirements. If any defects in the requirements are evident, then the users of the system will abandon using it since it does not satisfy their expectations. If the analysis isn't solid, then the resulting system functionality will result in erroneous operations halting the use of the system like wise.

If the system proves useful for a long period of time, then both requirements and analysis prove themselves to be solid. In such long running systems, the changes are usually done to cover quality attributes such as: scalability, reusability, or maintainability. All of which can be handled in the design phase. Starting at the design of the system, the process of redesigning the system will ensure the stability of the requirements. This have been proven to be solid through usability and stakeholders' satisfaction, and maintaining the right business logic. This business logic is obtained through the initial analysis of the system, and proven by the long and right results of the system.

It is essential to realize the difference between a quality attribute missing from the system, and a system defect. Quality attributes can be added through redesigning the system to incorporate them. However, if the defect is in the system requirements, or the correctness of the system functionality, redesigning the system will not solve it. A thorough look through these phases should be taken to solve such problems.

Why not start at the implementation phase? Starting at the design phase ensures we have a broader look at the system as a whole, and not consider implementation issues. It gives us a broader look at the system. Implementation will carry the design implications in it. For example: we cannot use an Object Oriented Programming Language to its full potential if the design does not adhere to such a paradigm. Usually, if we design in a procedural fashion, the resulting implementation will be procedural regardless of the programming language used. Besides, design applies to a

larger context than the implementation because of its abstract nature which is not constrained by language.

Starting at the design level of the system ensures five main advantages:

1. Do not change user requirements which the original system has delivered.
2. Keep the same business logic used in the current system for correct operation.
3. Do not waste time on re-gathering requirements already gathered, and re-analyzing them again, which carries the danger of changing the ongoing functionality of the current system.
4. Ensures a broader look at the system as a whole.
5. Ensures the use of the programming language to its full potential.

4.2 Incorporating an organizational point of view

Each organization has a set of goals it tries to achieve, and a set of interests. These goals and interests represent the vocal point on which all of the organizations operations try to achieve or serve.

The majority of enterprise systems are built to serve a purpose in an organization. A successful system has to touch these interests and goals through its business logic. In doing so, it serves the strategically interest of the organization. It will add value to both: the organizations' other systems and to its own design if the design is done through the organization's point of view.

Designing software which serves a certain problem domain works, but it won't apply or serve a solution in another problem domain. This can be solved when the problem domain is considered through the scope of the organization for which the software is developed. This broadens the problem domain and hence broadens the designed solution.

If the design is achieved by taking the organizational point of view of the problem domain, it will make it apply to more problem domains in the same organization. This is because it serves the strategically important interests and goals of the organization.

This builds a common ground between the solutions causing them to aid each other's operation and business logic.

The Simplified O-O Redesign Approach takes into consideration the main interests of the organization. This must be done at the beginning of the design process, because it will affect the nature of the objects on which the design will revolve. These objects will serve as a basis for future designs and will standardize the design of such pivotal objects representing the long lasting interests of the organization.

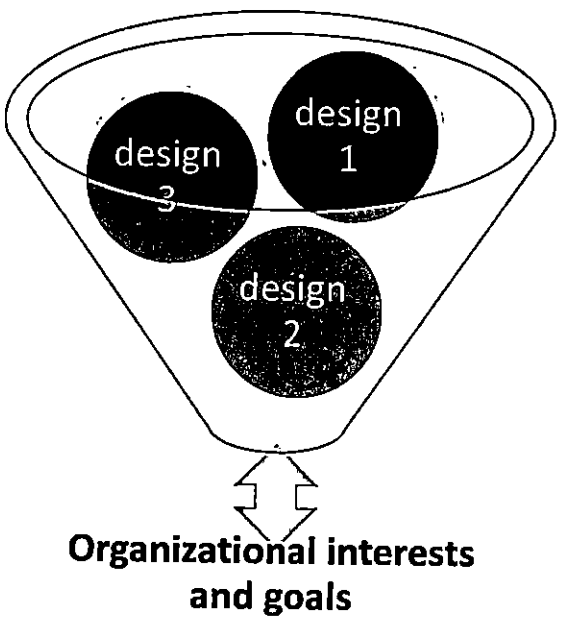

Figure 4.2 Organizational Interests and Goals in relation to Systems' Designs

4.3 Provides reusability

Reusability is essential for the solution because it adds to the lifetime of each design. Instead of constraining it to the problem domain at hand, it makes it applicable to a wider problem domain. This will become inherent to the object oriented design resulting from this approach since the resulting objects are either representing organizational level interests or system specific attributes in an independent abstract way.

The design level reusability is superior to the usual implementation level reusability because of three reasons:

1. It is independent of any implementation level restrictions of language specific limitations.
2. It represents the objects in the most abstract level without the implementation specifics. Hence, less effort and time spent in reuse. At the same time, the reused objects won't be too abstract to be applicable.
3. It gives a better view of the objects in the context of the problem domain and better understanding of how the object is used.

Reusability will cut down the design effort and time in future systems and sometimes in the same system if the same object is used in different parts of it. This leads to faster development. It also affects the testing phase of the system, since the designed objects need not be tested as units, but only in the context of the problem domain.

4.4 Makes it easy to use by development teams

One of the major obstacles which stands in the way of development teams when trying to use the object oriented approach is the knowledge base of object oriented paradigms. Some of the main reasons behind not adopting the object oriented approach for new systems are:

1. The shift in the way of thinking from a procedural "function and data" oriented design. In procedural design, the problem domain is viewed as a linear one dimensional space, and the solution design is a series of steps which follows this domain in its linearity. However, in O-O design, a more abstract and hierarchal problem domain, and the solution design is represented through objects and relations between them.
2. The knowledge base associated with object oriented design is large. It includes new notations, new diagrams, and new relations. It is a whole new paradigm by its own.
3. Development teams have been using the procedural approach for development for too long. They are used to it, and change will cost a delay in upcoming new software development.

The proposed detailed steps of design phase needs to be easy to use, and doesn't need the learning process of the object oriented paradigm. It needs to provide the basis for an object oriented design without the new notation. The pivots of an object oriented design are the objects and that's what this approach produces.

4.5 The Simplified OO Re-Design Approach

The Simplified OO Re-Design Approach is a series of simple steps that extracts the basis for an object oriented design from a traditional procedural design. This basis is the collection of objects which will be:

1. Strategically important to the organization and to the system which is being re-designed.
2. Provides reusability in their design and can be used in designing new systems according to an object oriented approach or any applicable approach.

This collection can be used to design the object oriented design in full. The steps are:

1. Identify Organizational Interests.
2. Identify Modules in the system to be re-designed.
3. Categorize Modules.
4. Module Decomposition.
5. Filter and categorize the resulting objects.

4.5.1 Step 1: Identify Organizational Interests

Every organization has long lasting interests which identify the context on which the operations conducted in the organization run. These interests provide a guide line for developers on which objects reside in the outer problem domain outside the systems they develop and might affect the designs they come up with for the systems.

In the proposed approach, this step is essential to the coming steps for the following reasons:

1. Identify the candidate vessels for an organizational level objects which can be reused across all other designs in the organization.
2. These interests are the material manifestation of the mission of the organization at hand. They represent its core business logic which has to be served on ever operation level.

This step is done on a level higher than the design of the system at hand. It takes a wider look at the system from an organizational perspective not only in the context of the problem domain.

These interests can be identified in using different methods, like:

1. **Analyze mission statements of the organization at hand:** Mission statements usually describe the core business logic and long term goals and interests of the organization.
2. **Study shared parts of existing systems:** long running systems usually have interoperable parts. If these parts are studied they can provide an overview of the organization wide important entities. These parts can be found in:
    a. **Inter-System exchanged data through messaging mechanisms like:** message queues, messaging services...etc. Which are set on an organizational level between systems
    b. **Shared database objects between systems: shared database objects like:** tables, views, and store procedures represent entities needed across the organization's systems.

This step is the corner stone of the approach because it provides an organizational view of the needed objects. This is essential in coming into a design which is reusable on a broader level, which means a larger problem domain.

4.5.2 Step 2: Identify Modules in the System to be Re-Designed

Even within a procedural design, the system is made up of smaller parts or modules. This is mainly due to the business logic which divides the system's logic into smaller business rules. Sets of business rules constitute a module.

Usually, a module is concerned with one coherent part of the system execution. It deals with a finite known set of attributes, functions, and database objects. A module is more of a logical unit which connects these entities in certain known operations.

Modules interact with each other through different means, like: Global variables, messages, or database objects. These means provide information about the state of the system and the state of its execution. This leads – in some cases- to inaccuracy of the information due to the global and linear nature of the procedural design. These information need to be contained in a more autonomous entities which can hold them and regulate their update and use.

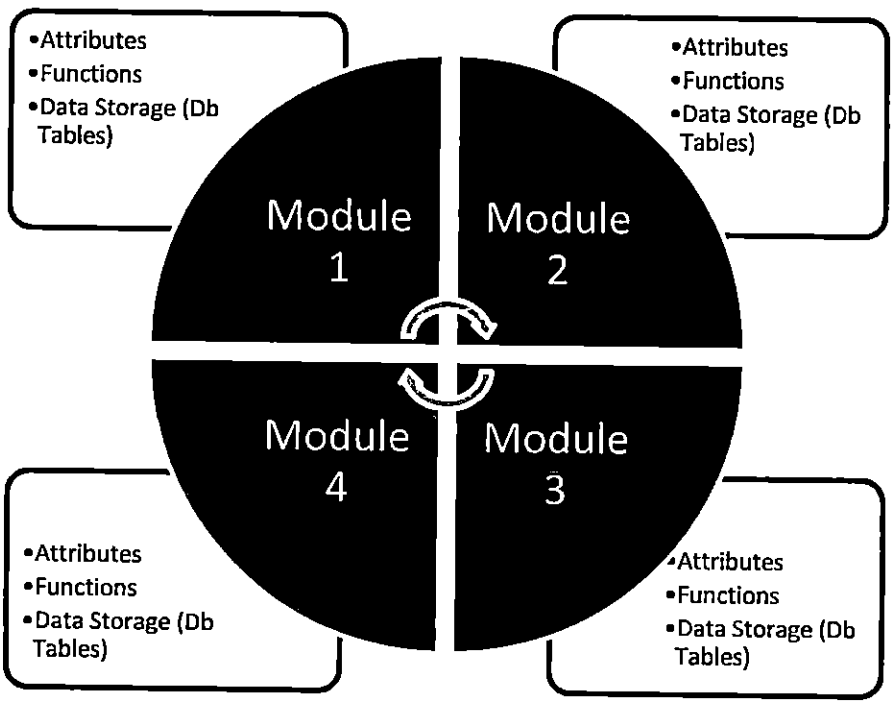

Figure 4.3 Contents of System Modules

Since modules are logical units which are concerned with a set of related operations and attributes in the system, each module can be identified if these operations and attributes can be identified. Here are two module identification guides used in this proposed approach:

1. **Documentation:** If the development team is lucky enough to find a full documentation of the current system, the design and the individual

modules can easily be identified in the documentation of the system. If part of the documentation is available the module can be either derived from the available diagrams, like: Data Flow Diagrams (DFDs) or through studying the available system description.

2. **Graphical User Interface (GUI) screens:** if no documentation is available, then the best option is to use the GUI screens to identify the modules. Usually, each GUI screen is concerned with one system module. It encompasses all needed information for the completion of one or a set of related tasks. For example: Retrieving, editing, adding personal information of a user in a system is usually done in one screen that encompasses these operations, and the needed user information.

Identifying the modules in a system is essential because it separates the work load, and it establishes that logical separation. This could aid into deriving the objects easily from the current design by identifying these related attributes and the operations which use them.

4.5.3 Step 3: Categorize Modules

A system is composed of modules as we have explained previously. These modules if identified can be investigated further and categorized. A module can affect a system by the following means:

1. **Direct relation to the business logic:** The module in this case is an integral part of the business module. It affects the operations in which essential data is manipulated or used into deriving the final critical result. These modules usually cannot be discarded and affect the value of the system in the problem domain the system is associated with. As a result it affects the organization's work as a whole.
2. **Related to the data manipulation but not with the business logic of the system:** The modules here are usually concerned with dealing with data representation and formats. They do not relate directly to the problem domain of the current system and they do not affect the business logic of the system design.

3. **Related to some technical aspects of the system:** The modules here are related to a more technical side of the system. They affect the operability of the system on its own and its interoperability in respect to other systems. Sometimes they deal with some technology specific issues like how to connect to different data storage technologies.

In dealing with the identified modules from the previous step in the approach, it is important to classify the degree in which the module is essential to the system at hand. According to how modules affect a system discussed previously, the detailed steps of the design phase categorize modules into three main categories as follows:

1. **Core Modules:** these modules can be identified using the following guidelines:
   a. They usually relate directly to the organizational interests and goals identified in the first step of the detailed steps of the design phase. If a module deals with the organizational interests, then it is not just essential to the current system's business logic but to the organization's business or operations. This means it is a part of the bigger problem domain not just the one related to the system.
   b. The business logic of the system relies on the module. It represents a set of the system requirements and this is either explicitly evident in documentation or can be identified by examining the requirements set.
   c. Discarding them from the current design will cause the system to fail producing the right results or will hinder current workflow of the system. Hence, these modules are essential to the continuity of the system in the current problem domain and can't be ignored.
2. **Secondary Modules:** these modules are usually peripheral to the system's business logic and operational flow. They usually deal with the representation or manipulation of data, such as: sorting, conversion, or transfer.
3. **Utility Modules:** These modules have the following characteristics:
   a. Like the core modules, discarding them from the design will disrupt the systems operational flow. But the problems caused by discarding them are more technical than logical –business logic related- problems.

b. Deal usually with systems' interoperability and the ability of the system to operate in the technological context of the organization. For example: how the system deals with different data storage or database technologies.

This step in the detailed steps of the design phase is important because it shows which modules contributing the most to the business logic of the system. This ensures three main advantages:

1. **Maintaining sound business logic:** The core modules represent the essence of the system's business logic and giving them a higher priority ensures maintaining the operations and attributes which give the system its value.
2. **Keeping the system interoperable and technology adaptable:** This is achieved by identifying the utility modules and prioritizing the issues which they address in their design.
3. **Optimizing the redesign process in the following steps:** This is because the next steps will only deal with the core modules to maintain the business logic intact. Also, the next steps will only choose from the utility modules which are still applicable in the current intended redesign, since some of the technical issues they deal with might be out of date or could be ignored.

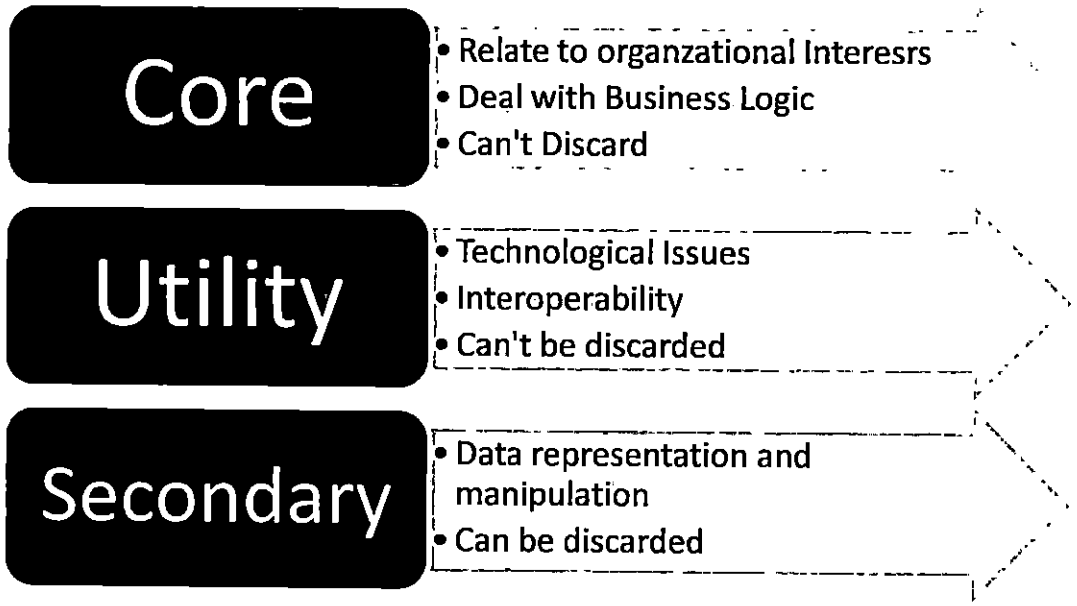

Figure 4.4 Modules Categories in a System

4.5.4 Step 4: Module Decomposition

A module in design is a logical unit. This unit encompasses related attributes-data- and functionality. This collection of data and functionality usually serves one aspect of the system and usually this collection relates to each other in a cohesive manner.

Because the attributes and functions in a module relate to each other in the context of the functionality which the module represents in the system, the module can be decomposed to the objects which represent the logical unit of these related attributes and functions.

In this step the approach of detailed steps of the design phase looks for attributes and functions with which it fills the objects which are represented by the organizational interests identified in the first step. These interests represent empty vessels and they are in turn representing object candidates. The developer in this step fills these empty vessels with related attributes and related function which use these attributes.

The attributes and functions used can be identified using any available documentation. If no documentation is available, then the screen in the graphical user interface of the system can serve as modules where fields represent attributes and functions are

represented by explicit functions in toolbars or by deriving them from the purpose of the screen or using it. Database tables can be used to identify modules since each table is a collection of a logically related columns and each column represents an attribute.

For example in Figure 4.5, the system has two object vessels of type Person and Telephone, and a Person Module which consists of a name, a birth date, and a telephone number attributes. The decomposition step in the approach will decompose the Person module by taking the attributes and filling the object vessels according to the following:

1. **Person object:** It will contain the name and birth date attributes. This will lead in moving getAge() function to it.
2. **Telephone object:** It will get the telephone attribute and an appropriate get function will be defined an called getNumber()

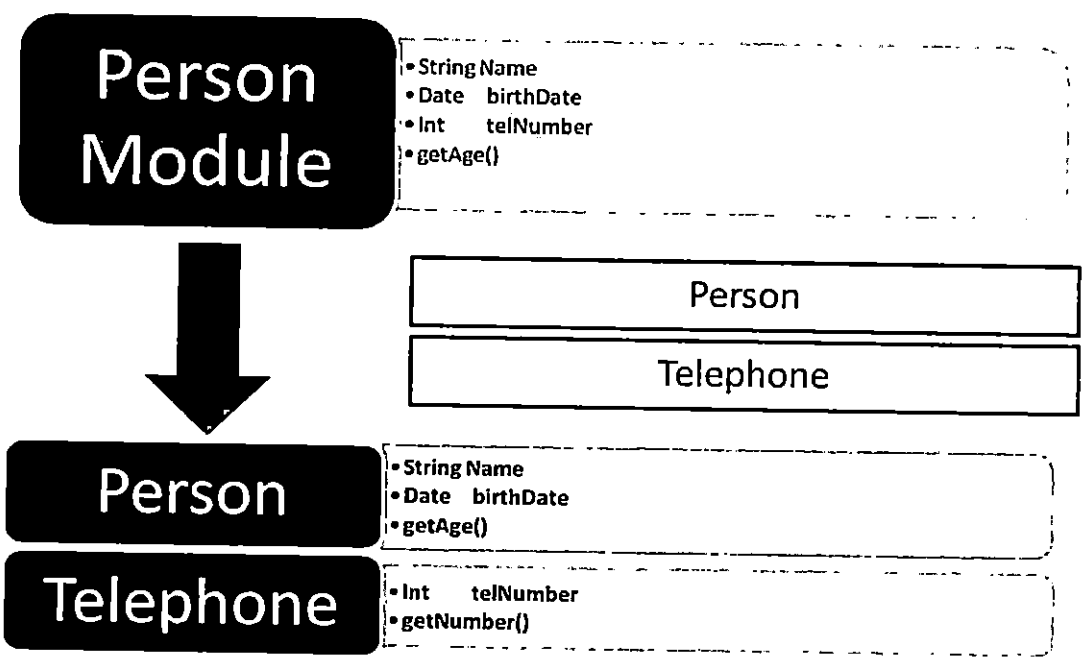

Figure 4.5 Example of Module Decomposition

The decomposition step (Step 4) consists of sub steps. Which are:

1. **Assign the attributes in the module to their related object vessel:** in this sub-step the attributes are moved to their related object. This relation can be identified by looking into database records and applying common logic into

grouping them into one object. For example, if there was a bank account object vessel and the module had the attributes: accountNumber, accountCreateDate, username, logDate, and Balance, and the accountNumber, accountCreateDate, and balance are all in the same database table. Looking at the attributes of the table, the developer could suggest creating a Bank Account object. The attributes accountNumber, accountCreateDate, and balance are all moved to the bank account object.

2. **Add a single empty object vessel called "Miscellaneous object":** this object will serve as the container for all attributes and functions left in the module.
3. Move all the attributes left from sub-step "1" to the "Miscellaneous object".
4. Move all the functions to the "Miscellaneous" object.
5. Move the functions from the "Miscellaneous" to their related objects by examining the attributes used by the functions as parameters and by applying the following rules:
    a. If the attributes belong to a vessel object, then move the function to that object.
    b. If the attributes belong to the Miscellaneous object, then:
        i. If the attributes collaborate naturally with each other but do not belong to any existing object then create new vessel object and move these attributes to the new vessel object.
        ii. If they don't collaborate naturally with each other, then keep them in the Miscellaneous object.
    c. If they are mixed between vessel objects and the Miscellaneous object, then keep the attributes and related functions in the Miscellaneous object.
6. Repeat sub-step 5 until all attributes and functions are removed from the Miscellaneous object. Keep in mind a threshold of repetitions. If the repetitions' threshold is reached, then the Miscellaneous object is left as it is. The iterations in sub-steps "1" and "5" have to take into account the new vessel object created in step "5".

This step is essential to the approach because it specifies the objects which represent the business logic contained in the module being decomposed. The resulting set of

objects represents as a whole the module which has been decomposed. In essence, this identifies the objects which represent the business logic in the module.

It is noteworthy in sub-step "5" of this step it is possible to use the Lack of Cohesion of Methods for more experienced development teams. This method computes the lack of cohesion in a single module by testing for common attributes between pairs of functions.

4.5.5 Step 5: Filtering and Categorizing the Resulting Objects

In this step, the approach takes the resulting sets of objects which have resulted from decomposing the core modules of the system. The step consists of intersecting, comparing and deriving new objects from the resulting sets.

Since each module will result in a set of objects after decomposing it, the step will intersect each set of objects with other resulting sets. Then the results of this intersection of each pair of sets are compared. This comparison will rely on the fact that some of the objects will be redundant in some modules – especially objects which represent the organizational interests-. The better candidate to represent the object is picked and the other one is dropped. The redundant objects are filtered according to the following rules:

- If the object is originating from a module which primarily holds information on the same logical unit then it is picked over any other object in any decomposition set. For example, if an object which holds information on bank accounts originated from a bank account module, and another originating from a budget module, then the Bank Account object from the bank account module is chosen because it is native to the original module.
- If the object is more descriptive (e.g. has more attributes), then it is chosen over the other redundant ones after analyzing the extensibility of that object.

Any objects left out of the intersection and haven't been dropped in the comparison are used to derive new objects in light of the organizational interests and the system's business logic.

The resulting objects are then categorized into three main categories:

1. **Organizational Objects:** They represent the organizational interests directly, and should have priority in the design of new systems or the redesign of existing ones. They are usually the result of intersecting all of the objects sets.
2. **System Specific Objects:** They reflect the business logic of the system. They are usually the result of intersecting pairs of object sets.
3. **Miscellaneous Objects:** They represent some technical considerations, or platform dependency workarounds. They are usually the result of decomposing utility modules.

After categorizing the resulting objects, they are assembled into a large design repository. This repository will contain essential objects on the organization level, on systems level and on a technical level.

4.6 The Result of the Simplified O-O Re-Design Approach

The result of the approach is basis for an O-O design. This basis incorporates an organizational view of the system objects, which adds value to the solution for the organization as a whole, and deals with its core business logic. The repository resulting from the approach can be used as a source of object of new designs. This has two main advantages:

1. **It is design reusability at its best**, where object designs in the repository are reused in new systems or even in the redesign process of existing system. This reusability will lead to less design time and will affect the development time directly.
2. **It imposes a standard representation of objects on the whole organization.** The new systems will adhere to the designs in the repository and this will cause less systems' integration issues in the future because the objects being communicated are designed in a unified manner.

The proposed approach (Simplified O-O Redesign Approach) itself is not meant for Software Engineering professionals. It needs minimal knowledge of O-O principals, and uses fewer notations. It is suitable for teams which want to move fast and need

just a point to start for a full O-O design. Hence fulfilling the easiness of use of the approach the solution requires for it to be more appealing for adoption by development teams.

العنوان:	Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design
المؤلف الرئيسي:	الشندودي، أحمد بن سيف بن محمد
مؤلفين آخرين:	Al Khanjari, Zuhoor Abdullah Salim(Advisor)
التاريخ الميلادي:	2014
موقع:	مسقط
الصفحات:	1 - 95
رقم MD:	972147
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة السلطان قابوس
الكلية:	كلية العلوم
الدولة:	عمان
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، تصميم البرمجيات، المنهجية التسلسلية التقليدية، برمجة تحليل الشبكات الاجتماعية، إعادة التصميم، هندسة البرمجيات الموضوعية
رابط:	https://search.mandumah.com/Record/972147

Chapter 5: Case Study: Social Network Analysis Tool

The case study is about a Social Network Analysis (SNA) tool. The tool basically keeps track of relations using two entities: People and Organizations. The relations are as follows:

- People to people.
- Organizations to Organizations.
- People to Organizations.

The tool provides information on the people, organizations and the relations between them. This information provide references for future enquiries for employment and candidacy for governmental positions, enquiries about organizations which represent public and private sectors, and the relations between people and organizations as in employment for example.

5.1 The SNA Tool Design

The tool has been designed for MARKAZ organization in 2000, and developed and deployed in early 2001. The initial design concentrated on the use of the system and the data as individual parts of data. The design concentrated on the business logic and how to present the data at the screens level. It didn't invest in defining objects to hold these collections of related data and functions. The resulting design was a pure procedural design.

The system suffered from some quality issues:

1. **Integration issues with other systems:** The system had to be used in isolation from the other systems which are related directly to its use. For example, the Human Resources system depends on the information provided by the SNA tool in the process of recruitment and follow up on current employees and their relations. This information had to be supplied manually to the HR system due to the different design in data structures used in the HR system and the SNA tool. Another example would be integration with the financial system.

Where the companies which offer tenders for jobs are investigated for security reasons for approval. The SNA had the information, but communication between the financial system and the SNA had to go through an intermediate tier so the data is manipulated to match each system back and forth.

2. **System limitations:** The system screens and forms were initially based on paper work provided by the concerned department at the organization. So, it literally duplicated the limited fields on the system screens. This had limited the possibility of extending the data the system offers in both quantity and type.

The design was never reused in new systems. This is largely due to the nature of the procedural design. The design concentrated on functionality rather than the data collections being used along the business logic. The design localized the solution to the problem domain of the system, and didn't take into account the larger problem domain which is the organization itself.

The system has been used for early 2012 where the approach was used to derive the objects out of the old design and achieve a full object oriented design out of them. In the next sections the application of the approach on the old system will be shown.

5.2 Application of the Simplified O-O Re-Design Approach on the SNA tool Design

The design of the SNA tool was not documented, so the approach had to rely on the Graphical User Interface of the system as the available source of modular separation in the system.

5.2.1 Step 1: Identify Organization's Interests

The organization where the system is used –Markaz- is a government organization in Oman. The mission statement clearly describes its mission as: "*Provide stratigical overviews about organizations and people of interest and relations between them to the interested government entities*". The mission statement clearly identifies 3 main entities:

1. People.

2. Organizations: public or private.
3. Relations between them.

The government entities of the organization at hand deals with are either: organizations or people who represent the government. So the main interests of the organization derived from the mission statement are: People (Persons), Organizations, and Relations.

These interests will serve as our object candidates or object "vessels" in the next steps. The reason they are important is that they bring an organizational depth to the solution rather than a localized solution to the problem.

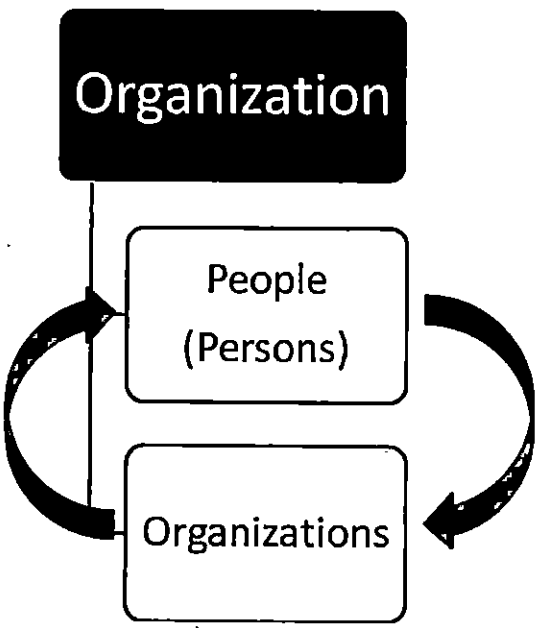

Figure 5.1 Organizational Interests in Markaz

5.2.2 Step 2: Identify the Modules in SNA tool

Since no formal documentation of the current system's design was available to identify the modules from it directly, the Graphical User Interface and the Database tables were the main guide to identify the modules which the system consists of.

The screens provided the functional separation in the system. Usually, each screen is concerned with one aspect of the system functionality, like adding a new person, editing its information, or deleting it. At the same time, the fields of information in each screen provided a guide to the available attributes in the module.

The database tables on the other hand, gave a precise look on the used attributes. Each DB table is a collection of related data fields portrayed in each system screen. The data are represented by columns in each table. These columns collaborate naturally to create one logical unit to represent the attribute of one module or part of it.

As a consequence, each screen or database table is a candidate for a module. Our approach choose the screens as the source of modular separation rather than DB tables, because they are more visual and functionality can be derived from them more than DB tables. At the same time, the DB tables provided guidance on what the technical aspect of the design.

There were three main screens to the system:

1. **System Initial screen:** In this screen the user chooses to connect to the old legacy database. This had information gathered by different applications prior to the development of the current system, and these information are used in the current system. The other option was to connect to the new SQL server database which the current system was built upon. When the user makes the choice then a different method of connection is used depending on the choice.
2. **Personal screen:** This screen showed the basic personal information of the person in question. It also showed the relations this person had with:
    a. Other person (people) and the type of the relation.
    b. Organizations, but not necessarily on the system database.

c. Two addresses: since the initial design was based on the paper work used which had only two blocks for addresses, so did the screen. This limited this type of relation to two addresses only.
d. Three telephone numbers: for home, work, and mobile. This also limited the use of these fields since a person can have one or more telephone number for his/her home, work, or mobile.

3. **Organizations Screen:** this screen showed the related information about the organization, like:
   a. **Organization Number.**
   b. **Organization Name.**
   c. **Establishment date.**
   d. **Organization Address.**
   e. **Organization Telephones.**
   f. **Organization Structure:** This represented the hierarchy of the organization as in job designations and the person in that position.

Since each screen represents a distinctive feature of the system, each screen can serve as module. So according to the screens the SNA tool consists of three modules:

1. **Personal Module:** This module is concerned with personal information, its relation and any additional information relates to a person.
2. **Organization Module:** This module holds the information about an organization and its hierarchy.
3. **DB Module:** This module is represented by the initial screen. It dictates the database connection used in the system and any other DB related issues.

5.2.3 Step 3: Categorize Modules

The purpose of the system is to maintain information on people, organizations, and the relations between them. So the system's business logic revolves around these three main entities. The personal and organizational modules are considered core modules according to the approach for the four following reasons:

1. They are part of the main business logic of the system. Discarding one of them will hinder the operation flow of the system. For example, if the organizational module was discarded, then the information about organizations can't be maintained in the system. Therefore, the system will lose an integral part of its business logic. The same goes for the personal module.
2. Both the personal module and the organizational module have dedicated database tables. This means they are part of the data design of the system, and the information they hold is a main part.
3. Each one of these modules has dedicated screens in the graphical user interface. This means they constitute a large portion of the requirements for the end user, and the initial design put so much emphasis on them.
4. They match the organizational interests identified in the first step directly.

So the core module category for this system includes two modules: the personal module and the organizational module.

The Database module dictates the access to the database and the operations done on the database level by the following:

- It is used in the initial screen to let the user decide which database to connect to. There are two databases used by the system. The first one is a legacy Oracle database, and the second one is a Microsoft SQL Server. The DB module sets up the system's connection to either one of these databases.
- According to the database connection selected by the user, the DB module then is used across the Personal and Organizational module for the Read and Write operations from and to the database of choice. This could provide a logging capability to the system. It acts like a logical Data Access layer above the actual database.

The DB module is categorized as a utility module for two reasons:

1. It doesn't represent any part of the business logic of the system. It deals directly with DB connection and DB operations.

2. Even though it's not part of the business logic of the system, discarding it will cause the system to lose an important feature. This feature is connecting to both the old legacy database by oracle and the new MS SQL server.

So, the system consists of two Core modules:

1. Personal Module.
2. Organizational Module

The DB module represents the utility module.

5.2.4 Step 4: Module Decomposition

According to the organizational interests which were identified in the first step, the main objects candidates or object "vessels" are the: People (persons), organizations, and relations vessels. The object vessels are empty object candidates that will serve as starting point in the decomposition process in this step.

It was mentioned previously in the module identification step, that screen in the GUI served as the modular separation guide. They also contain the used attributes in each module, and the main operations.

In the case study, the personal module was chosen as an example. The step was performed on it and then it was performed on the organizational module. Each of the modules' decompositions resulted in two sets. These sets will be used in the next step.

The personal information screens are represented in the figures next:

Figure 5.2 Main Personal Screen With personal relationships

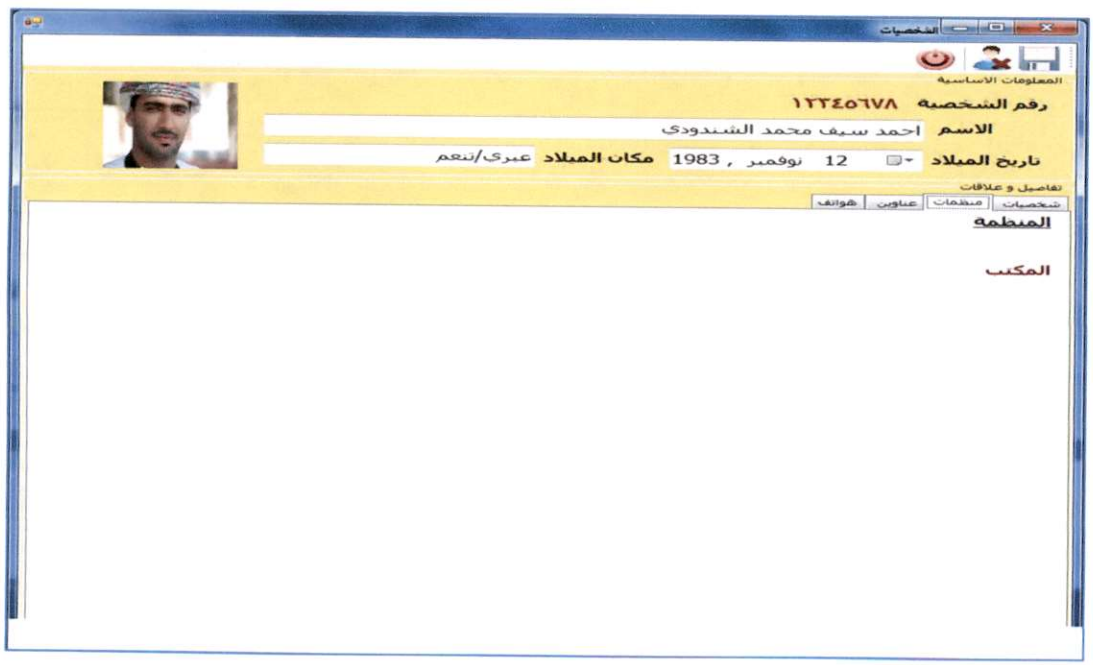

Figure 5.3 Main Personal Screen With organization relationships

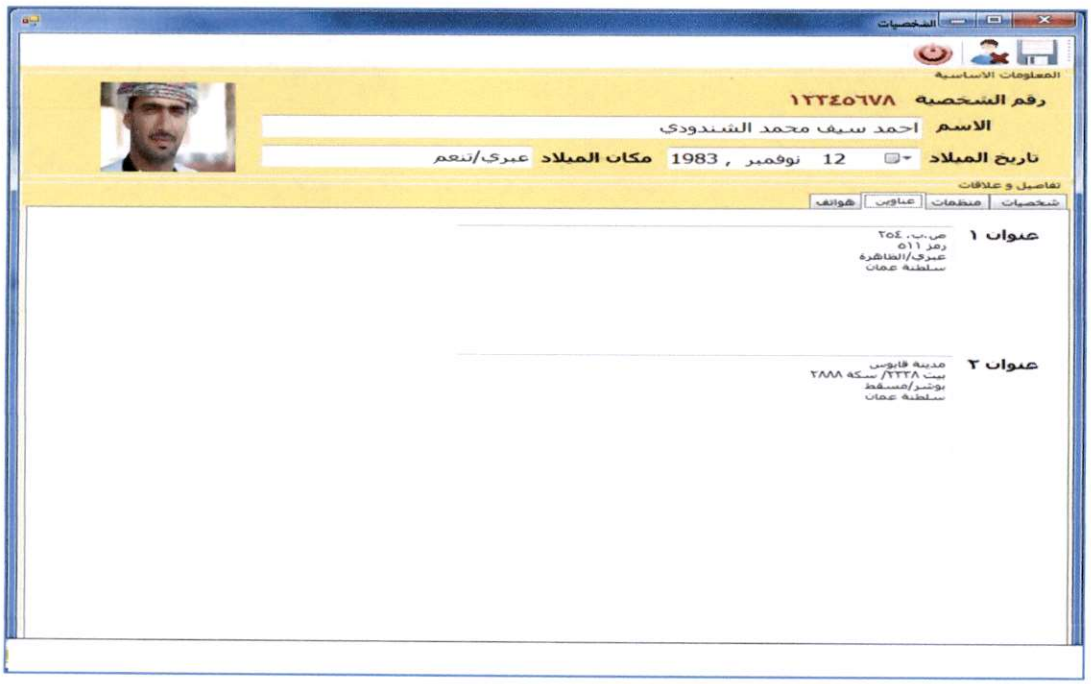

Figure 5.4 Personal screen with relations addresses

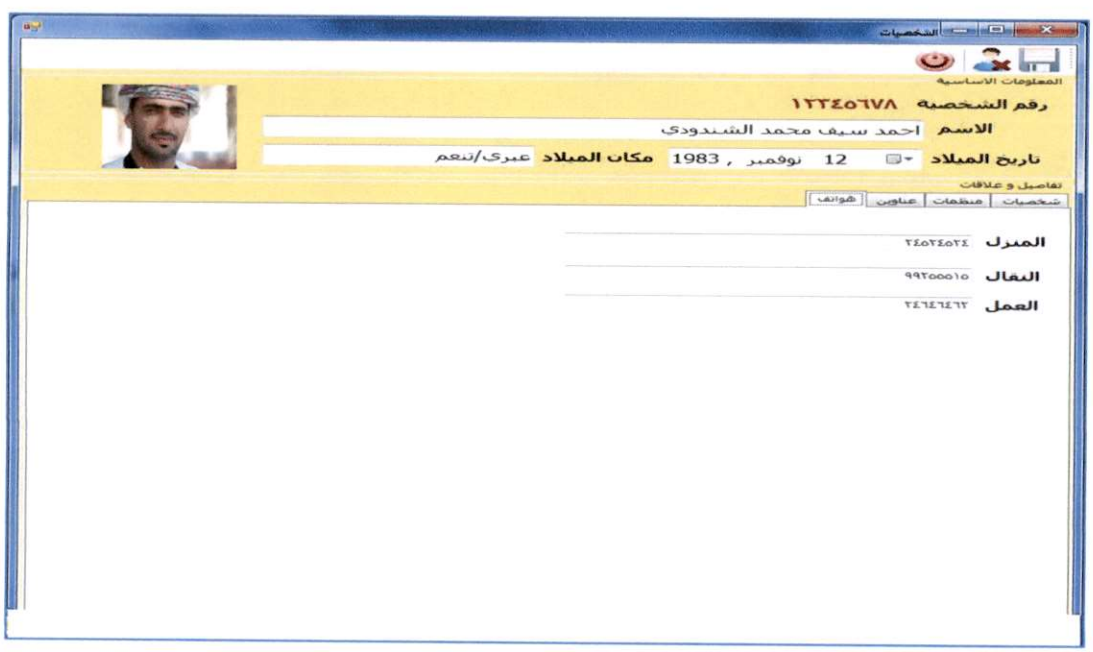

Figure 5.5 Personal screen with relations to telephones

The personal screen in this case study represents the personal module. It contains the following information:

- Person's main information. (To part of the figures 5.2-5.5)

- Relations to other persons. (Figure 5.2)
- Relations to organizations. (Figure 5.3)
- Address information. (Figure 5.4)
- Three telephone numbers. (Figure 5.5)

The personal main information includes the following attributes as derived from the screen:

- PersonNumber: unique system identifier.
- PersonName : a long string representing the person's name.
- DateOfBirth.
- PicturePath: a file path to a picture of the person.

The person to person relations include:

- The related person's name.
- Relation type.

The person to organizations relations include only the organization's name. The person's address information includes two addresses. The telephone information includes three phone numbers: home, work, and mobile.

The operations on the top toolbar include: a save operation, a delete operation, and exiting the screen. The ones of interest are the saving and delete operations. Operations are not as direct as the attributes to identify, so some analysis should be applied. Taking the save operation as an example, five operations or methods can be derived in the light of the five different parts in the personal screen. It is as the following:

- Save Person's main information.
- Save Relations to other persons.
- Save Relations to organizations.
- Save Address information.
- Save three telephone numbers.

The first three methods take a person's number as a parameter. The latter two take the address type and telephone type as parameters respectively. The delete operation can be considered either a general delete for the whole five parts of the personal screen, or several delete operations can be derived like how it was done with the save operation. According to the system use, the delete is a collective delete which deletes all the information on a person. So, the personal module looks like the following figure:

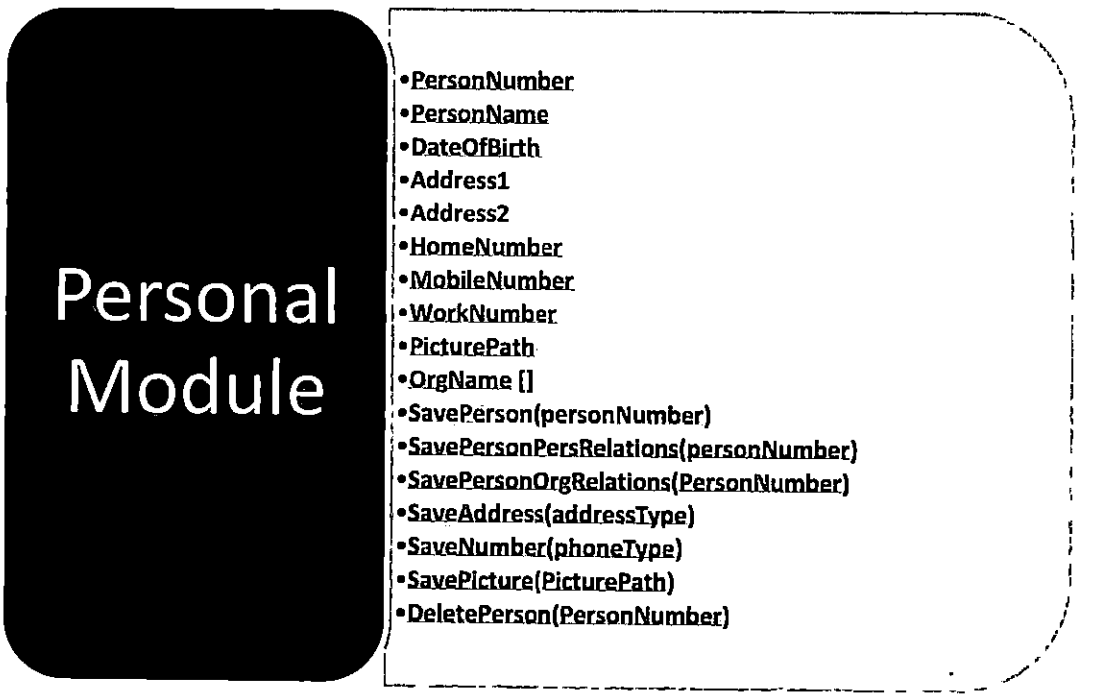

Figure 5.6 Personal Module

The approach's decomposition process is applied in this step as follows:

1. Assign Attributes to Vessels:

In step one of the approach, three main entities were identified, which are: Person, Organization, and relations the approach uses these three as the object vessels where the attributes will be assigned as the following:

- The Person object will take the Person Number, Person Name, and Date of Birth attributes, because they map naturally to a person. They are also part of the personal information stored in the database.

- The Organization object will take the Organization Name, since it's the only attribute which maps naturally to the organization's information directly.
- The Relation object doesn't have any attribute which maps to it naturally. The relation type in the person to person relation is too generic to be included in an object. This object vessel can be taken into consideration when all the objects are identified and filtered at the final step.

The Person object and the Organization object are shown below:

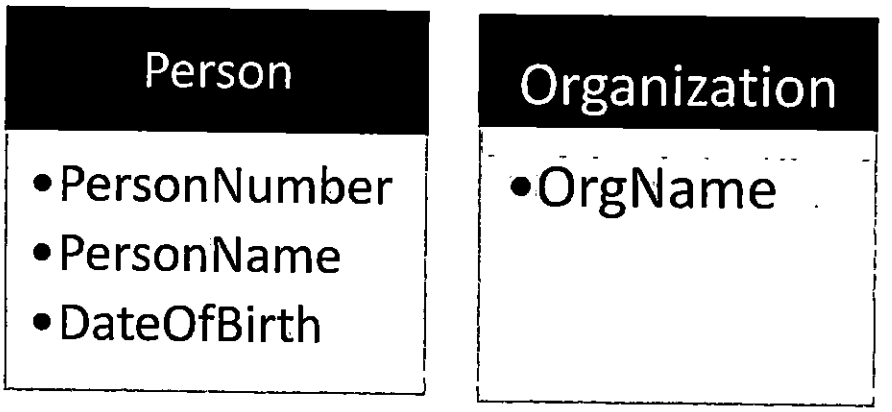

Figure 5.7 Person and Organization Objects

2. Add the empty object called the Miscellaneous Object.
3. Attributes left in the Personal module are put in the single Miscellaneous Object. The Miscellaneous object is represented in Figure 5.8:

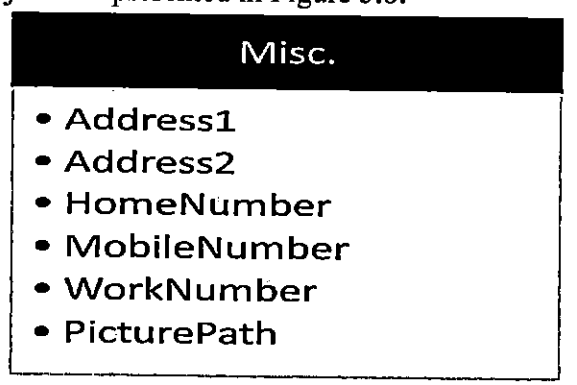

Figure 5.8 Miscellaneous Object

4. Put all the methods in the Miscellaneous Object. The Miscellaneous object is shown in Figure 5.9:

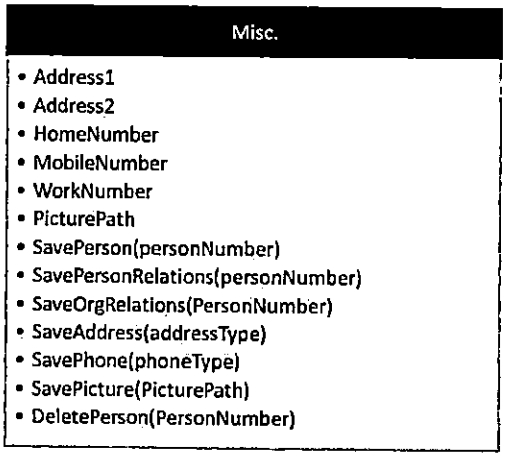

Figure 5.9 Miscellaneous Object after Adding Methods

5. Move each method to its object by applying the rules in the chapter 4. Each method is examined by looking at its parameters as the following:

   i. If they belong to a Vessel Object, Move the method to the vessel object. This applies to the SavePerson() SavePersonRelations(), and SaveOrgRelations() methods since they use Person Number as a parameter. They are moved to the Person Object.

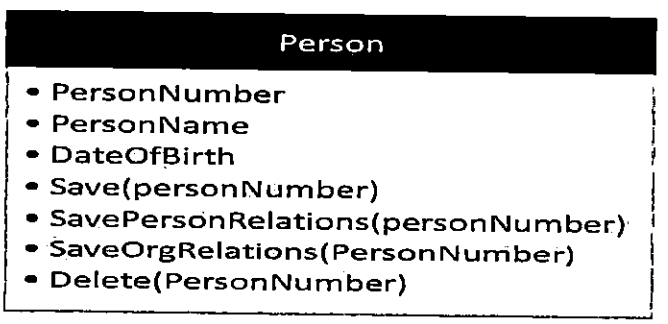

Figure 5.10 Person Object after Adding Methods

   ii. If they belong to Miscellaneous Object, then leave the method or create new Vessel if the attributes collaborate naturally. Address1 and Address2 collaborate naturally, and represent the same logical

entity which is an Address. Same is for the Telephone Numbers where a Telephone Object can be created. The picture path can be used to create a picture object too. So, the new derived objects from the collaborating attributes are: An Address Object, a Telephone Object, and a Picture Object. Some analysis must be applied to these objects' design to extend their design to be more usable. The resulting object will look like the following figures:

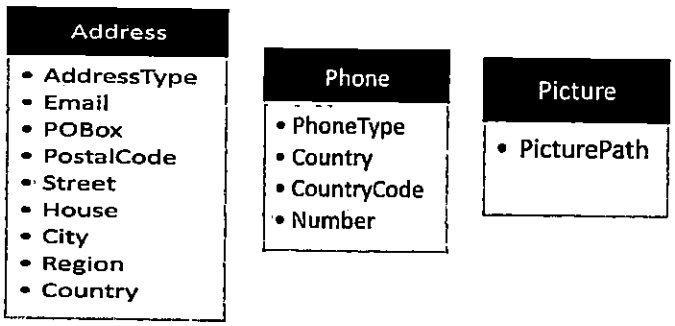

Figure 5.11 Address, Phone, and Picture Objects

iii. If the parameters are mixed between the Miscellaneous Object and the other objects, then leave the methods in the Miscellaneous Object.

The Miscellaneous Object will look like the following figure:

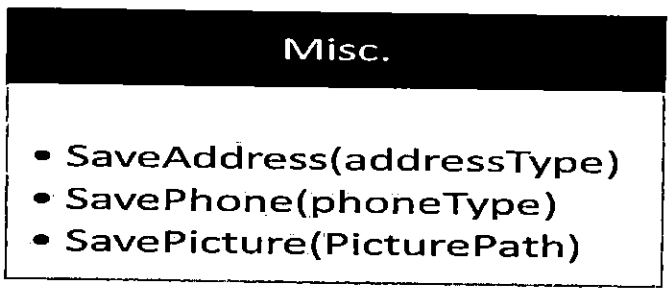

Figure 5.12 Miscellaneous Object after Adding Methods

6. Repeat the process while taking into account new Vessel Object. The objects at the end of this step will look like the following.

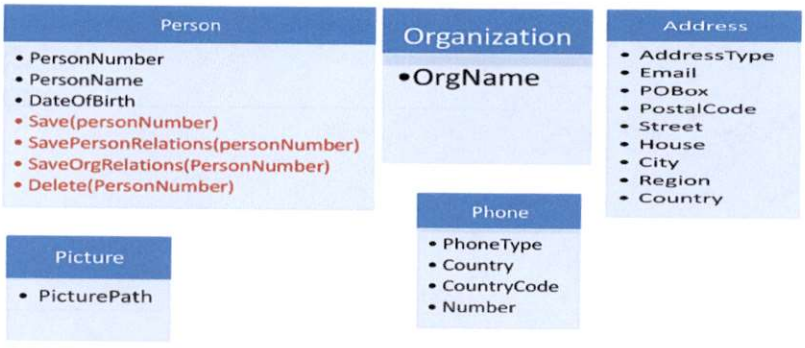

Figure 5.13 Objects resulting from step 4

At the end of the decomposition step the Personal Module has been decomposed to five main objects: Person, Organization, Address, Phone and Picture objects.

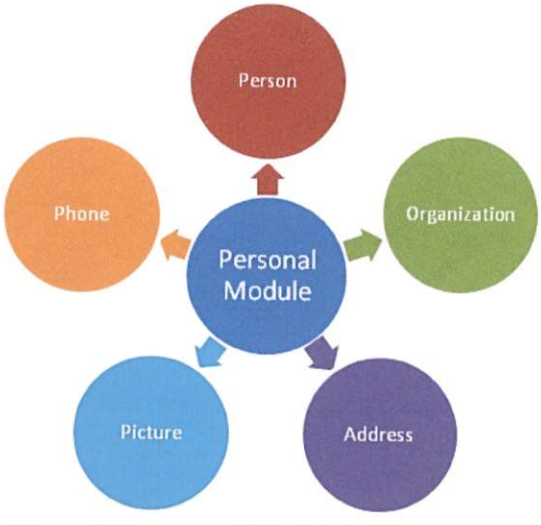

Figure 5.14 Decomposition of the Personal Module

The organizational module has undergone the decomposition process and the resulting objects were: Organization, Person, Address, Phone, Logo (Picture), and Designation objects.

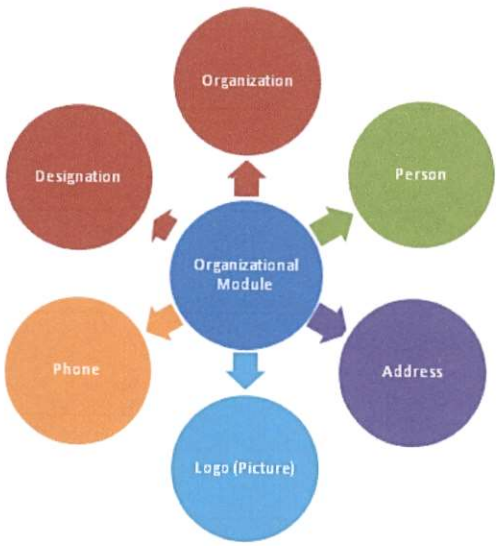

Figure 5.15 Decomposition of the Organizational Module

5.2.5 Step 5: Filtration of the Object Sets

The set of objects resulting from the modules after decomposition are filtered and categorized to obtain the final set of objects that will be used for the O-O design of the system. There are redundant objects in the object sets resulting from the decomposition process like: Person, Organization, Address, Phone, Picture. The objects are chosen among the different sets as the following:

- The Person Object from the Personal Module is chosen because it is original to the personal information found in the Personal Module.
- The Organization Object is chosen from the set of the Organizational Module, because it is part of the original module of the organizations' information.
- The other objects are chosen according to their extensibility. The Address Object is chosen from the Personal Module. The Phone and Picture objects are the same in both cases, so either one will suffice.

The Designation Object left from the Organizational Module is then analyzed in light of the organizational interests. One of the organizational interests object vessel has not been addressed in the Decomposition Process is the Relation Object. In an abstract look, a person's designation in a company is the position of a person in an organization. This position is nothing but a relation between the two. Hence,

Designation can be looked at as an object of type relation which connects two objects. This is demonstrated in the following figure:

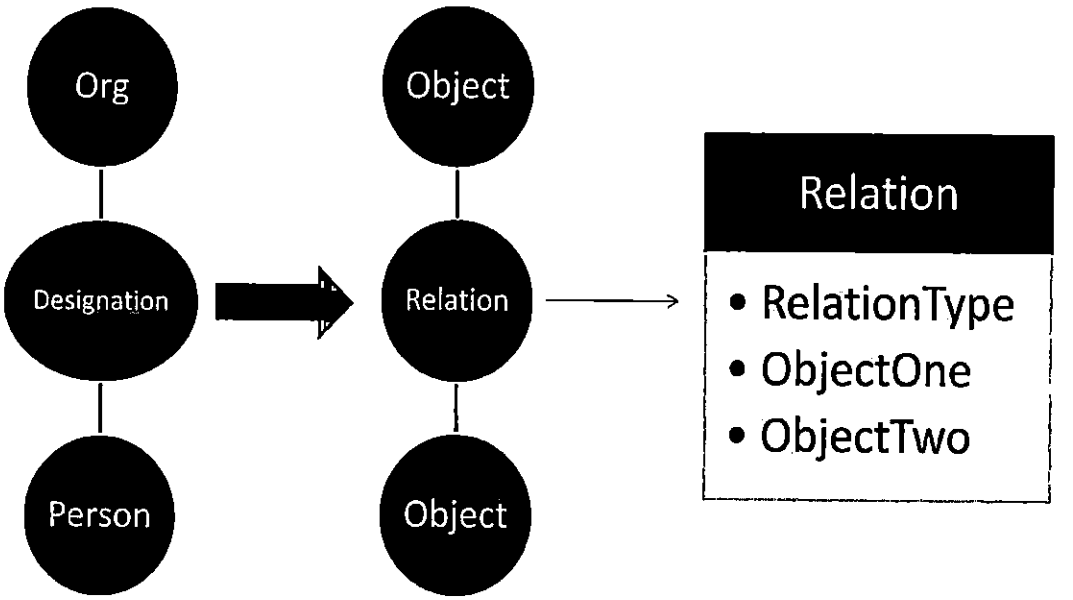

Figure 5.16 Deriving the Relation Object

Now, the Relation Object is an encapsulation which holds two objects and defines a relation type between the two. It is a generic object which maps to the organizational interests and puts the objects together.

The objects after filtration are: Person, Organization, Relation, Address, Phone, and Picture. They are categorized as follows:

- **On the organizational level:** Person, Organization and Relation are categorized as organizational level objects because they represent the organizational interest.
- **On the system level:** Address, Phone, and Picture are categorized as system objects (specific to the SNA tool).

The repository after filtration and categorization looks like the following figure:

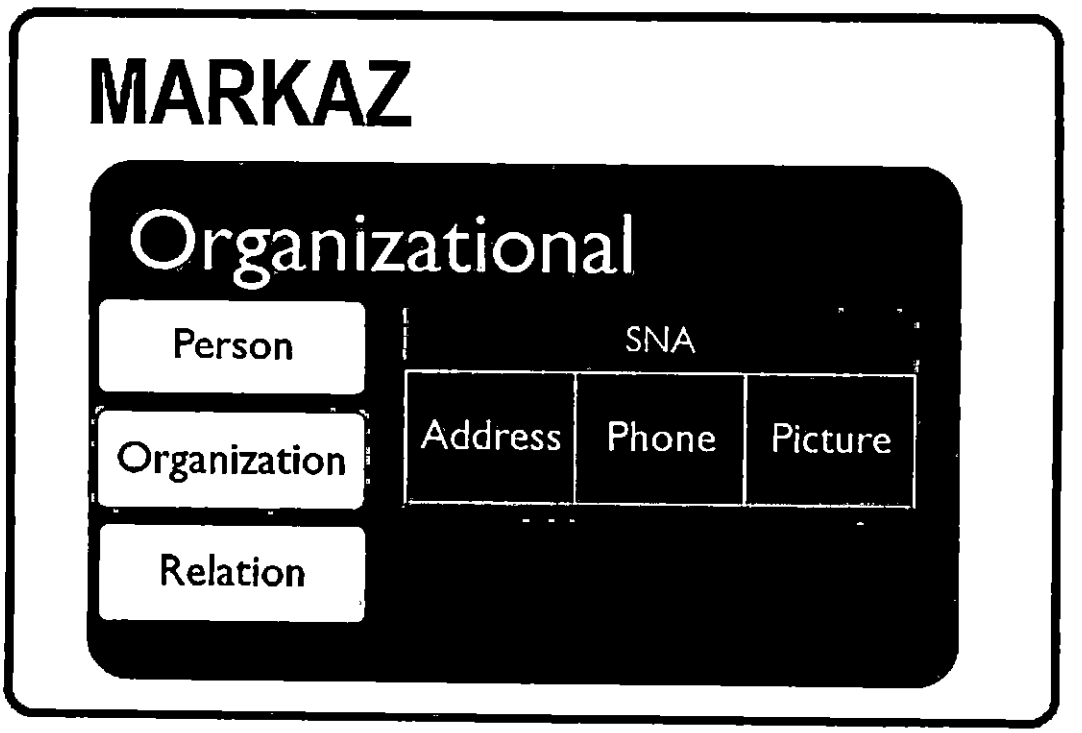

Figure 5.17 The Repository in MARKAZ

This repository is the basis for the new O-O design of the system. The organizational part of it represents long term goals and interests, and the system part of it represents objects specific to the system at hand.

The resulting objects made it possible to:

- Extend the relation between entities in the system. For example, a person under this design can have more than two addresses like in the old system design. The person to person relation is more than just a name and a relation type. It is now a relation between two persons. Each of them is uniquely identified and has its own data.
- The system to system integration issues are now minimized due to the standard design of objects which are ready to use from the repository of objects.
- The repository gives better choices for future designs when it comes to object design. Each object can be extended and encapsulated to tailor it better for a specific system, but still keep it original base object standard design for interoperability between systems.

5.3 Going further with the Simplified O-O Redesign Approach

The nature of the repository and the objects is dynamic. For instance, the Person Object has been used in several systems' designs as in the HR system, the Recruitment System, and the Retirement System by applying O-O principles.

A person in the real employment process is transferred between three systems: The Recruitment System as a candidate or a recruit, the HR System as an employee and in the Retirement System as a retired employee. In the old design, this cycle was done manually by adding in one system and deleting in another. With the new design it is simply done by inheriting the Person Object into a Candidate Object and a Recruit Object to be used in the Recruitment System. Then the base Person Object data is transferred from Recruitment to HR System and then using the CurrentEmployee object to live in the employment domain. The CurrentEmployee object is inherited from the Person Object. When the employee is retired the base object is used again to transfer the data to the Retirement System on the fly. The person now is a Retired Object. The information in all of the cycle is kept, and the communication between the systems was done on line. The standard design and the O-O approach eased the process and made the three systems interoperable.

The Scenario above is shown in Figure 5.18.

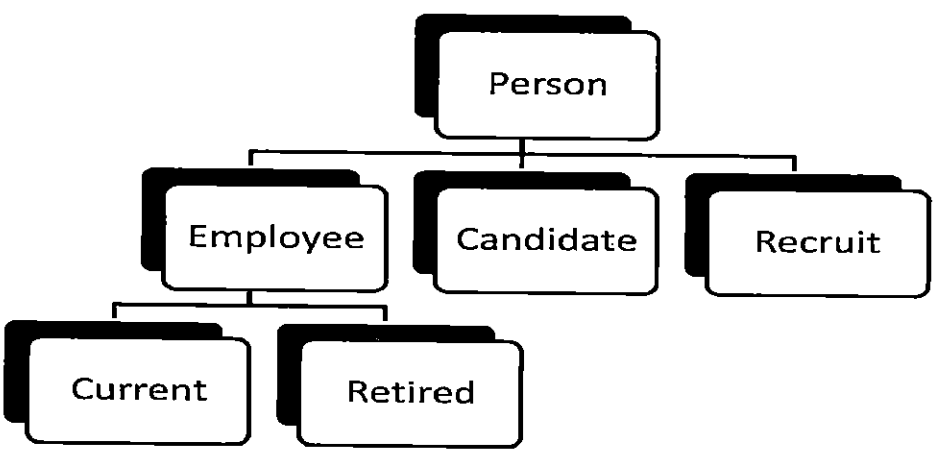

Figure 5.18 Applying Iheretance to the Repository Objects

العنوان:	Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design
المؤلف الرئيسي:	الشندودي، أحمد بن سيف بن محمد
مؤلفين آخرين:	Al Khanjari, Zuhoor Abdullah Salim(Advisor)
التاريخ الميلادي:	2014
موقع:	مسقط
الصفحات:	1 - 95
رقم MD:	972147
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة السلطان قابوس
الكلية:	كلية العلوم
الدولة:	عمان
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، تصميم البرمجيات، المنهجية التسلسلية التقليدية، برمجة تحليل الشبكات الاجتماعية، إعادة التصميم، هندسة البرمجيات الموضوعية
رابط:	https://search.mandumah.com/Record/972147

Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design

Ahmed Al-Shandoudi

Abstract

Object oriented software engineering has proven to be a powerful tool in developing huge enterprise libraries. Yet, it comes with new notations and a number of diagrams, which traditional software engineering practitioners are not well acquainted with. The change of the development approach from a traditional approach to an O-O approach can prove to be costly, and incorporates a lot of new techniques and procedures. This thesis identifies the steps in moving an application from a traditional design paradigm to an object-oriented paradigm. It uses an example of Social Network Analysis tools where modules were decomposed to objects and operations, implementation changed, data storage was changed, and free relations between objects were introduced in the final OO application.

The main purpose for the change was the versatile nature of objects. Using the ready traditional design will cut the time for the designers and developers alike. The introduction of objects will lead to more reusability. The OO design provides clearer separation between objects; hence there is more specialization.

This move can be achieved by identifying the main modules in the original design. Then they are decomposed into processes and components. In each component, there is a need to identify objects and operations, and define possible relations between objects. On the data storage side, each derived object's data should stand alone in a relational data base. Relations between objects are represented as pair of keys in relation tables.

Studying the move, it was found that deriving the new OO design depends on the main interests of the hosting organization. The level of abstraction of the original design dictates the change. By using OO techniques such as inheritance and such would be used for creating specialized components from the new design.

The findings are significant because they show that the team knowledge affects the move greatly. The degree of abstraction in an object is essential for future benefit of the OO design. The move to an OO design is an intermediate step. It can be the basis for more specialized solutions using the produced objects.

منهجية مبسطة في اعادة التصميم: إستخلاص تصميم موضوعي التركيز من تصميم تسلسلي تقليدي.

أحمد بن سيف بن محمد الشندودي

الخلاصة

أثبت هندسة البرمجيات بمنهج موضوعي التركيز أنها أداة قوية في تصميم البرمجيات على مستوى التصاميم الضخمة. لكنها تأتي بالعديد من الرموز و الاشكال البيانية التي لم يعتادها مهندسو البرمجيات التقليديين. إن التغيير في منهجية التصميم من المنهجية التسلسلية التقليدية إلى تلك المركزة على المواضيع بالامكان أن تكون مكلفة، كما أنها تنطوي على العديد من التقنيات و الأساليب الجديدة. هذه الأطروحة تتناول منهجية على هيئة خطوات لهذه النقلة؛ حيث أنها تتناول تطبيقاً عملياً على تصميم لبرمجية تحليل الشبكات الاجتماعية حيث تم استخلاص المواضيع أو العناصر المكونة للبرمجية من المكونات الاساسية في تصميم النظام.

الهدف الرئيسي من التغيير هو ما تقدمة المواضيع من مرونة في التعامل. حيث بالامكان إعادة إستخدامها . كذلك فان خصوصية كل موضوع تجعل تحديد استخداماته أدق و أكثر تخصصية.

هذا التغيير ممكن من خلال التعرف على المكونات الرئيسية للنظام، ثم تحليل هذه المكونات الى مكوناتها الاصغر و العمليات التي تتضمنها. ومن خلال المكونات الاصغر يتم استخلاص المواضيع و العمليات التي ترتبط بها. كذلك التعرف على العلاقات التي تربط المواضيع بعضها ببعض.

من خلال دراسة التغيير وُجد أنه يعتمد على الاهتمامات الرئيسية للمؤسسة التي تحتض النظام المردا تغيير تصميمه. كذلك فإن مقدار التجيرد و العزل بين اختصاصات المكونات في التصميم يتحكم في الاستفادة من التقنيات المركزةعلى المواضيع مثل: الوراثة البرمجية. حيث يتم تصميم مواضيع جديدة من تلك الناتجة من عملية التغيير.

تكمن أهمية الأطروحة في أنها تبين أن معرفة فريق التصميم بأسس هندسة البرمجيات الموضوعية التركيز تؤثر على عملية التغيير. كذلك أن درجة التجريد في المواضيع الناتجة من عملية التغيير تأثر على مقدار الاستفادة منها من خلال التقنيات موضوعية التركيز. كما ان خطوة التغيير هي خطوة وسطية و التصميم الناتج بالامكان استغلاله في انتاج تصاميم أخرى بتقنيات هندسة برمجيات مختلفة.

العنوان:	Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design
المؤلف الرئيسي:	الشندودي، أحمد بن سيف بن محمد
مؤلفين آخرين:	Al Khanjari, Zuhoor Abdullah Salim(Advisor)
التاريخ الميلادي:	2014
موقع:	مسقط
الصفحات:	1 - 95
رقم MD:	972147
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة السلطان قابوس
الكلية:	كلية العلوم
الدولة:	عمان
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، تصميم البرمجيات، المنهجية التسلسلية التقليدية، برمجة تحليل الشبكات الاجتماعية، إعادة التصميم، هندسة البرمجيات الموضوعية
رابط:	https://search.mandumah.com/Record/972147

Table of Contents

العنوان:	Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design
المؤلف الرئيسي:	الشندودي، أحمد بن سيف بن محمد
مؤلفين آخرين:	Al Khanjari, Zuhoor Abdullah Salim(Advisor)
التاريخ الميلادي:	2014
موقع:	مسقط
الصفحات:	1 - 95
رقم MD:	972147
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة السلطان قابوس
الكلية:	كلية العلوم
الدولة:	عمان
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، تصميم البرمجيات، المنهجية التسلسلية التقليدية، برمجة تحليل الشبكات الاجتماعية، إعادة التصميم، هندسة البرمجيات الموضوعية
رابط:	https://search.mandumah.com/Record/972147

Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design

Ahmed Saif Mohammed Al-Shandoudi

A thesis submitted in partial fulfillment

of the requirements for the degree

Master of Science

in

Computer Science

Department of Computer science

College of Science

Sultan Qaboos University

Sultanate of Oman

2014

Thesis of: Ahmed Saif Mohammed Al-Shandoudi (I.D. #: m084744)

Title of Thesis: Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design

Thesis Committee:

1. Supervisor: Dr. Zuhoor Al-Khanjari

Title: Associate Professor

Department: Computer Science

College: College Of Science

Institution: Sultan Qaboos University

Signature:.....**Date**..20/7/2014

2. Member: Dr. Abdullah Al-Hamdani

Title: Assistant Professor

Department: Computer Science

College: College Of Science

Institution: Sultan Qaboos University

Signature:.....**Date**..21/7/2014

3. Member: Dr. Asma Al- Busaidi

Title: Assistant Professor

Department: Computer Science

College: College Of Science

Institution: Sultan Qaboos University

Signature:.....**Date**..2/8/2014

Thesis Examining Committee:

1. Chair: Prof. Mujibur Rahmen

Title: Chairman of the Examination Committee

Department: Physics

College: College Of Science

Institution: Sultan Qaboos University

Signature: ..... **Date...** 20/7/2014.

2. Supervisor: Dr. Zuhoor Al-Khanjari

Title: Associate Professor

Department: Computer Science

College: College Of Science

Institution: Sultan Qaboos University

Signature: ..... **Date..** 20/7/2014

3. Member (HoD representative): Dr. Youcef Baghdadi

Title: Associate Professor

Department: Computer Science

College: College Of Science

Institution: Sultan Qaboos University

Signature: ..... **Date..** 20/7/14

4. External Examiner: Dr. Mohammed Sarrab

Title: Research Associate

Center: Communication and Information Research Center

Institution: Sultan Qaboos University

Signature: ..... **Date..** 20/7/2014

Acknowledgment

Thanks are due to Allah, for every success is due to Allah's mercy. I thank Allah for the strength which was bestowed upon me, and the patience which was given to my family, coworkers, and supervisor. For without them I would have given up.

I thank my mother, my wife, and children for they have brought everything good in me. They sacrificed their time and neglected their joy to support me.

I would like to thank my supervisor for her relentless effort in guiding me. She has been patient and above all understanding. I would like to thank my thesis committee for their support.

I would like to thank Dr. Naoufel Kraiem for the time and help he has given me.

Finally, I thank my late father. For he has been my inspiration to be strong and triumphant.

Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design

Ahmed Al-Shandoudi

Abstract

Object oriented software engineering has proven to be a powerful tool in developing huge enterprise libraries. Yet, it comes with new notations and a number of diagrams, which traditional software engineering practitioners are not well acquainted with. The change of the development approach from a traditional approach to an O-O approach can prove to be costly, and incorporates a lot of new techniques and procedures. This thesis identifies the steps in moving an application from a traditional design paradigm to an object-oriented paradigm. It uses an example of Social Network Analysis tools where modules were decomposed to objects and operations, implementation changed, data storage was changed, and free relations between objects were introduced in the final OO application.

The main purpose for the change was the versatile nature of objects. Using the ready traditional design will cut the time for the designers and developers alike. The introduction of objects will lead to more reusability. The OO design provides clearer separation between objects; hence there is more specialization.

This move can be achieved by identifying the main modules in the original design. Then they are decomposed into processes and components. In each component, there is a need to identify objects and operations, and define possible relations between objects. On the data storage side, each derived object's data should stand alone in a relational data base. Relations between objects are represented as pair of keys in relation tables.

Studying the move, it was found that deriving the new OO design depends on the main interests of the hosting organization. The level of abstraction of the original design dictates the change. By using OO techniques such as inheritance and such would be used for creating specialized components from the new design.

The findings are significant because they show that the team knowledge affects the move greatly. The degree of abstraction in an object is essential for future benefit of the OO design. The move to an OO design is an intermediate step. It can be the basis for more specialized solutions using the produced objects.

منهجية مبسطة في اعادة التصميم: إستخلاص تصميم موضوعي التركيز من تصميم تسلسلي تقليدي.

أحمد بن سيف بن محمد الشندودي

الخلاصة

أثبت هندسة البرمجيات بمنهج موضوعي التركيز أنها أداة قوية في تصميم البرمجيات على مستوى التصاميم الضخمة. لكنها تأتي بالعديد من الرموز و الاشكال البيانية التي لم يعتادها مهندسو البرمجيات التقليديين. إن التغيير في منهجية التصميم من المنهجية التسلسلية التقليدية إلى تلك المركزة على المواضيع بالامكان أن تكون مكلفة، كما أنها تنطوي على العديد من التقنيات و الأساليب الجديدة. هذه الأطروحة تتناول منهجية على هيئة خطوات لهذه النقلة؛ حيث أنها تتناول تطبيقاً عملياً على تصميم لبرمجية تحليل الشبكات الاجتماعية حيث تم استخلاص المواضيع أو العناصر المكونة للبرمجية من المكونات الاساسية في تصميم النظام.

الهدف الرئيسي من التغيير هو ما تقدمة المواضيع من مرونة في التعامل. حيث بالامكان إعادة إستخدامها . كذلك فان خصوصية كل موضوع تجعل تحديد استخداماته أدق و أكثر تخصصية.

هذا التغيير ممكن من خلال التعرف على المكونات الرئيسية للنظام، ثم تحليل هذه المكونات الى مكوناتها الاصغر و العمليات التي تتضمنها. ومن خلال المكونات الاصغر يتم استخلاص المواضيع و العمليات التي ترتبط بها. كذلك التعرف على العلاقات التي تربط المواضيع بعضها ببعض.

من خلال دراسة التغيير وُجد أنه يعتمد على الاهتمامات الرئيسية للمؤسسة التي تحتض النظام المردا تغيير تصميمه. كذلك فإن مقدار التجيرد و العزل بين اختصاصات المكونات في التصميم يتحكم في الاستفادة من التقنيات المركزةعلى المواضيع مثل: الوراثة البرمجية. حيث يتم تصميم مواضيع جديدة من تلك الناتجة من عملية التغيير.

تكمن أهمية الأطروحة في أنها تبين أن معرفة فريق التصميم بأسس هندسة البرمجيات الموضوعية التركيز تؤثر على عملية التغيير. كذلك أن درجة التجريد في المواضيع الناتجة من عملية التغيير تأثر على مقدار الاستفادة منها من خلال التقنيات موضوعية التركيز. كما ان خطوة التغيير هي خطوة وسطية و التصميم الناتج بالامكان استغلاله في انتاج تصاميم أخرى بتقنيات هندسة برمجيات مختلفة.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Software, as a product, delivers the computing potential embodied by computer hardware. It is used to transform, produce, manage, acquire, modify, display, or transmit information. Information can be as simple as a single bit or as complex as a multimedia simulation (Pressman, 2014).

Software development or application development is the process of developing a single application or a full functional integrated system. In its basic form, it is the coding or implementation of the application at hand, but in a more broader professional sense it is all that is involved between the conception of the desired software's specifications through to the final manifestation of the software, ideally in a planned and structured process.

As the complexity of the desired solution or system increases, the need for a well planned and well executed process increases. The engineering aspect of the software development addresses this issue. Software Engineering is defined as the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; in other words it is the application of engineering techniques and strategies to software. It is based on three layers: process, methods, and tools. Software Engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Software Engineering methods provide the technical "how to's" for building software. Methods encompass a broad array of tasks that include requirements analysis, design, implementation, testing, and maintenance. Finally, Software Engineering tools provide automated or semi-automated support for the development process and the methods used in Software Engineering (Pressman, 2014).

Software Engineering defines an abstract representation of a process methodology, known as the process model. Each methodology constitutes a framework used to structure, plan, and control the process of developing a system. Waterfall and Agile represent process models. They don't specify how to do things, but they outline the types of activities, which are done. For example, Waterfall identifies the phases that a

project goes through without saying what artifacts to produce or what tools to use. This is also the case with Agile model, which defines core values in the form of the Agile manifesto, time-boxed iterations, and continuous response to change, but it doesn't say how long your iterations should be or how your response to changes should be.

On the other hand, a software process methodology is a specific way of conducting a software project, like the Rational Unified Process and Scrum. They define exactly what, when, and/or how various artifacts are produced. They might not be entirely explicit with all regards. For example, Scrum doesn't identify what documents to produce or not to produce, since its focus is on delivering value to the customer. But they define, in some way, the actions that members of the project team must follow.

Many models and methodologies of Software Engineering have been theorized. Early examples of the software development methodologies were the Waterfall model, Spiral, and Prototyping models. It is worth noting that these models have emerged out of need to control the development and carry it in a systematic manner. Some of them have emerged to help overcome the challenges that earlier models couldn't; such as, rapid delivery, or excess documentation overhead.

It is important to stress on a pivotal point. There is no such this as the "one for all" solution when it comes to software development. Each model can be refined and customized to meet the needs and standards of the development team. In some cases, a model can be a hybrid model of many, or a tailored cut of one model. It just needs to meet the standards of the management and the requirements of the client in order for it to work.

1.1 A General Overview for Software Engineering

According to Pressman in his book in 2014, Software Engineering moved into its fourth decade in the 90s. Also, throughout the industry, "software engineer" has replaced "programmer" as the job title of preference. Software process models, Software Engineering methods, and software tools have been adopted successfully across a broad spectrum of industry applications (Pressman, 2014). The history of

Software Engineering begins from a traditional procedural approach that is the first methodology in the software world. Then object-oriented and component based approaches came into this world and brought many new useful features.

Traditional methods for the analysis and design of computer-based systems have now been promoted for more than 40 years. Many of the organizations, which deal with the design and construction of computer-based systems apply traditional system development methods with varying degrees of success and there are still a great many system developers who do not use traditional methods at all, though some of these have attempted to introduce methods into their work practices. Where some organizations may have found increased benefits from the adoption of such methods, others have met only with dismay and failure (Kautz, 1999). Traditional methods relied on defining the system as two separate entities: Data and Function entities. These two entities would go on being traditional, planned, and controlled separately during the process of development. This would result in a solution that might access duplicates of data in different functions in a function oriented design, or a data that use duplicate functions in a data oriented design.

This dispersion of data and function in traditional design created the need for a new paradigm in Software Engineering resembling the new –back then- Object Oriented paradigm in implementation languages. This paradigm compresses the data and function in a single entity called objects. It defines the behavior of these objects internally, and in relation to other objects or users in the developed system.

1.2 The Aim of the Project

The main aim of this research is to focus on identifying the steps in shifting from a traditional procedural design to an Object Oriented (O-O) design. The steps are not meant only for Software Engineering professionals, but also for programmers with little Software Engineering principles knowledge. The goal is not to fully redesign the system, but to at least have the minimum requirements, which are the objects.

Also, this project aims to apply the proposed approach on a Social Network Analysis system in an Omani organization referred to as "Markaz". This could be achieved by

concerned with redesigning the application.The thesis at first provides background for O-O software engineering first and then it explains the implementation. The proposed detailed steps will serve into identifying the objects needed for redesigning the system into the O-O paradigm. The future work would take that a step forward to derive more of the current system in regards of diagrams and other needed documentation instead of creating it from scratch.

As a case study, only one module of Markaz's SNA tool will be redesigned using the O-O detailed steps. Since Markaz environment is a private and closed one, this research skips searching for ready-made objects. Therefore, each required object is developed by using .NET frame work 4.0 In addition; GUIs (Graphical User Interface) of each various system are developed using this approach and the MS Visual Studio 2010 IDE and MS Expressions Blend. Meanwhile, although hardware and software architectures of Markaz are described in the case study chapters such as Chapter 4 and 5, specific vendors, providing technology for Markaz, software codes and database schemas are not mentioned for security reasons.

The phases in the research methodology are shown in Table 1.1

Table 1.1 Research Methodology phases

Phase	Steps	Duration	Notes
Proposal	N/A	17-20 July 2012	
Literature Review	1.Search for literature related to Procedural Design 2.Search for literature related to O-O desing	21 July 2012 – 21 November 2012	
Literature Analysis	N/A	1 December 2012 -1 March 2013	
Problem Definition	1.Define problem scope. 2.Define problem parts.	2 March 2013 – 1 May 2013	
Develop Solution	1.Define Solution parameters. 2.Develop solution idea and approach	5 May 2013 – 1 September 2013	
Case Study	1.Apply solution on case study 2.Evaluate Findings	15 September 2013 – 5January 2014	
Write up and validation	N/A	10 January 2014 –15 May 2014	

1.5 Organization of the Thesis

Chapter 2 explains Software Engineering approaches in general. Chapter 2 also describes the various approaches and process models for all these approaches are described. Chapter 3 describes O-O development and its basic concepts, tools, modeling languages, technologies, and promises. Chapter 4 describes the current structure of Markaz SNA System in terms of both hardware and software architectures. Chapter 5 describes each step of implementation of the personal module in Markaz's SNA system. Chapter 6 is reserved for Conclusions and Suggestions in which has some suggestions for future of Markaz's SNA system and some inferences about Object Oriented software engineering studies.

Chapter 2: Software Engineering Approaches

This Chapter describes in detail three major approaches in Software Engineering such as traditional, component-based, and object-oriented approaches. It describes these approaches in details and some of the process models used for development.

2.1 Various Approaches for Software Engineering

2.1.1 Traditional Approach

This approach contains basic steps of a software development process such as analysis, design, implementation, testing, and maintenance. This thesis focuses on only analysis and design phases and their detailed steps. Each of the steps of the analysis phase (Pressman, 2014) provides information that is required to create a design architecture. The flow of information during software design is illustrated in Figure 2.1. For specifying software, this approach offers some variety of elements such as a data dictionary, data flow diagrams, state transition diagrams, entity-relationship diagrams, process specifications, control specifications, and data object descriptions for analysis phase. The design phase produces a data design, an architectural design, an interface design, and a procedural design with the help of various methods and techniques such as transaction mapping and transform mapping for architectural design and structured programming, graphical design notation, tabular design notation, and program design language for procedural design. Brief descriptions of some favorite elements of analysis and design models are mentioned below (Brooks, 1987):

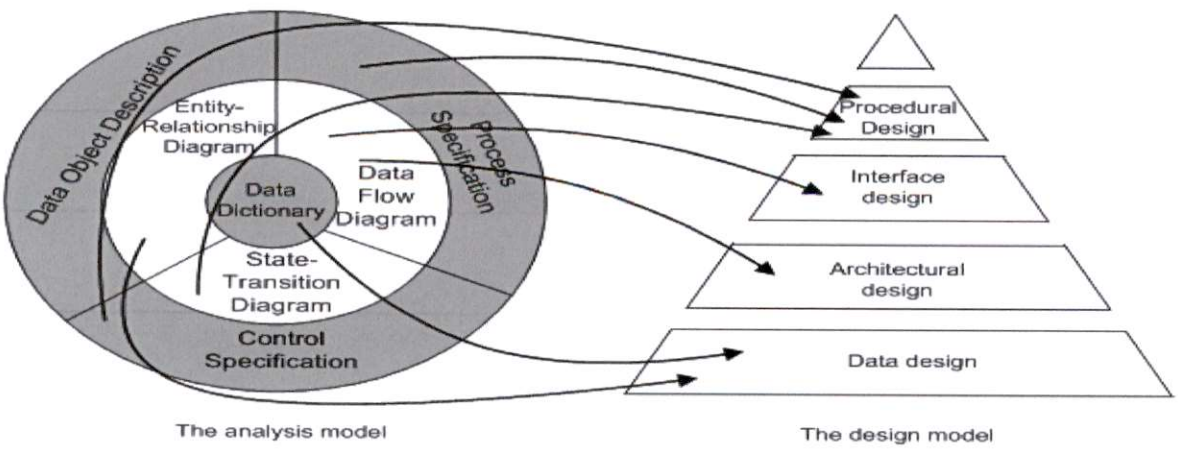

Figure 2.1 Mapping between design steps and related techniques

Data flow diagrams represent the transformations of data as it flows through a system and are the focus of SA/SD (Structured Analysis/ Structured Design). A data flow diagram consists of processes, data flows, actors, and data stores. Starting from the top-level data flow diagram, SA/SD recursively divides complex processes into sub diagrams, until many small processes are left that are easy to implement. When the resulting processes are simple enough, the decomposition stops, and a process specification is written for each lowest-level process. Process specifications may be expressed with decision tables, pseudo code, or other techniques.

The data dictionary contains details that cannot be included from data flow diagrams. The data dictionary defines data flows and data stores and meaning of various names. State transition diagrams illustrate time dependent behavior. Most state transition diagrams describe control processes or timing of function execution and data access triggered by events.

Entity-relationship (ER) diagrams highlight relationships between data stores that otherwise would only be seen in the process specifications. Each ER data element corresponds to one data flow diagram data store. In design phase, the most favorite technique is structured programming to produce procedural design. It is performed by languages such as Pascal, Ada and C. The broad definition of structured programming refers to any software development technique that includes structured design and results in the development of a structured program. Structured programming allows programs to be broken down into blocks or procedures, which can be written without detailed knowledge of the inner workings of other blocks. Thus allowing a top-down design approach or stepwise refinement (Brooks, 1987). Large-scale systems, built using this approach, are often deployed on only mainframes and minis. They feature as mainframe-based or other non-relational database systems. Therefore, both feeling the heat of competition, and simply looking for ways to improve software development can be the reason for moving into object-oriented approach in industry (Kautz, 1999).

2.1.2 Component-Based Approach

This approach is expected to revolutionize the development and maintenance of software systems. The Gartner Group, for example, estimates that "... by 2003, 70% of new applications will be deployed as a combination of pre-assembled and newly created components integrated to form complex business systems." The resulting increase in reuse should dramatically improve time-to-market, software lifecycle costs, and quality (Atkinson et al., 2005). In this section the emphasis is how this approach and its seed, CBD, are taken from previous approaches and intermediary approaches such as distributed objects and distributed systems. Figure 2.2 depicts the transformation that occurs after object-oriented approach. With distributed object approach extends the object-oriented approach with the ability to call objects across address space boundaries, typically using an "object request broker" capability. The distributed system approach is a development approach for building systems that are distributed, and are often represented as multi-tier.

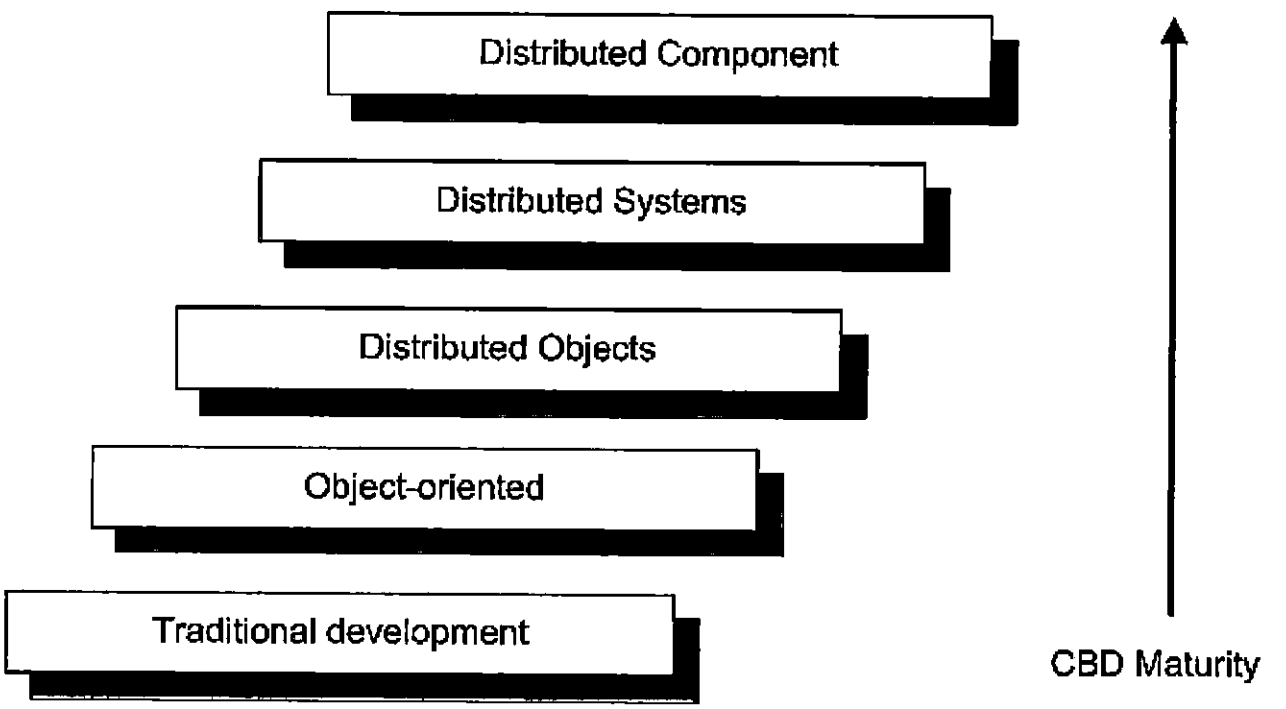

Figure 2.2 CBD maturity phases

Many companies today claim to be following component-based development when what they are really following is distributed system development, or using some kind of distributed object technology. While this can deliver significant benefits, such as allowing the technical bridging of heterogeneous systems, it does not decrease the cost of development. It is slightly addressed by using object-oriented techniques but not enough to make a big difference. At this point, distributed component approach is embraced in industry to reap the desired benefits, often looked for by a software development organization. This uses current build-time and run-time technologies such as Enterprise Java Beans that is attempted to reduce cost and development time for distributed systems. It becomes apparent that what is needed is something that addresses both the challenge of distributed systems interoperability and the challenge of how to build individual systems that can be treated as atomic units and can easily be made to cooperate with each other (Kautz, 1999).

Despite the industrial evolution, mentioned above, component-based technology introduces abstraction and lower-level mechanisms but has to be orchestrated into a comprehensive Software Engineering process (Dorgu and Tanik, 2003).

2.1.3 Object-Oriented Approach

The steps of software development mentioned above are common for all Software Engineering approaches. Therefore, analysis and design phases are inevitable for object-oriented approach, as well. In this approach, design is divided into four different steps as illustrated in Table 2.1.

Table 2.1 Analysis and design phases for Object-Oriented Approach

Phase	Techniques	Key Deliverables
Analysis	• Collaboration Diagrams • Class and Object models • Analysis Modeling	Analysis Models
System Design	• Deployment Modeling • Component Modeling • Package Modeling • Architectural Modeling	Overview design and Implementation architecture
Class Design	• Class and Object Modeling • Interaction Modeling • State Modeling • Design Patterns	Design Models

Object-oriented approach promises a way for implementing real-world problems to abstractions from which software can be developed effectively. It is a sensible strategy to transform the development of a large, complex super-system into the development of a set of less complicated sub-systems. Object-orientation offers conceptual structures that support this sub-division. Object-orientation also aims to provide a mechanism to support the reuse of program code, design, and analysis models (Hill and McRobb, 2002).

This approach uses classes and objects as the main constructs from analysis to implementation. It normally involves using an Object-Oriented language such as C++ or Java that provides (build-time) encapsulation, inheritance and polymorphism, and the ability for objects to invoke each other within the same address space. This last point is an important constraint when it comes to distributed systems.

UML (Unified Modeling Language) contains a number of concepts that are used to describe systems and the ways in which the systems can be broken down and modeled. The UML Specification defines the terms class and object as follows (Hill and McRobb, 2002):

- A class is a description of a set of objects that share the same attributes, operations, methods, relationships and semantics. Moreover, the purpose of a

class is to declare a collection of methods, operations and attributes that fully describe the structure and behavior of objects.

- An object is "an instance that originates from a class. It is structured and behaves according to its class.' Interface is another important construct defined as a group of externally visible operations. The interface contains no internal structure; it has no attributes, no associations, and only abstract operations."

In object-orientation, three main principles are important. Encapsulation, which is also known as information hiding, provides the internal implementation of the object without requiring any change to the application that uses it. The ability of one class of objects to inherit some of its properties or methods from an ancestor class is named inheritance in object technology. Polymorphism is producing various results for a generalized request based on the object that is sent to.

In Object-Oriented Approach objects of software can be (Brooks, 1987):

- External entities: printer, user, sensor
- Things: reports, displays
- Occurrences or events: alarm, interrupt
- Roles: manager, engineer, salesperson
- Organizational unit: team, division
- Places: manufacturing floor
- Structures: employee record

In this approach de facto standard notation (Hill and McRobb, 2002), UML, reveals analysis and design phases of software development.

Use cases specify the functionality that the system will offer from the users' perspective. They are used to document the scope of the system and the developer's understanding of what it is that the users require.

Classes might interact to deliver the functionality of the use case and the set of classes is known as collaboration. Collaborations can also be represented in various ways that

العنوان:	Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design
المؤلف الرئيسي:	الشندودي، أحمد بن سيف بن محمد
مؤلفين آخرين:	Al Khanjari, Zuhoor Abdullah Salim(Advisor)
التاريخ الميلادي:	2014
موقع:	مسقط
الصفحات:	1 - 95
رقم MD:	972147
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة السلطان قابوس
الكلية:	كلية العلوم
الدولة:	عمان
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، تصميم البرمجيات، المنهجية التسلسلية التقليدية، برمجة تحليل الشبكات الاجتماعية، إعادة التصميم، هندسة البرمجيات الموضوعية
رابط:	https://search.mandumah.com/Record/972147

Chapter 6: Conclusion and Future Work

The Simplified O-O Redesign Approach is a process which extracts the objects from a procedural design. The object repository which contains these objects forms a basis for any O-O design of the system at hand. The repository represents the reference for the needed object in any current or future system design. It adds the following advantages to the design and the development processes:

- **The objects represent organizational business logic:** The Organizational category in the repository represents the stratigical and long term interests of the organization. They add more value to the design and hence to the system when it's implemented.
- **Standard design of object:** Since the repository of objects is the single source of designed objects, the design is then standardized. The new systems will adhere to the designs of the objects being reused.
- **Less development time:** The repository holds objects which represent real life entities. Their reusability is enhanced by the fact that they represent the organization's interests. Even the system specific objects can be used in designing new systems if they share the same problem domain or parts of it. This cuts down the time needed for designing the same objects. Any enhancements can be achieved through O-O principles, like:
    - Inheritance for more specialization
    - Encapsulation to collaborate different objects to extend their use.
- **Less system integration issues:** The standard design imposed by the repository helps the system communicate the data of concern by using standard designed objects. This minimizes the intermediate processing to communicate the data.

The Proposed Approach meets the three requirements of the required solution for the redesign, which are:

1. **Starts at the design level:** It concentrates on the design of the system rather than leave the redesign to a later stage like the implementation phase. This adds to the quality of the solution. Unlike the Refactoring AntiPattren, which concentrates on the implementation of the system rather than the design. The

Refactoring AntiPattren considers the shortcomings of the procedural design as an implementation issue, where they are a design issue which affects the way the problem is perceived. The MOF's MDA models a solution from scratch, it doesn't start at the design level.

2. **It incorporates an organizational view of the problem:** By identifying the organizational interests, and building the objects of the system around these interests. The Simplified O-O Redesign Approach broadens the problem domain to incorporate the strategically valuable entities to the organization in which the system is used. This adds to the value of the solution to the organization, and adds to the ability of the system to add to the organization's goals. It has the advantage over both MOF's MDA and the Refactoring AntiPattren. In MOF's MDA the problem domain is localized to the problem at hand because it doesn't consider the strategical view of the organizations using the modeled systems.
3. **It introduces Reusability:** The Simplified O-O Redesign Approach introduces reusability on three levels:
    - **The Organizational level:** By using the object designs in the organizational category of the repository. This ensures satisfying the startigical interests of the organization in every system.
    - **The System Level:** By reusing the object designs in solution for similar problem domains or which have to interoperate.
    - **The Utility level:** This is a level concerned with technical considerations. The platform of the organization is usually the same for all systems, so it is produced to reuse ready designs from the repository to meet such issues.

    The Reusability introduced by the Simplified O-O Redesign Approach introduces standards in design. This is largely due to the central repository of object designs from which these objects are reused. None of the two other approaches introduce such reusability of the three levels. Both Refactoring AntiPattren and MOF's MDA introduce reusability on a system level, because they do not consider the organizational view of the system.
4. **It is easy to use by development teams:** The complex notations and the formal approach of the O-O approach are not used in the proposed approach

(Simplified O-O Redesign Approach), so the learning curve for is relatively low. The proposed approach uses basic O-O concepts and relies on object definitions to draw the O-O design in latter stages after the initial hard step of identifying the objects is achieved. This makes the proposed approach superior and more appealing to development teams than MOF's MDA.

Table 6.1 Comparison between Simplified O-O ReDesign Approach, MOF' MDA and Refactoring AntiPattren

	MOF'S MDA	Refactoring AntiPattren	Simplified O-O ReDesign Approach
Design Level Solution	★		★
Incorporates Organizational View			★
Introduces Reusability	★	★	★
Easy to Adopt by development teams		★	★

6.1 The Object Repository from another Perspective

The repository serves as the basis for the O-O designs in the organization. The objects represent the organization, the systems within the organization, and address some of the technical issues related to the organization's systems' architecture. The designs included in the repository help the development team in the implementation phase to use the Object Oriented Programming Language to its full potential. It doesn't leave the Object Orientation to the latter stages of development and treat it as an implementation issue. This has a huge impact on the way implementation is carried on in any Software Engineering approach. The repository can be used as the basis for designs using other approaches, such as: Service Oriented or Component Based Software Engineering.

If the development team decides to expose some of the aspects of a system as a service, the designs included in the repository can be used to create such a service.

The interface parts of the system needed to be used without exploiting the parts which are supposed to be hidden from the outside world.

The same concept applies to the Component Based Approach. Components are collection of objects, and the designs in the repository can be grouped together to form ready to use, and standardized components without having to design the parts which are present at the repository. (Wang, 2009)

The repository doesn't limit the end solution to the O-O approach. It is a basis for it. In the same time, it makes use of the powerful O-O principles which uses the Object Oriented Programming Languages to their full potential without being confined to the O-O approach in the latter stages of the design.

6.2 Future Work

Future work will concentrate on formalizing a second version of the Approach for the use of Software Engineering professionals. The process of formalizing the process will concentrate on two aspects:

- **The process model:** The Approach will concentrate on the O-O approach's phases and take the re-engineering process in the design stage and expand it to other stages. So it will define an approach to re-engineer a system on different levels starting from requirements and including the maintenance process. This will take into account the different phases iterations in the O-O approach: inception, elaboration, construction, and transition.
- **The product model:** The work will concentrate on translating the ready products of the procedural approach to products of the O-O nature. It will formulate a framework in which a product model of the procedural paradigm is changed to its closest equivalent in the O-O paradigm then deriving the rest of the O-O product model. For example, deriving Class Diagrams from Data Flow Diagrams, and then deriving Entity Relationship Diagrams from the result. This will make use of the Platform Independent Modeling in MOF standards. So it can make use of the modeling tools provided for the MOF's MDA for automatic generation.

The work will concentrate also on deriving the O-O design from different approaches, like: Component Based or Service Oriented. This will prove beneficiary if any of the latter approaches have design issues. The equivalent O-O design will dissemble the main entities in their designs and will help reconstruct a new design after addressing the issues. The new Design can be either in O-O approach or using the original approach of the system.

العنوان:	Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design
المؤلف الرئيسي:	الشندودي، أحمد بن سيف بن محمد
مؤلفين آخرين:	Al Khanjari, Zuhoor Abdullah Salim(Advisor)
التاريخ الميلادي:	2014
موقع:	مسقط
الصفحات:	1 - 95
رقم MD:	972147
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة السلطان قابوس
الكلية:	كلية العلوم
الدولة:	عمان
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، تصميم البرمجيات، المنهجية التسلسلية التقليدية، برمجة تحليل الشبكات الاجتماعية، إعادة التصميم، هندسة البرمجيات الموضوعية
رابط:	https://search.mandumah.com/Record/972147

References:

Al-Ahmed, W. (2006), Object-Oriented Design Patterns for Detailed Design, *Journal of Object Technology*, 5(2), pp. 11-17.

Atkinson, C., Bayer, J., Laitenberger, O., and Zettel J. (2005) , Component-Based Software Engineering: *The KobrA Approach*, [online] http://se2c.uni.lu/tiki/se2c-bib_download.php?id=700, [accessed on: 01/12/2013]

Barton, R. (1997), Report on a conference sponsored *by the NATO Science Committee Report*, Garmisch, Germany, 7th to 11th October 1997, 3(7), pp. 23-28.

Basili, V., Briand, L., and Melo, W. (1996), How reuse influences productivity in object-oriented systems. *Communications of the ACM,* 1(9), pp. 2-8.

Batory, D. and Smaragdakis, Y. (1997), DiSTiL: a Transformation Library for Data Structures, *USENIX Conference on Domain-Specific Languages (DSL),2(31),* San Fransisco, USA, 22nd to 25th May, 4(5), pp. 12-16.

Boehm, B. W. (1976) , Software Engineering: IEEE Transactions on Computers, C-25, 12, pp. 1216-1241.

Boehm, B. W. (1988), A Spiral Model of Software Development and Enhancement, 5/1988, 1(6), pp. 61-72.

Brooks F. (1987), No silver bullet, *IEEE Computer*, 20(4), pp. 29-38.

Buyya, R. (2009), Object Oriented Programming with Java: Essentials and Applications, Tata McGraw-Hill Education.

Carlsen, I. C., and Haaks, D. (1992), Concept and Implementation of an Object-Oriented Framework for Image Processing, *Philips Journal of Research*, 46(6) pp,.311-340.

Chen, K. (2004), Comparison of Object Oriented and Procedure-Based Computer Languages, Issues in Information Systems, 5(1)

Craig, I. (1999), The interpretations of object oriented programming, Berlin, Springer-Verlag, 8(2), pp.17-25.

Dogru, H., and Tanik, M. (2003), A Process Model for Component- Oriented Software Engineering, *IEEE Software*, March/April, pp.31-52.

Eden, A. (2004), Principles in Formal Specification in Object Oriented Design and Architecture, Proc. Centre for Advanced Studies CONference—CASCON Report, 7(5), pp. 22-38.

Fayad, M., and Tsai, W. and Fulghum, M.(1996), Transition To Object-Oriented Software Development, *Communications of the ACM*, Vol. 39, pp. 2-9.

Floyd, C, Reisin, F, and Schmidt, G. (1989), Steps to software development with users, 2nd *European Software Engineering Conference*, University of Warwick, Coventry, Lecture Notes in Computer Science No. 387, Heidelberg, Springer, pp. 48 - 64.

Hill, B.and McRobb, F. (2002), Object Oriented System Analysis and Design (using UML), 2nd Edition, McGraw Hill.

Johnson, R.A. (2000), The Ups and Downs of Object-Oriented Systems Development, *Communications of the ACM*, 43(10), pp. 69-73.

Kahler, H., Stiemerling, O. and Wulf, V. (2000), How to make Software Softer - Designing Tailorable Applications, *Proceedings of the Second International Symposium on Designing Interactive Systems (DIS'00)*, ACM - Press, New York, pp. 365 – 376.

Kautz K. (1999), Introducing Structured Methods: An Undelivered Promise?, *Scandinavian Journal of Information Systems*, Oslo, Norway, 6(2), pp.59-78.

Kroll, P.and Kruchten, P. (2003), The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP. ISBN 0-321-16609-4.

Lewallen, R. (2005), [online] 4 major principles of Object-Orientation, http:// www. CodeBetter.com [accessed on: 12/10/.2013].

Mack, W. (2010), Fundamental Flaws In Procedural Designs, pp. 3-5.

Meyer, B. (1988), *Object-oriented software construction*, Prentice Hall, pp. 112-115.

Morch, A. (1997), Method and Tools for Tailoring of Object-oriented Applications: An Evolving Artifacts Approach, PhD-Thesis, University of Oslo, Department of Computer Science, Research Report , pp. 241.

Narzt, W., Pichler, J., Pirklbauer, K. and Zwinz M. (1998), A Reusability Concept for Process Automation Software, *Proceedings of 11th. International Conference Software Engineering,* Paris, Preprints , 2(6), pp. 73-81.

Object Management Group (OMG), (2014). OMG Specifications: MDA Specification, Retrieved from: [Online]http://www.omg.org/mof/.

Perry, D. and Wolf, A. (1992), Foundation for the Study of Software Architecture. *ACM Electronic Journal,3(11),* pp. 41-47.

Piefel, M. (1996), Information Engineering: Object Oriented Software Development, *Coursework 'Information Engineering'*, Department of Computing, University of Bradford, 1(2), pp. 12-19.

Pressman, R. (2014), "Software Engineering: A Practitioner Approach", McGraw Hill.

Schach, S. R. (1999), Classical and Object Oriented Software Engineering, 4th Edition, McGraw Hill.

Shvets, A. (2013), The BLOB, [online] http://sourcemaking.com/antipatterns/the-blob, [accessed on:12/01/2014]

Simons, A. (1994), Object-Oriented Analysis and Design, *Course Work on Object Oriented Software Engineering*, Department of Computer Science, University of Sheffield, 2(1), pp. 8-17.

Taft, D. (2002), IBM Acquires Rational, *EWeek*, 2(6), pp. 17-19.

Thomas, D. (2013), MDA: Revenge of the Modelers or UML Utopia*?*, *IEEE SOFTWARE,* pp. 22-24

Wang, J. (2009), Towards Component-Based Software Engineering, [online]http://delivery.acm.org/10.1145/360000/357729/p182-wang.pdf?key1=357729&key2=7337455011&coll=GUIDE&dl=GUIDE&CFID=36365381&CFTOKEN=91792709, [accessed on: 07/01/.2014]

Xiong, J. (2011), New Software Engineering Paradigm Based on Complexity Science, *Introduction to NSE, 3(9), pp.58-66.*

العنوان:	Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design
المؤلف الرئيسي:	الشندودي، أحمد بن سيف بن محمد
مؤلفين آخرين:	Al Khanjari, Zuhoor Abdullah Salim(Advisor)
التاريخ الميلادي:	2014
موقع:	مسقط
الصفحات:	1 - 95
رقم MD:	972147
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة السلطان قابوس
الكلية:	كلية العلوم
الدولة:	عمان
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، تصميم البرمجيات، المنهجية التسلسلية التقليدية، برمجة تحليل الشبكات الاجتماعية، إعادة التصميم، هندسة البرمجيات الموضوعية
رابط:	https://search.mandumah.com/Record/972147

Appendix A: SNA System Screens and Part Implementation

The tool is used to keep track of a person's information and social and professional connections. These connections are shown as connections to other people and organizations as well.

Figure A.1 Selection Screen

Figure A.1 is the login screen for the SNA tool where a user is authenticated against the database records.

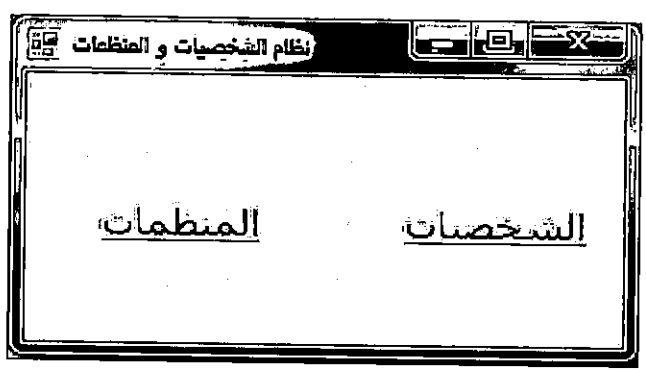

Figure A.2 Selection Screen

The Above screen (Figure A.2) is a selection screen where the user chooses to go to the Personal Information screen (Figure A.3) or the Organization screen (Figure A.8)

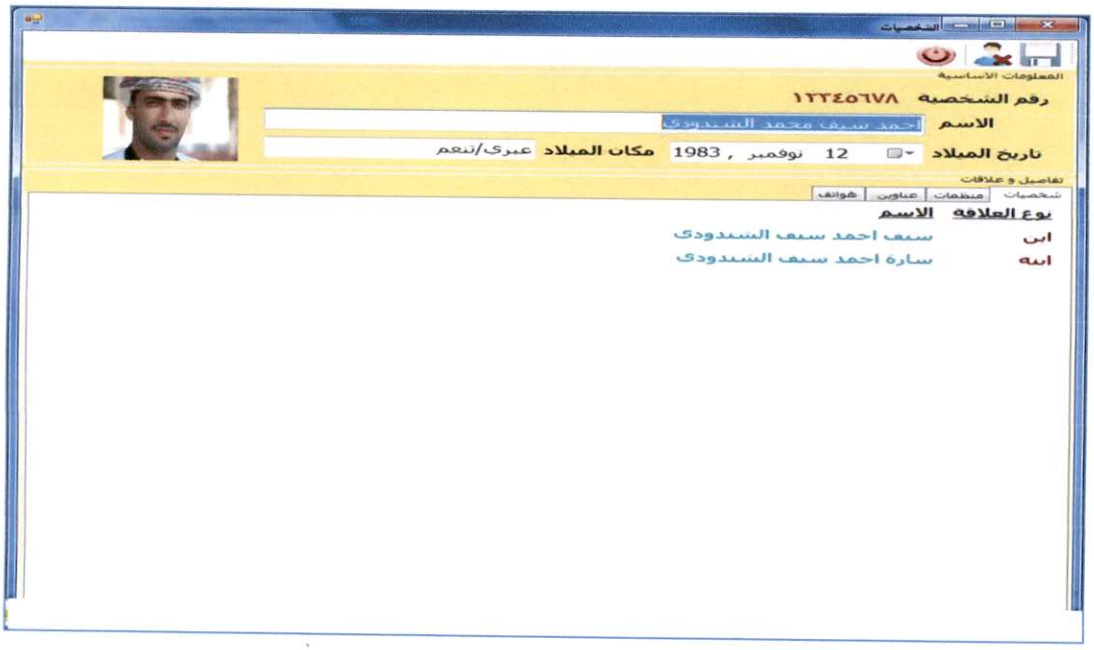

Figure A.3 Person Personal Relations

The Above screen (Figure A.3) is the main Personal screen with Personal relations information to other persons connected the loaded person.

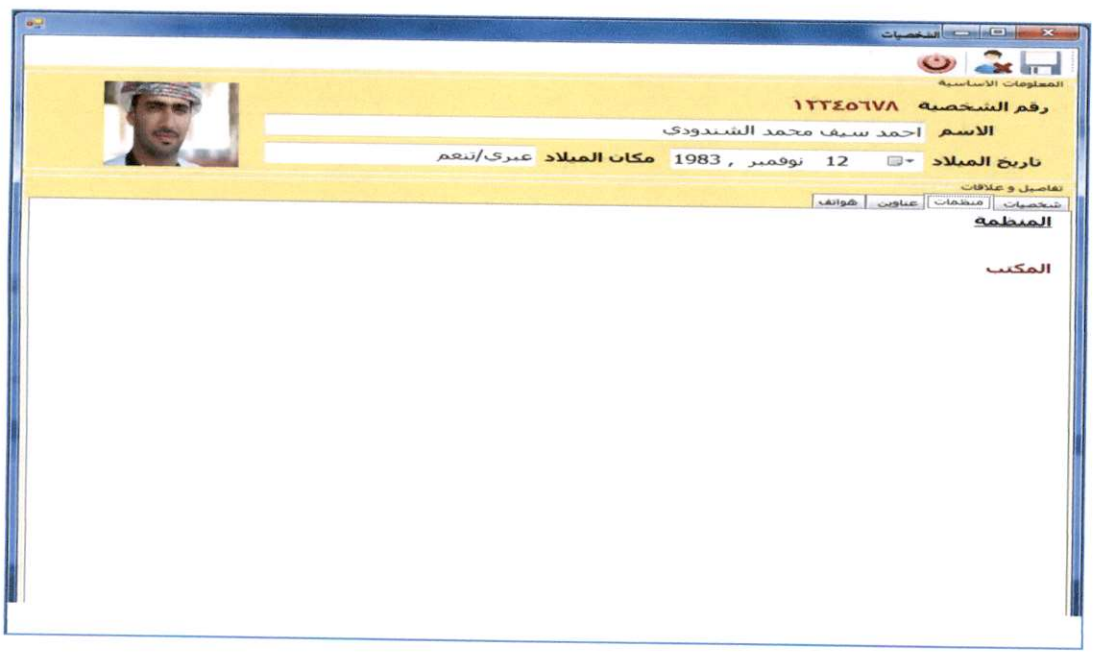

Figure A.4 Person Organization Relations

The Above screen (Figure A.4) is the main Personal screen with relations to Organizations information.

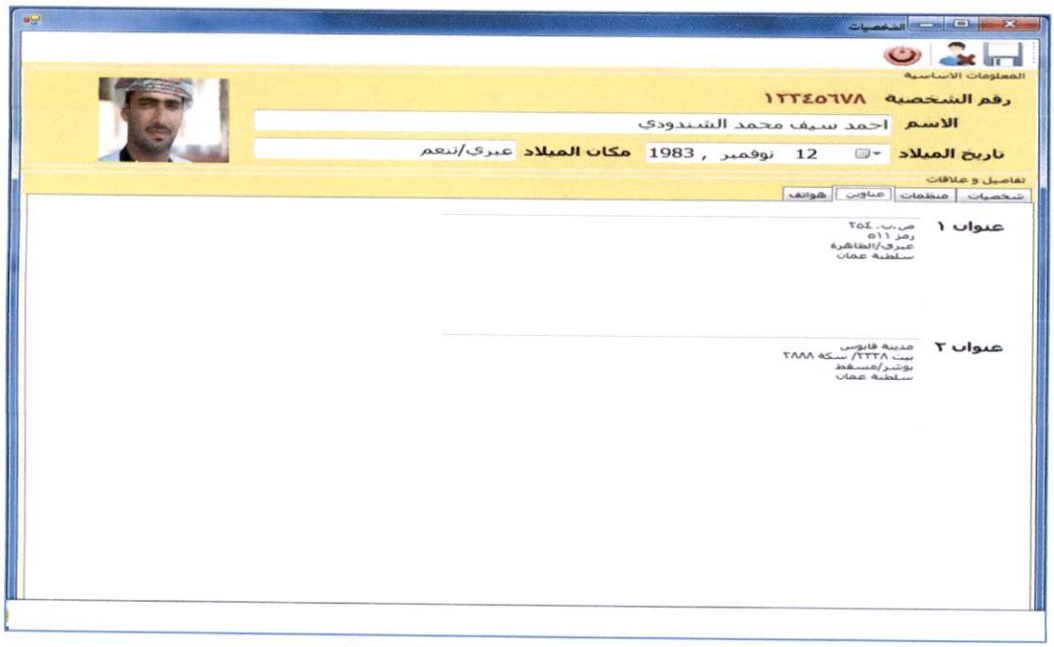

Figure A.5 Person Address Relations

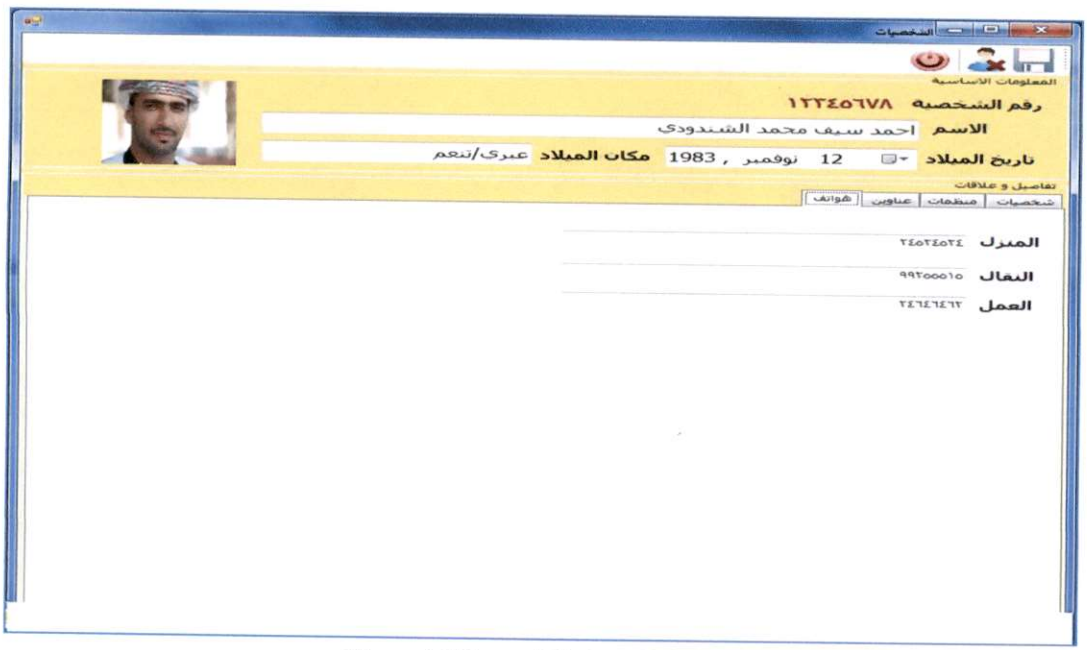

Figure A.6 Person Telephone Relations

The Above screens (Figure A.5, A.6) is the main Personal screen with relations to Addresses information ((Figure A.5) and Telephones relations (Figure A.6).

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class PersonForm
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.components = New System.ComponentModel.Container()
        Dim resources As System.ComponentModel.ComponentResourceManager = New System.ComponentModel.ComponentResourceManager(GetTyp
        Me.ToolStrip1 = New System.Windows.Forms.ToolStrip()
        Me.tsbSave = New System.Windows.Forms.ToolStripButton()
        Me.tsbDel = New System.Windows.Forms.ToolStripButton()
        Me.ToolStripSeparator1 = New System.Windows.Forms.ToolStripSeparator()
        Me.tsbExit = New System.Windows.Forms.ToolStripButton()
        Me.GroupBox1 = New System.Windows.Forms.GroupBox()
        Me.TextBox7 = New System.Windows.Forms.TextBox()
        Me.Label18 = New System.Windows.Forms.Label()
```

Figure A.7 Person Class

The Above screen is the implementation of the Person Module (Figure A.7)

Figure A.8 Organization Person Relations

The Figure A.8 shows the main Organization information screen with relations to persons in it.

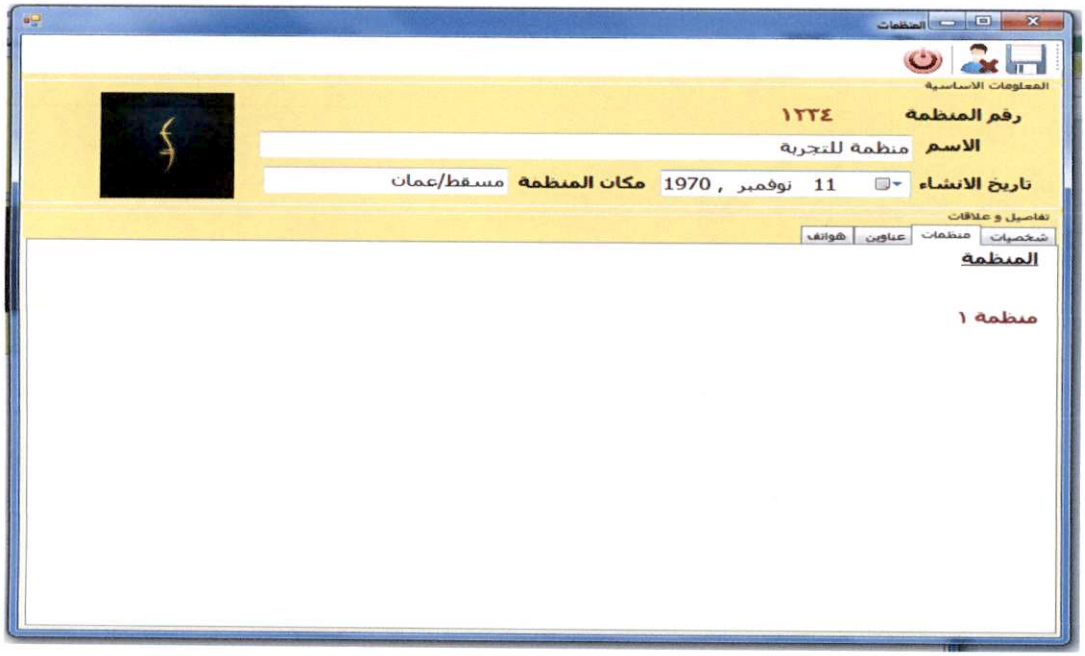

Figure A.9 Organization Organization Relations

The above screen (Figure A.9) is the organization relations with other organizations.

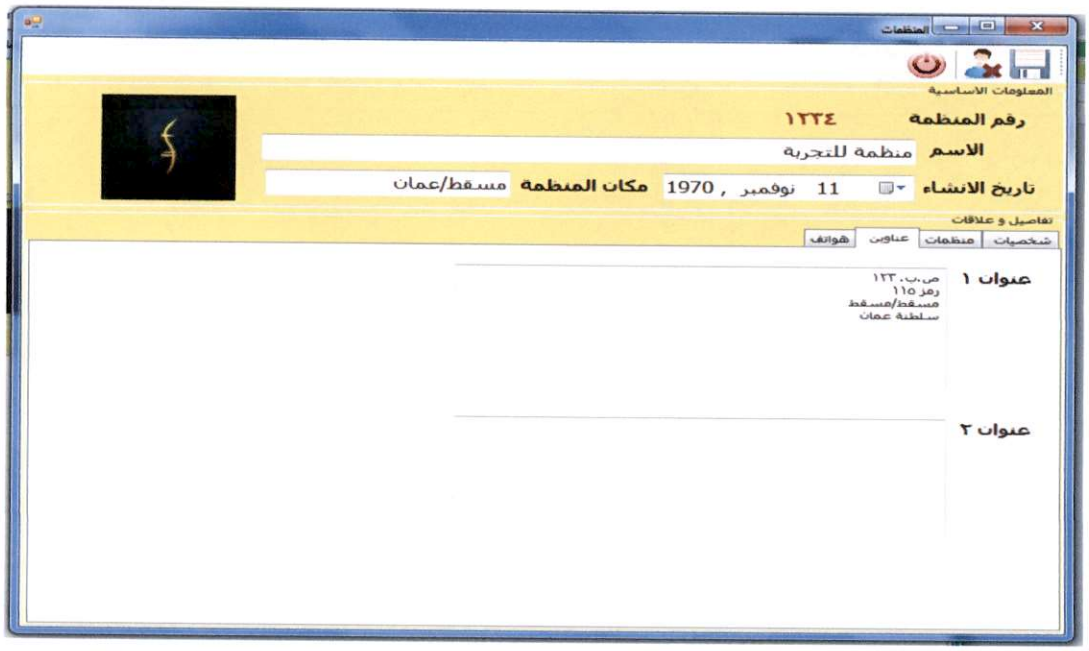

Figure A.10 Organization Addresses Relations

Figure A.10 shows the relations between the organization and the addresses.

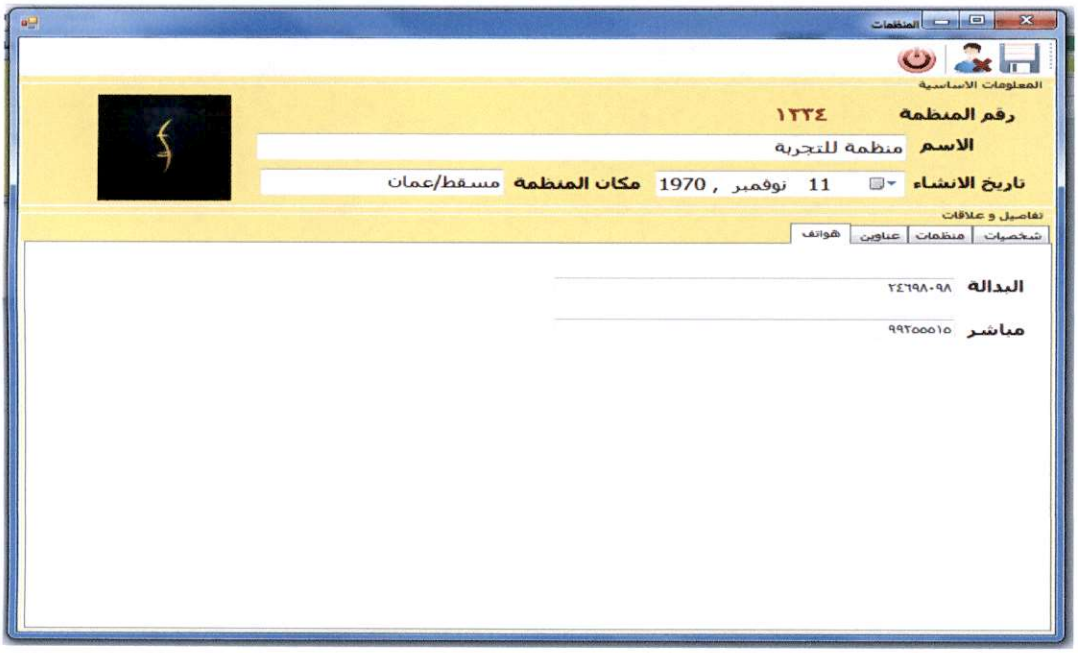

Figure A.11 Organization Telephone Relations

Above screen (Figure A.11) shows the relation between the organization and related telephones.

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class OrgForm
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.components = New System.ComponentModel.Container()
        Dim resources As System.ComponentModel.ComponentResourceManager = New System.ComponentModel.ComponentResourceManager(GetTyp
        Me.ToolStrip1 = New System.Windows.Forms.ToolStrip()
        Me.tsbSave = New System.Windows.Forms.ToolStripButton()
        Me.tsbDel = New System.Windows.Forms.ToolStripButton()
        Me.ToolStripSeparator1 = New System.Windows.Forms.ToolStripSeparator()
        Me.tsbExit = New System.Windows.Forms.ToolStripButton()
        Me.GroupBox1 = New System.Windows.Forms.GroupBox()
        Me.TextBox7 = New System.Windows.Forms.TextBox()
        Me.Label18 = New System.Windows.Forms.Label()
```

Figure A.12 Organization Class

Figure A.12 shows part of the Organizational module design.

العنوان:	Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design
المؤلف الرئيسي:	الشندودي، أحمد بن سيف بن محمد
مؤلفين آخرين:	Al Khanjari, Zuhoor Abdullah Salim(Advisor)
التاريخ الميلادي:	2014
موقع:	مسقط
الصفحات:	1 - 95
رقم MD:	972147
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة السلطان قابوس
الكلية:	كلية العلوم
الدولة:	عمان
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، تصميم البرمجيات، المنهجية التسلسلية التقليدية، برمجة تحليل الشبكات الاجتماعية، إعادة التصميم، هندسة البرمجيات الموضوعية
رابط:	https://search.mandumah.com/Record/972147

Simplified O-O Re-Design Approach: Deriving Object Oriented Design from Traditional Procedural Design

Ahmed Saif Mohammed Al-Shandoudi

A thesis submitted in partial fulfillment

of the requirements for the degree

Master of Science

in

Computer Science

Department of Computer science

College of Science

Sultan Qaboos University

Sultanate of Oman

2014