Chapter 1

Introduction

One of the largest application areas for computers is the storage and retrieval of data. We call this class of systems "information-based systems" or simply "information systems" because the primary purpose of such systems is managing information. Examples of information systems are banking systems, library-cataloguing systems, and personal information systems. At the heart of such systems is a database against which we apply transaction to create, retrieve, update, or delete items. Many systems provide a web interface to operate on the information[1].

Information systems have gained importance because of the increasing volume of information as resources. The data that these systems manage is often the most valuable resource of an enterprise. Such data concern both the processes and the resources internal to the enterprise-plants, goods, people, etc... as well as information on external sources-competitors, suppliers, clients, etc [1].

Information systems are data oriented that can be characterized on the basis of the way they treat data. The following are some of the qualities that characterize the information system

- **Data integrity.** Under what circumstances will the data be corrupted when the system malfunctions?
- **Security.** To which extent does the system protect the data from unauthorized access?
- **Data availability.** Under what conditions will the data become unavailable and for how long?

- **Transaction performance.** Because the goal of information systems is to support transactions against data, the performance of such systems can be uniformly characterized in terms of the number of transactions carried out per unit of time [1].

1.1 Types of information system

The applications of information systems in the real world can be classified in several different ways. For example, several types of information systems can be classified as either operations or management information systems. Figure 1.1 illustrates this conceptual classification of information systems applications.

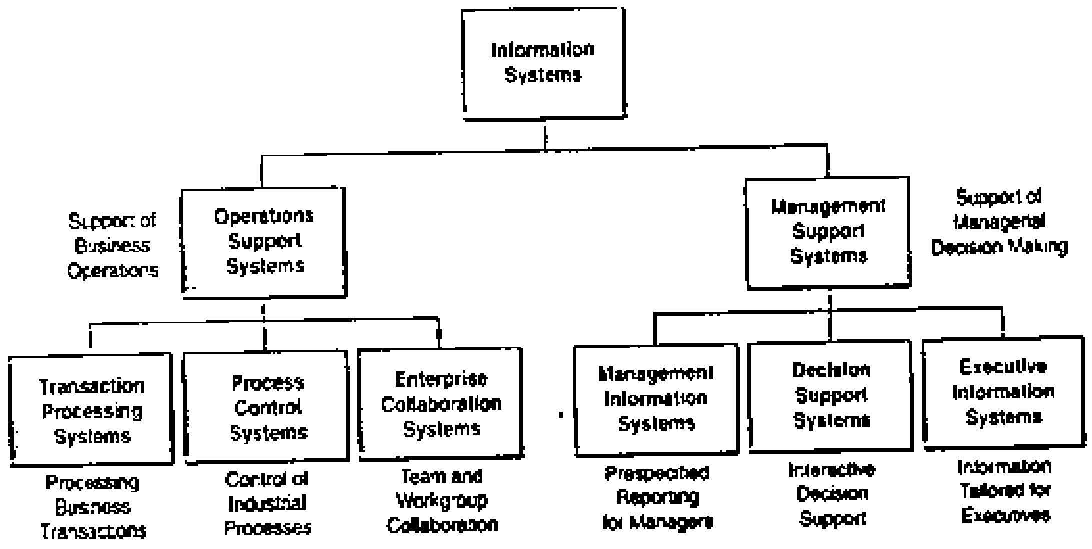
Information Systems
Support of Business Operators
Operations Support Systems
Management Support Systems
Support of Managerial Decision Making
Transaction Processing Systems
Process Control Systems
Enterprise Collaboration Systems
Management Information Systems
Decision Support Systems
Executive Information Systems
Processing Business Transactions
Control of Industrial Processes
Team and Workgroup Collaboration
Prespecified Reporting for Managers
Interactive Decision Support
Information Tailored for Executives

Figure 1.1 Operation and management classification of information systems

Information systems have always been needed to process data generated by, and used in, business operations. Such **operations support systems** produce a variety of information products for internal and external use. However, they do not emphasize producing the specific information products that can best be used by managers.

Further processing by management information systems is usually required. The role of a business firm's operations support systems is to efficiently process business transactions, control industrial processes, support enterprise communications and collaboration, and update corporate databases.

Transaction Processing System (TPS) are an important example of operations support systems that record and process data resulting from business transactions. They process transactions in two basic ways. In batch processing , transaction a data are accumulated over a period of time and processed periodically. In *real-time* (or online) processing, data are processed immediately after a transaction occurs. For example, Point-Of-Sale (POS) systems at many retail stores use electronic cash register terminals to electronically capture and transmit sales data over telecommunications links to regional computer centers for immediate (real-time) or nightly (batch) processing.

Process control systems monitor and control physical processes. For example, a petroleum refinery uses electronic sensors linked to computers to continually monitor chemical processes and make instant (real-time) adjustments that control the refinery process.

Enterprise collaboration systems enhance team and workgroup communications and productivity, and sometimes called office automation systems.

When information system applications focus on providing information and support for effective decision making by managers, they are called **management support systems.** Providing information and support for decision making by all types of managers and business professionals is a complex task. Conceptually, several major types of information systems support a variety of decision-making responsibilities: (1) management information systems, (2) decision support systems, and (3) executive information systems [2].

Management information systems that take information captured by transaction processing systems and produce reports that management needs for planning and controlling the business. Management information systems are possible because the information has been captured by the transaction processing systems and placed in organizational databases [3].

Decision support systems give direct computer support to managers during the decision-making process. For example, advertising managers may use an electronic spreadsheet program to do what if analysis as they test the impact of alternative advertising budgets on the forecasted sales of new products.

Executive information systems provide critical information from a wide variety of internal and external sources in easy-to-use displays to executives and managers. For example, top executives may use touchscreen terminals to instantly view text and graphics displays that highlight key areas of organizational and competitive performance [2].

1.2 Two approaches of system development

System development is done in many different ways. this diversity can confuse new employees when they works a system developers. sometimes it seems that every company develops information systems has its own approach. sometimes different development groups within the same company uses different approaches, and each person in the company may have his or her own way of developing the system.

All system developers should be familiar with two general approaches to system development, because they form the basis of virtually all methodologies, the traditional approach and the object-oriented approach.

1.2.1 The traditional approach

The traditional approach includes many variations based on techniques used to develop information systems with structured and modular programming. This approach is often referred to as structured system development. A refinement to structured development approach, is called Information Engineering (IE), a popular variation.

- **structured system development approach**

Structured analysis, structured design, and structured programming are the three techniques that make up the structured approach. Sometimes these techniques are collectively referred to as the Structured Analysis and Design Technique (SADT) , the structured programming technique developed in the 1960s. was the first attempt to provide guidelines to improve the quality of computer programs. The structured design technique was developed in the 1960s, to makes it possible to combine separate programs into more complex information systems. the structured analysis technique evolved in the early 1980s is to help clarify the requirements for the computer system to the developers before they designed the programs.

- **Structured Programming**

High-quality programs are not only to produce the correct outputs each time the program runs but also make it easy for other programmers to read and modify the program later. it programs need to be modified all the time. A structured program is one that has one beginning and one end. and each step in the program execution consists of one of three programming constructs:

1. A sequence of program statements.
2. A decision where one set of statements or another set of statements to the execute.
3. A repetition of a set of statements [3].

A complex system may be divided into simpler pieces called modules. A system that is composed of modules is called modular. The main benefit of modularity is that it allows the principle of separation of concerns to be applied in two phases: when dealing with the details of each module in isolation (and ignoring details of other modules) and when dealing with the overall characteristics of all modules and their relationships in order to integrate them into a coherent system. If the two phases are executed in sequence first by concentrating on modules and then on their composition. then we say that the system is designed bottom up; the converse when we decompose modules first and then concentrate on individual module design is top down design [1].

- **Structured design**

As information systems continued to become increasingly complex through the 1970s, each system involved many different functions. Each function performed by the system might be made up of dozens of separate programs. The structure design technique was developed to provide some guidelines for deciding what the set of programs should what each program should accomplish, and how the programs should be organized into a hierarchy.

Two main principles of structured design are that program modules should be designed so they are (1) loosely coupled and (2) highly cohesive. Loosely coupled means that each module is as independent of the other modules as possible, which allows each module to be designed and later modified without interfering with the performance of the other modules. Highly cohesive means that each module accomplishes one clear task. That way, it is easier to understand what each module does and to ensure that if changes to the module are required, none will accidentally affect other modules [2].

Information engineering is a refinement to structured development that begins with overall strategic planning to define all of the information systems that the organization needs to conduct its business (the application architecture plan). The plan also includes a definition of the business functions and activities that the systems need to support, the data entities about which the systems need to store information, and the technological infrastructure that the organization plans to use to support the information systems.

Each new system project begins by using the defined activities and data entities created during strategic systems planning. The activities and data are refined in the project progresses. At each step, the project team creates models of the processes, the data, and the ways they are integrated.

The information engineering approach refines many of the concepts of the structure approach into a rigorous and comprehensive methodology. Both approaches define information systems requirements, design of information systems, and construct information systems by looking at processes, data, and the interaction of the two.

This text merges key concepts from these two approaches into one, which we will refer to hereafter to traditional approach. The traditional approach. in one version or another, is still widely for information system development. although many information systems project. Using Object-Oriented (OO) technology-which requires a completely different approach [6].

1.2.2 The object-oriented approach

An entirely different approach to information systems. the object-oriented approach view an information system as a collection of interacting objects that work together to accomplish tasks [2]. This means that objects in a computer system, like objects in the world around us, are view as thing. These things have certain features, or attributes, and they can exhibit certain beavers [7]. There are no processes or programs; there are no data entities or files. The system consists of objects, an object is a thing in the computer system that is capable of responding to messages, this radically different view of a computer system requires a different approach to systems analysis, systems design, and programming.

Because the object-oriented approach views information systems as collections of interacting Object-Oriented Analysis (OOA) defines all of the types of objects that do the work in the system and shows how the objects interact to complete tasks. Object-Oriented Design (OOD) defines all of the additional types of objects necessary to communicate with people and devices in the system and refines the definition of each type of object so it can be implemented with a specific language or environment. Object-Oriented programming (OOP) consists of writing statements in a programming language to define what each type of object does [3].

What is different about the object-oriented approach? The object-oriented approach is based on a view of computer systems that is fundamentally different from that of the structured approach. Once the business events involving the system have been identified. defining what the user requires means defining all of the types of objects that are part of the user's work environment (object-oriented analysis).

Second, to design a computer system means to define all the additional types of objects involving the user interface and operating environment of the computer system and the ways they interact with objects in the user's work environment (object-oriented design). Third, programmers must write statements that define all types of objects, including their attributes and behaviors (object-oriented programming). Therefore, in some ways, everything is different with the object-oriented approach [7].

1.3 Research objectives

This research have several objectives which can be summirized as follows:.

- Understanding of large system design such industrial information systems for which all software design of large scale are applied.
- Understanding and applying the different phases of system developments life cycle as applied to large scale system development and implementation.
- Use the object oriented software engineering to provide a system requirement definition and system requirement specification using some evolutionary and throw-away prototyping models to satisfy user needs.
- Useing Unified Modeling Language(UML) in requirement specification and in system design and implementation.
- Study of life cycle of software engineering project and applying knowledge that has been learned in the field.
- Compare the design approach using OOD and OOP with that obtained using structural programming and how the new system approach can be used to replace the existing system with as less changes as possible.
- Among the different software life cycles, the most appropriate life cycle that fits the needs for such systems will be selected and used to provide a reasonable reasoning for selecting such a model.

1.4 Thesis Organization

The thesis consist of 6 chapters. Chapter 1 includes introduction to the information management system.

In chapter 2 litrature review of object-oriented analysis and design with UML.

In chapter 3 system analysis and spcification of industrial information system.

In chapter 4 system designing of the industrial information system.

Im Chapter 5 about coding testing unites of the system as well as the integrated testing of the system and implementation.

Finally Chapter 6 discusses the conclusions and future work that can carried out on industrial information system.

العنوان:	Object oriented sottwore engineering approach to industriol intormotion system design and implementation
المؤلف الرئيسي:	مانيطة، عيسى حسين أحمد
مؤلفين آخرين:	الفقيه، إدريس ساسـ(مشرف)
التاريخ الميلادي:	2008
موقع:	سرت
الصفحات:	1 - 78
رقم MD:	859171
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة التحدي
الكلية:	كلية العلوم
الدولة:	ليبيا
قواعد المعلومات:	Dissertations
مواضيع:	نظم المعلومات، هندسة البرمجيات، البرمجيات الشيئية، نظم المعلومات الصناعية
رابط:	https://search.mandumah.com/Record/859171

Chapter 2

Literature review

2.1 Overview

The System Development Life Cycle (SDLC) is a widely used framework for organizing and managing the process. A system development life cycle typically defines phases that are completed by the project team as they move from the beginning to the end of the development project. The term life cycle is used because every information development project has a beginning and eventually an ending. Between these points in time. the project "lives" in one form or another.[7]

The object modeling approach prescribes the use of methodologies and diagramming notations that are completely different from the ones used for data modeling and process modeling. In the late 80s and early 90s many object-oriented methods were being used throughout industry. The existence of so many methods and associated modeling techniques was a major problem for the object-oriented. system development industry. It was not uncommon for a developer to have to learn several object modeling techniques depending on what was being used at the time. This use of different object modeling techniques limited the sharing of models across projects (reduced reusability) and development teams. It hampered communication between team members and the users, which led to many errors being introduced into the project. These problems and others led to the effort to design a standard modeling language[13].

The Unified Modeling Language (UML) is a set of modeling conventions that is used to specify or describe a software system in terms of objects [6].

The UML does not prescribe a method for developing systems-only a notation that is now widely accepted as a standard for object modeling. The Object Management Group (OMG), an industry standards body, adopted the UML in November 1997 [8].

Because the object-oriented approach views information systems as collections of interacting Object, Object-oriented Analysis (OOA) defines all of the types of objects that do the work in the system and shows how the objects interact to complete tasks. Object-Oriented Design (OOD) defines all of the additional types of objects necessary to communicate with people and devices in the system and refines the definition of each type of object so it can be implemented with a specific language or environment. Object-Oriented Programming (OOP) consists of writing statements in a programming language to define what each type of object does [3].

2.2 The unified doftware development process

The Unified Software Development Process (in short, Unified Process, or UP) has its roots in the industrial experience within Ericsson manufacture, A manufacturer of telecommunication equipment, in the late 1960s. Subsequently, that experience led to successor methodologies developed by two other companies: Objectory and Rational. gained the status of a widely adopted industrial standard [1].

The Rational Unified Process (RUP) is an object-oriented system development methodology offered by Rational Software. RUP is their attempt to define a complete methodology that, in addition to providing several unique features, uses UML for system models. In RUP, the term development process is synonymous with development methodology [3].

UP uses the unified modeling language throughout the software life cycle. As we have observed, the rich collection of languages that constitute UML provides specific notations to specify, analyze, visualize, construct, and document the artifacts that are developed in the life cycle of a software system. Software engineering is the notations to produce standardized blueprints that contain a number of different diagrams, each enlightening a certain aspect of the application being developed. UML is, however, an important step in direction of a standardized language that would allow all software engineers to communicate and interact on rigorous grounds while developing or evolving a given software system [1].

2.3 The phases of the unified process

UP consist of four phases: Inception , Elaboration, Construction, and Transition (see figure 2.1).

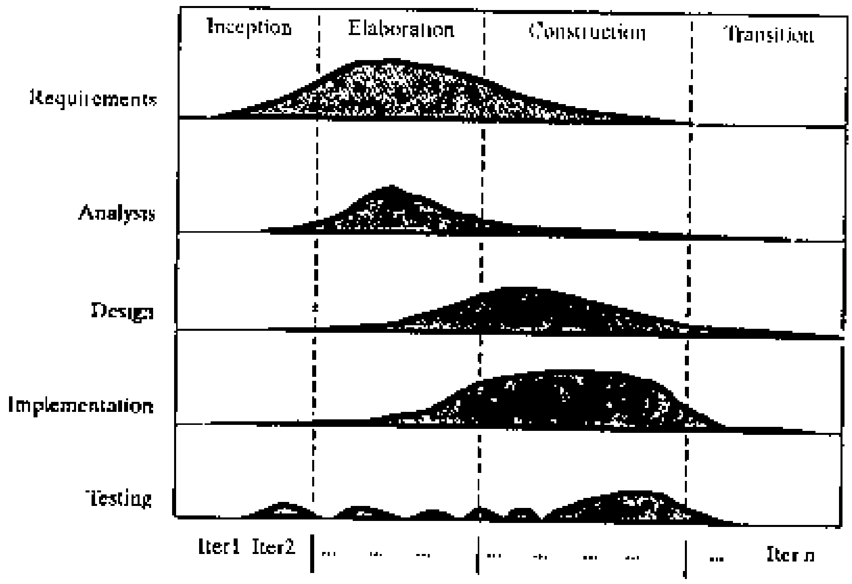

Figure 2.1 Activity (workflows), with phases and iteration.

2.3.1 The inception phase

The purpose of the inception phase is to determine whether it is worthwhile to develop the target information system. In other words, the primary aim of this phase is to determine whether the proposed information system is economically viable [6]. The inception phase defines the scope of the project by specifying use cases (which are similar to the user stories) as with any OO development approach. The project team also completes a feasibility study to determine whether resources should be invested in the project [3] .

2.3.2 The elaboration phase

The aim of the elaboration phase is to refine the initial requirements (use cases). refine the architecture, monitor the risks and refine their priorities, refine the business case, and produce the project management plan. The reason for the name "elaboration phase" is obvious; the major activities of this phase are refinements or elaborations of the previous phase [6]. The elaboration phase focuses on two activities: (1) defining the requirements and (2) creating a basic plan for the system architecture [3]. The team defines requirements by creating use case diagrams, class diagrams, sequence diagrams, and other UML diagrams. Final cost and benefit estimates are also completed by the end of the elaboration phase

2.3.3 The construction phase

The purpose of the construction phase is to produce the first operational quality version of the information system. sometimes called the beta release [6]. During the construction phase, you build the software using several iterations, possibly creating multiple releases of the system.

RUP also defines workflows within each phase. They include business modeling. requirements modeling, analysis and design, implementation, testing, deployment. configuration and change management, and project management. Each iteration involves activities from all workflows. RUP also defines many roles played by developers and many models created during the project. Typical roles include designer, use case specifier, systems analyst, implementer, and architect [3].

2.3.4 The transition phase

The aim of the transition phase is to ensure that the client's requirements have indeed been met. This phase is driven by feedback from the sites at which the beta release has been installed. (In the case of a custom information system developed for a specific client, there will be just one such site.) Faults in the information system are corrected. Also, all manuals are completed. During this phase, it is important to try to discover any previously unidentified risks [6].

During the transition phase, we turn the system over to the end users and focus on end-user training, installation, and support kinds of structural things [3].

2.4 Project management

Software development is complex, expensive, time consuming, and usually done by groups of people with varying skills and abilities. If it is simply allowed to "happen" the result is chaos. Chaos is avoided when software development is managed. Software development must be planned, organized, and controlled, and the people involved must be led [9].

Process management is an ongoing activity that documents, manages the use of, and improves an organization's chosen methodology (the "process") for systems development. Process management is concerned with the activities, deliverables, and quality standards to be applied to all projects [13].

2.4.1 Project planning

The first step in planning a project is to define and document the assumptions, goals, and constraints of the project. A project needs a clear statement of goals to guide the engineers in their daily decision making. Many engineers on typical projects spend many hours discussing alternatives that are known clearly to the project manager, but that have not been documented or disseminated properly [1]. Estimation is calculation of the approximate cost, effort, time, or resources required to achieve some end. Thus project planning begins with estimation of the effort needed to do the work, how long it will take, and how many people and other resources (such as computers, software tools, and reference materials) will be required to do it [9]. Once estimates are available, a schedule can be devised. the project manager is responsible for scheduling all project activities. The project schedule should be developed with an understanding of the required tasks, task duration, and task prerequisites [13].

2.4.2 Project organization

There are many ways to organize people into groups and assign them responsibilities and authority, called organizational structures. There are also many ways for people in groups to interact, make decisions, and work together, called team structures. For example, groups might be responsible for carrying projects from their inception through completion, which is called a project organization, or they might be responsible for just part of the project, such as design or coding or testing, which is called a functional organization [9]. An organizational structure is necessary at any level of an enterprise, whether it is to coordinate the efforts of a group of vice presidents who report to the president of the corporation or to orchestrate the interactions among programmers who report to a common project manager [1].

2.4.3 Project staffing

Project staffing is the activity of filling the roles designated in an organizational structure and keeping them filled with appropriate individuals. This activity includes hiring and orienting new employees, supporting people with career development guidance and opportunities through training and education, evaluating their performance, and rewarding them with salaries and various kinds of awards and recognitions.

It is often noted that the single most important ingredient in a successful software project is the people doing the project. Thus, project staffing is an essential part of project management [9].

2.4.4 Project control

Perhaps the manager's most difficult and important function is controlling the project. Few plans will be executed without problems and delays. The project manager must monitor and report progress against goals, schedule, and costs and make appropriate adjustments when necessary [13].

2.5 Risk management

A risk is a condition or event that can cause a project to exceed its shortest possible schedule. Examples include changing user requirements; failure of hardware, support software, or tools; and loss of needed development resources such as funding or personnel. Risks also arise from dependency on others, including clients, suppliers, and other organizational units. Risks are present regardless of what approach is taken to system development [3].

Risk management is a systematic process of identifying and mitigating software development risks. Principles of risk management are based on the following ideas:

- Most risks can be identified if specific attention is directed to them.
- Risks appear, disappear, increase, and decrease as the development process proceeds.
- Small risks should be monitored, whereas large risks should be actively mitigated [13].

Figure 2.2 depicts a flowchart for risk management. The process begins at the start of the development project and continues until it is completed. The first step is to identify all risks that are likely to affect the project. This task is relatively difficult but critically important since risks that aren't identified can't be managed. A group of project participants (including users) usually identifies risks, since more heads are better than few to generate and evaluate a large number of ideas [9].

For each risk the next step is to estimate its probability, determine possible outcomes, and estimate the probability and schedule delay associated with each outcome For example, the risk of failing a performance test might be assigned a probability of 0.1 (10 percent) and determined to have two possible outcomes:

- Reprogram key software modules to improve performance.
- Purchase and install faster hardware [9].

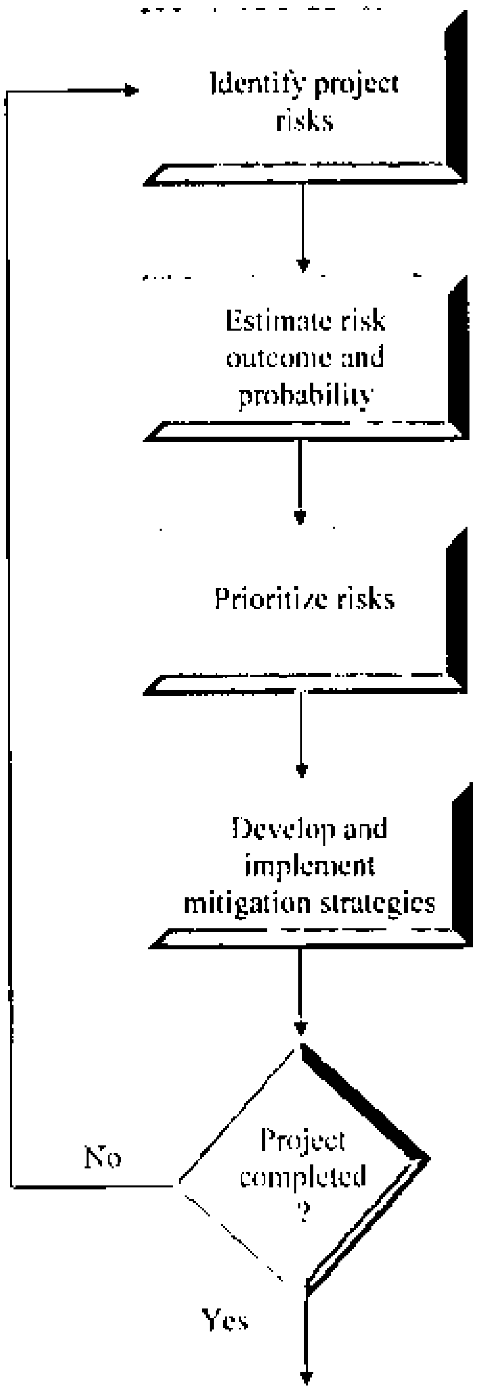
Identify project risks
Estimate risk outcome and probability
Prioritize risks
Develop and implement mitigation strategies
Project completed ?
No
Yes

Figure 2.2 steps in risk management.

2.6 Basic object-oriented concepts

The object-oriented approach to system development is based on the concept that objects exist within a system's environment. Objects are everywhere [13].

2.6.1 Objects

How do some of the authorities in the field define an object? Coad and Yourdon borrow their definition from the dictionary:

A tangible and/or visible thing; something that may be apprehended intellectually; something toward which thought or action is directed. An individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain. Anything with a crisply defined boundary[7].

A person or thing through which action, thought, or feeling is directed. Anything visible or tangible; a material product or substance [7].

An object is a thing in the computer system that is capable of responding to messages, this radically different view of a computer system requires a different approach to systems analysis, systems design, and programming [3].

Attributes are the data that represents characteristics of interest about an object, an instance (or object instance) of an object consists of the values for the attributes that describe a specific person, place, thing, or event [13].

2.6.2 Classes

Class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class implements one or more interfaces. Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations [4] . A class is therefore a type of thing, and all specific things that fit the general definition of the class are things that belong to the class. [7].

Each object in a class should be identifiable in some way so we can distinguish between each object. Usually one attribute, or piece of information about an object, is used to identify an object in the user's work context. When modeling data, an identifier is also required. Sometimes the object naturally has an identifier, such as the Social Security Number for a person or the name for a state. Other times the identifier must be created by the system to allow an object to be uniquely identified [7].

2.6.3 Encapsulation

Encapsulation is the packaging of several items together into one unit (also referred to as information hiding) [13]. With objects, both the attributes and the methods of the class are packaged together. So, the object knows things (attributes) and knows how to do things (methods). This is what makes the object much more than a data entity. A data entity just has attributes. Encapsulation allows us to think of the attributes and behaviors of the object as one package. It can help to think of encapsulation as the combining of attributes and methods [7].

2.6.4 Messages

We send messages to objects, and objects send messages to us (and to each other). Information hiding prevents the end user from changing an object's data; however, the end user can ask the object to invoke, or perform, a method, and the method might change the object's data. Therefore, when we ask an object to do something, we are sending a message to the object, asking it to invoke a method [7]

2.6.5 Inheritance

Inheritance means that methods and/or attributes defined in an object class can be inherited or reused by another object class. The approach that seeks to discover and exploit the commonalities between objects/classes is referred to as generalization / specialization [19] .

By inheritance, it is possible to add new features to an existing class without modifying the previous class, so this is the way to derive a new class from an existing one. The new class is called a subclass or a derived class. Class inheritance combines interface inheritance and implementation inheritance. Interface inheritance defines a new interface in terms of one or more existing interfaces. Implementation inheritance defines a new implementation in terms of one or more existing implementations [13].

2.6.6 Polymorphism

Polymorphism means the ability to take more than one form. Through polymorphism, it is possible to hide many implementations behind the same interface. Polymorphism is a concept in type theory, according to which a name may denote objects of many different classes that are related by some common super class; thus, any object denoted by this name is able to respond to some common set of operations in different ways [18].

Polymorphism plays an important role in allowing objects to have different internal structures but share the same external interface. This means that a general class of operations can be accessed in the same manner, even though specific actions associated with each operation may differ [20].

العنوان:	Object oriented sottwore engineering approach to industriol intormotion system design and implementation
المؤلف الرئيسي:	مانيطة، عيسى حسين أحمد
مؤلفين آخرين:	الفقيه، إدريس ساسـ(مشرف)
التاريخ الميلادي:	2008
موقع:	سرت
الصفحات:	1 - 78
رقم MD:	859171
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة التحدي
الكلية:	كلية العلوم
الدولة:	ليبيا
قواعد المعلومات:	Dissertations
مواضيع:	نظم المعلومات، هندسة البرمجيات، البرمجيات الشيئية، نظم المعلومات الصناعية
رابط:	https://search.mandumah.com/Record/859171

Chapter 3

Requirement analysis and specification

3.1 Overview

A requirement is a feature that the system must have or a constraint that it must satisfy to be accepted by the client. Requirements engineering aims at defining the requirements of the system under construction. Requirements engineering includes two main activities: requirements elicitation, which results in the specification of the system that the client understands, and analysis, which results in an analysis model that the developers can unambiguously interpret [4]. System requirements are all of the capabilities and constraints that the new system must meet [3]. The process of discovering the client's requirements is termed requirements elicitation (or requirements capture). Once the initial set of requirements has been drawn up, the process of refining and extending them is termed requirements analysis [8]. There is consensus, among both software developers and customers, that the activities of eliciting, understanding, and specifying requirements are the most critical aspects of the software engineering process. The result of the requirements activities is a requirements specification document, which describes what the analysis has produced [1].

3.2 Requirements elicitation concepts

The analysis phase involves gathering a considerable amount of information. Systems analysts obtain some information from people who will be using the system, either by interviewing them or by watching them work. They obtain other information by reviewing planning documents and policy statements. Documentation from the existing system should also be studied carefully. Analysts can obtain additional information by looking at what other companies (particularly vendors) have done when faced with a similar business need. In short, analysts need to talk to nearly everyone who will use the new system or has used similar systems, and they must read nearly everything available about the existing system [3]. Requirements elicitation focuses on describing the purpose of the system. The client, the developers, and the users identify a problem area and define a system that addresses the problem. Such a definition is called a system specification and serves as a contract between the client and the developers [4]. The specification document is specifies precisely what the product must do and the constraints on the product. Virtually every specification document incorporates constraints that the product has to satisfy [8]. The system specification is structured and formalized during analysis to produce an analysis model (see Figure 3.1) [4].

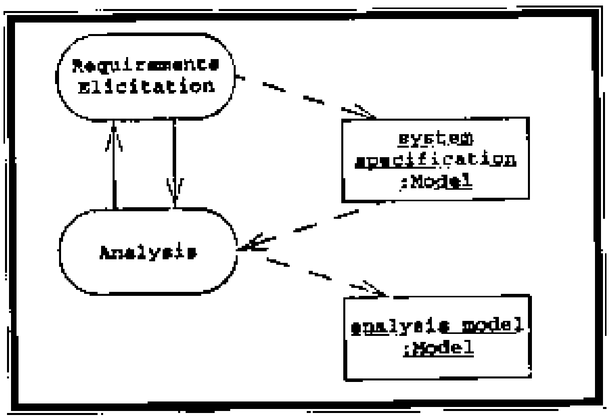

Figure 3.1 Products of requitaion and elicitation and analysis (UML activity diagram).

3.3 Elicitation techniques

There are many requirements elicitation techniques, some quite sophisticated. A technique's effectiveness depends largely on the stakeholders whose needs are sought [9]. The following techniques are the mainstays of needs elicitation.

3.3.1 Interview

An interview is a question and answer session during which one or more analysts ask questions to one or more stakeholders or problem-domain experts. Interviews are the primary means of obtaining verbal input from committed stakeholders, such as development organization stakeholders and customers for custom and niche-market products[9].

Interviewing stakeholders is by far the most effective way to understand business functions and business rules. It is also the most time-consuming and resource-expensive. In this method, systems analysts meet with individuals or groups of users. A list of detailed questions is prepared and discussion continues until all the processing requirements are understood and documented by the project team [3]. There are two basic types of question. A closed-ended question requires a specific answer. For example, the client might be asked how many salespeople the company employs or how fast a response time is required. Open-ended questions are asked to encourage the person being interviewed to speak out [8].

3.3.2 Observation

We can observe the work in several ways, from a quick walkthrough of the office or plant to doing the work yourself. A quick walkthrough gives a general understanding of the layout of the office, the need for and use of computer equipment, and the general work flow. Spending several hours to observe a user actually doing his or her job provides an understanding of the details of actually using the computer system and carrying out business functions. By being trained as a user and actually doing the job, you can discover the difficulties of learning new procedures, the importance of a system that is easy to use, and the stumbling blocks and bottlenecks of existing procedures and information sources [3]. Observation is especially useful for eliciting derivative product and maintenance release needs because it can reveal many opportunities for product improvement [9].

3.3.3 Focus group

A focus group is an informal discussion among six to nine people led by a facilitator who keeps the group on topic. Requirements elicitation focus groups consist of stakeholders or stakeholder representatives who discuss some aspect of the product. Focus groups are especially useful for eliciting user-level needs and desires. Focus groups are the main technique of obtaining needs for consumer products, especially new products and those with visionary or leading-edge technologies. People may be paid to participate in focus groups to compensate them for their time [3].

3.3.4 Elicitation workshop

An elicitation workshop is a facilitated and directed discussion aimed at describing the product design problem or establishing stakeholder needs and desires. Elicitation workshops are similar to focus groups but are more tightly controlled, have more precisely defined goals, and require more time and effort from participants. Workshops may be held to describe business processes, list or describe use cases work out data formats, or specify user interface needs. Elicitation workshops are a powerful technique useful for learning about the design problem and determining needs at all levels of abstraction, but they are appropriate only for committed s stakeholders [3].

3.3.5 Prototype demonstrations

A prototype is an initial, working model of a larger, more complex entity. Prototypes are used for many different purposes, and there are many names to differentiate these uses: throw way prototypes, discovery prototypes, design prototypes, and evolving prototypes. Each can be used during different phases of the project to test and validate ideas. Prototypes are used to test feasibility and to help identify processing requirements [3]. Prototypes provide a useful basis for conversations with stakeholders about features, capabilities, and user interface issues such as interaction protocols [9].

3.4 Types of requirements

Requirements are often categorized and used on different levels in the requirement process. This is done since often a large number of requirements are elicited and grouping them makes them easier to handle. The two most common categories are functional requirements and non-functional requirements [9].

3.4.1 Functional requirements

A functional requirement is a function or feature that must be included in an information system to satisfy the business need and be acceptable to the users [13].

Functional requirements are the typical user requirements for a system, in that they define the desired functionality of a system. User requirements should have some observable effect on a particular user of a system, otherwise they are possibly being defined at too low level [15].

3.4.2 Non-functional requirements

the Non-functional requirements, which are also sometimes referred to as "Implementation requirements" or "constraints". These non-functional requirements constrain functional requirements, which means that they may limit functional requirements in some way [15].

A nonfunctional requirement is a description of the features, characteristics, and attributes of the system as well as any constraints that may limit the boundaries of the proposed solution [13].

3.5 Requirements elicitation activities

In this section, we describe the requirements elicitation activities. These maps a problem statement into a system specification that we represent as a set of actors, scenarios, and use cases.

3.5.1 Identifying Actors

Actors represent external entities that interact with the system. An actor can be human or an external system. An actor represents anything that needs to interact with the system to exchange information. An actor is a user or a role, which could be an external system as well as a person[13].

When identifying actors, developers we may ask the following questions:

- Which user groups are supported by the system to perform their work?
- Which user groups execute the system's main functions?
- Which user groups perform secondary functions, such as maintenance and administration?
- Will the system interact with any external hardware or software?[4]

3.5.2 Identifying Use case

A use case is a behaviorally related sequence of steps (a scenario), both automated and manual, for the purpose of completing a single business task.

Use cases describe the system functions from the perspective of external users and in the manner and terminology they understand. To accurately and thoroughly accomplish this demands a high level of user involved (see figure 3.2 [13].

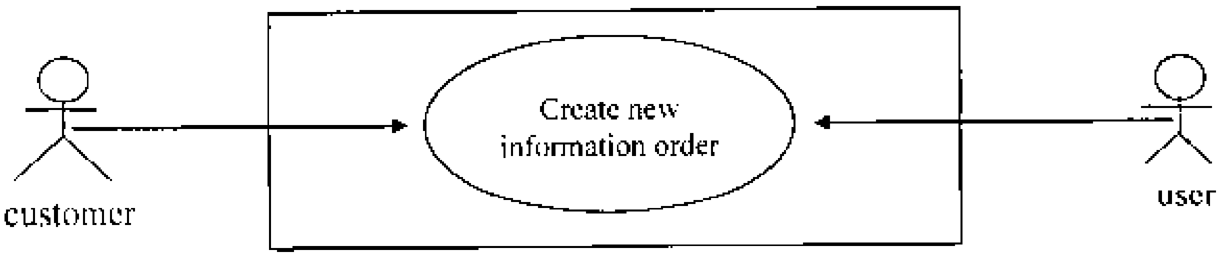

Figure 3.2 A simple use case with two actor

3.5.3 The use case diagram

A use case diagram is a graphical model that summarizes the information about the actors and use cases. The purpose of the use case diagram is to identify the "uses", or use cases, of the new system. To do use case analysis, we look at the system as a whole and trying to identify all of its major uses. Another way to think about a use case diagram is that it is a functional description of the entire system. It identifies the functions that the new system must support (figure 3.3 illustrate a use case diagram) [3].

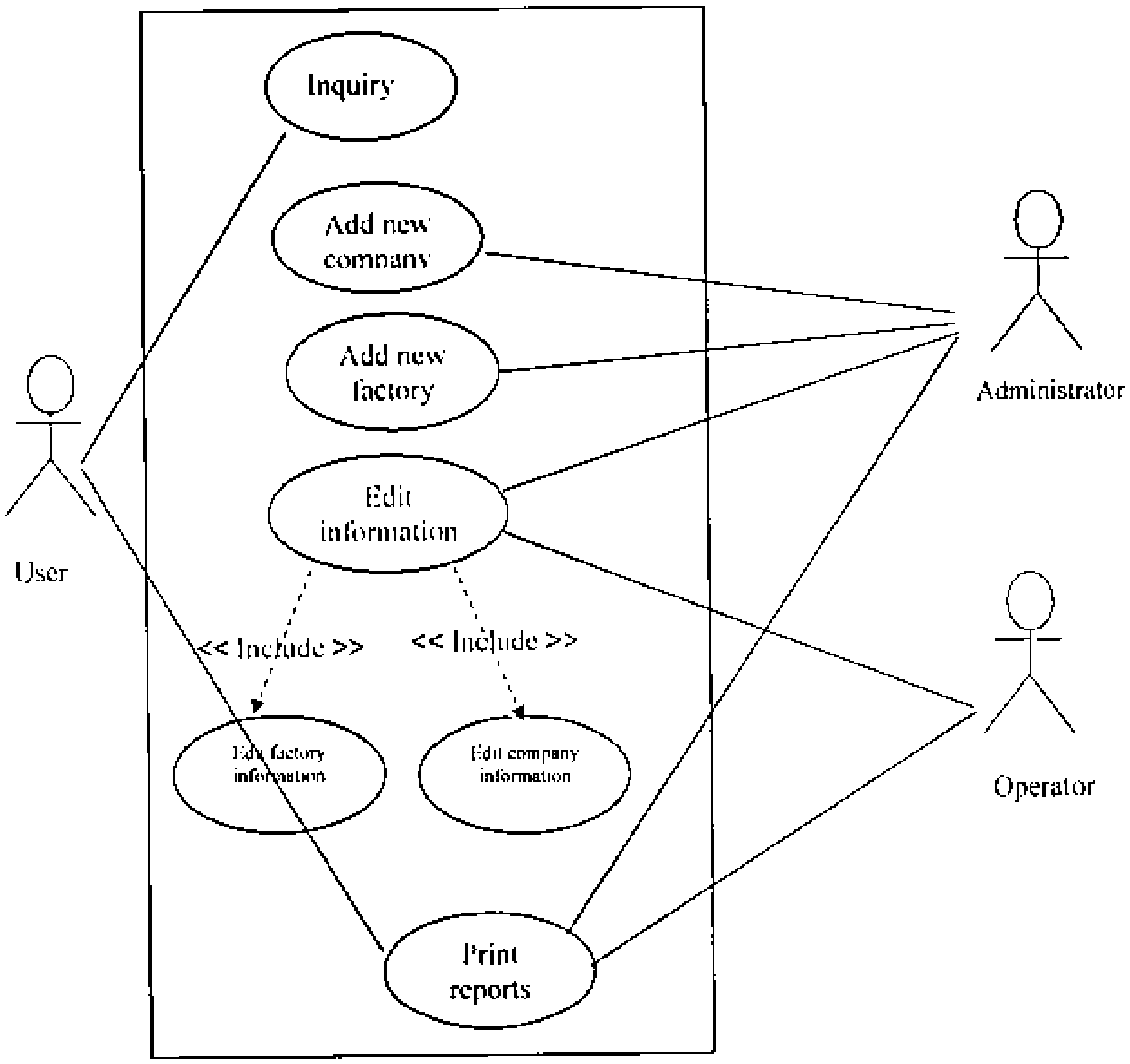

Figure 3.3 Example showing use cases diagram of the industrial information system

3.6 Analysis activities from use cases to objects

In this section, we describe the activities that transform the use cases and scenarios produced during requirements elicitation into an analysis model [4].

3.6.1 Identifying entity objects

Entity objects usually correspond to items in real life and contain information, known as attributes, that describes the different instances of the entity. They also encapsulate those behaviors that maintain its information or attributes. An entity object is typically stored in a database [13].

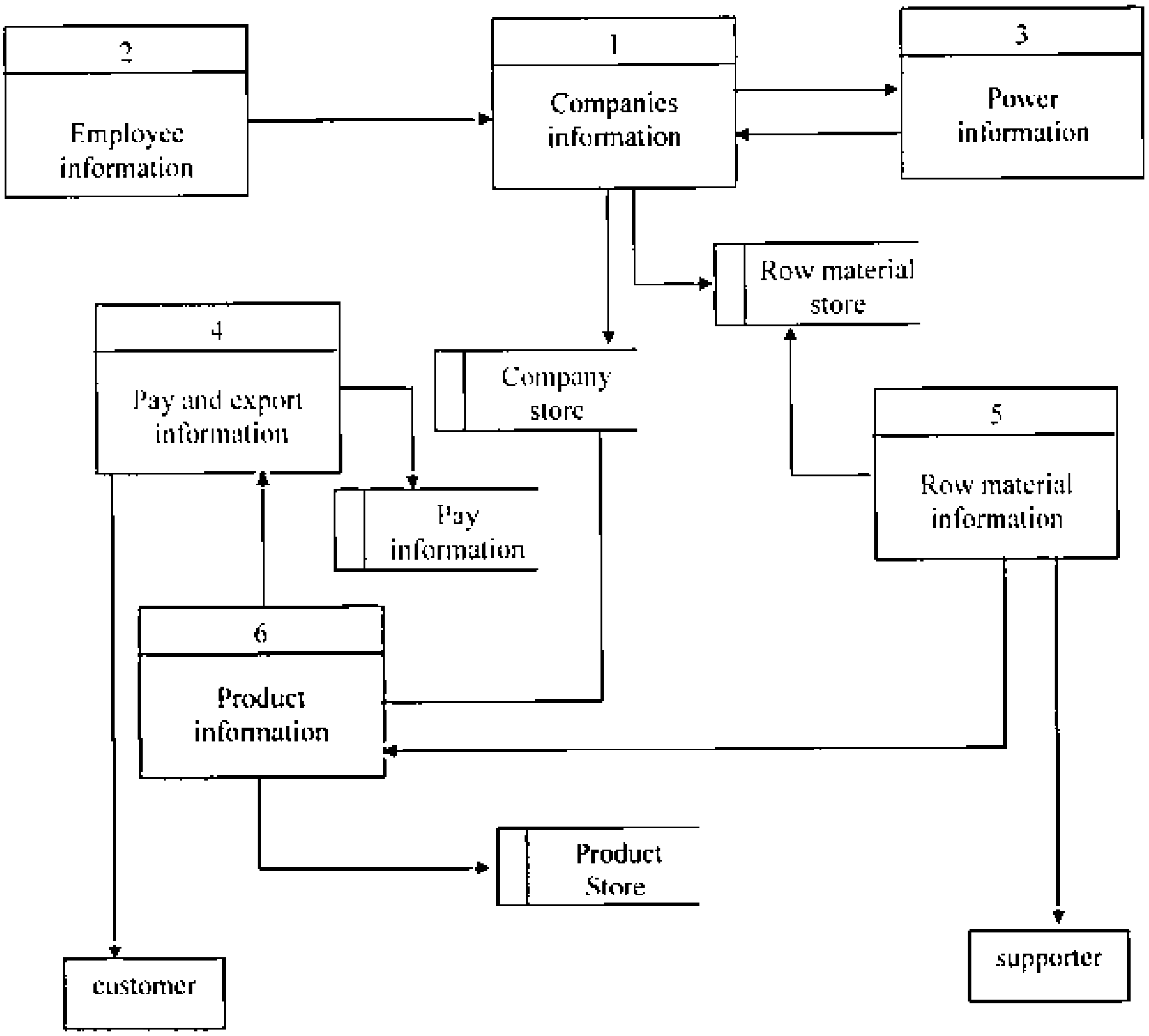

Figure 3.4 A simple dataflow diagram of the industrial information system

3.6.2 Object interaction

A characteristic of object-oriented development, which will be more evident as we learn about interaction diagrams, for example, as we try to specify the user requirements, we develop some of the system details that later become design elements. In other words, analysis and design models are much more close aligned than in the traditional approach [3].

Two types of interaction diagrams can be used to describe the flow of information and interactions between objects. A collaboration diagram and a sequence diagram contain the same information, but each has a slightly different focus. A collaboration diagram emphasizes the objects that interact to support a use case, while a sequence diagram focuses more on the details of the messages themselves. During system development, you may use both types or either type of diagram. Analysts who prefer a top down approach tend to draw a collaboration diagram first to get an overview of the objects that collaborate to carry out the use case [13].

3.6.2.1 Sequence diagram

A sequence diagram shows the sequence of the interactions between objects that occurs during the flow of events of a single scenario or use case. A sequence diagram includes four basic symbols: (1) the actor symbol, represented by a stick person, (2) the object symbol, represented by a rectangle with the name underlined, (3) the lifeline symbol, represented by a dashed line, and (4) the message symbol, represented by a directional arrow with a message descriptor. Figure 3.5 illustrates A sequence diagram of the industrial information system.

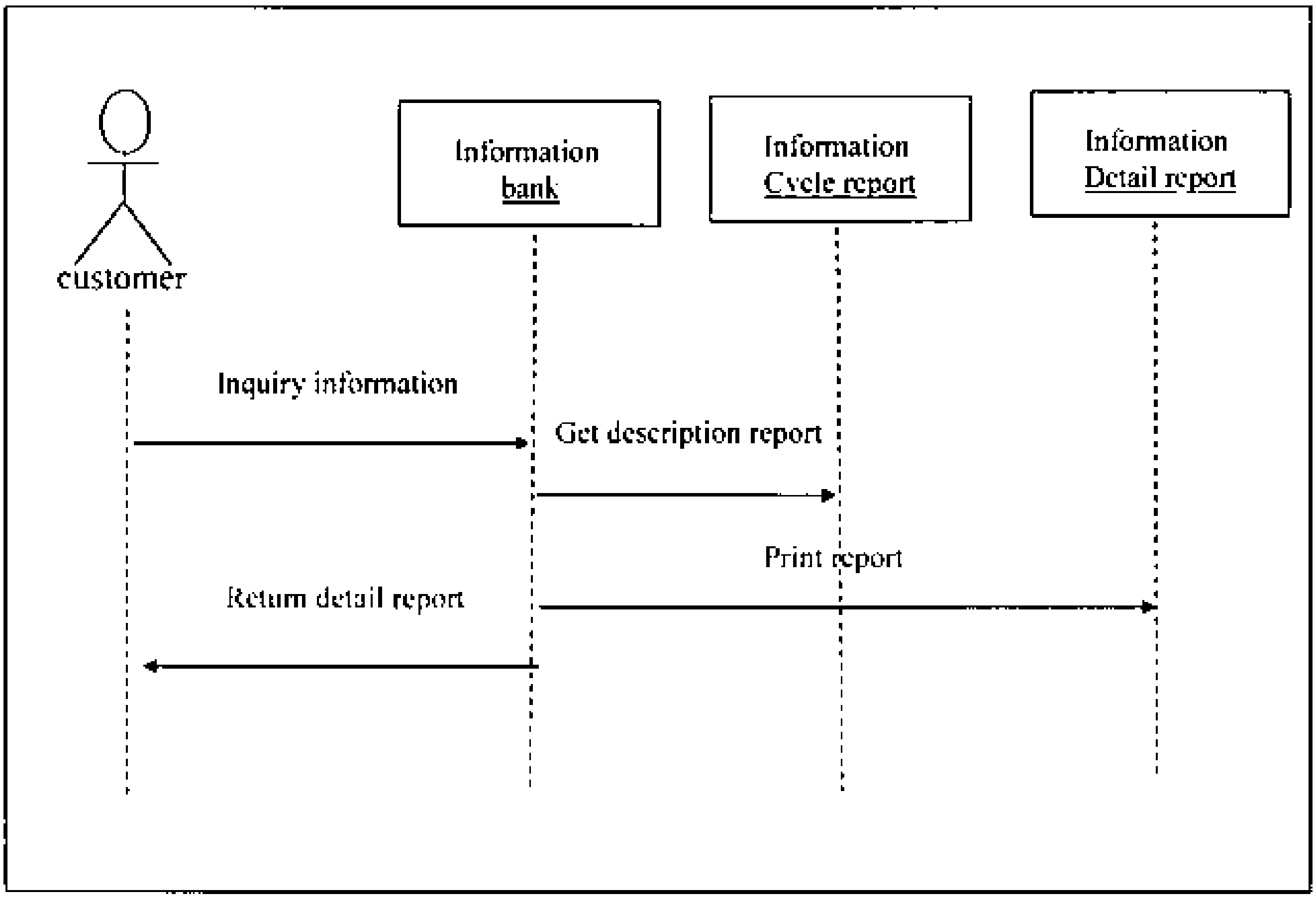

Figure 3.5 A sequence diagram of the industrial information system.

3.6.2.2 Collaboration diagram

The primary use of a collaboration diagram is to get a quick overview of all the objects that collaborate to support a given scenario. A collaboration diagram uses the same symbols for actors, objects, and messages found in a sequence diagram. The lifeline symbol is not used. However, a different symbol, the link symbol, is used. Figure 3.6 illustrates collaboration diagram of the industrial information system.

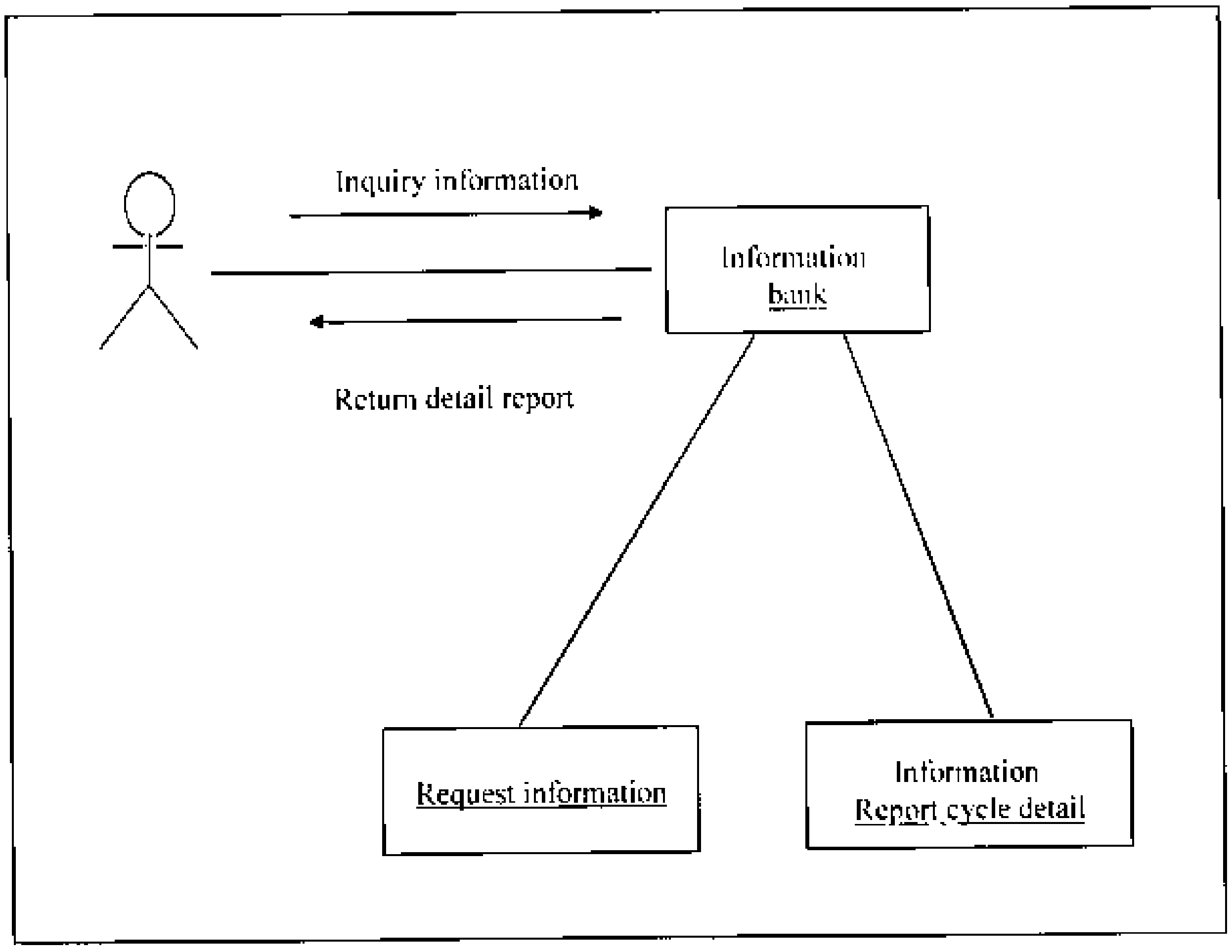

Figure 3.6 Collaborate diagram of the industrial information system.

Chapter 4

System design and implementation

4.1 Use case Design

Industrial Information System designed for Industrial Information and Documentation Center (IIDC) for seeking information about industrial companies and their factories, number of employee, quantity and quality of production, and other attributes. The system will be made flexible enough so that it can benefit the normal user and the decision makers. Decision maker will be capable to view problems and constraints that faces industrial activity through their view of information provided by the system. They will also be able to make decision toward correcting industrial sector from a subjective overview of the whole sector.

The system shall be accessible to the different privileged levels as follows:

- Public users.
- Operator.
- Company and staff.
- Administrator.

All users, need legal user name and password to access the system. The system shall be accessible for multiple users simultaneously. The system shall operate on the Industrial Information and Documentation Center (IIDC) server and database.

4.1.1 Functions for public users

- Access to the System, information about company ,inquiry, generate reports

4.1.2 Functions for Operator

- Manage the information of the company so as to edit the basic information such as email, sector, activity ,employee, and location
- Manage the product information .
- Manage row material information
- Generate Statistics about employee, products, exports, and etc.
- Generate reports on the active window . then can he printing it

4.1.3 Functions for company and staff

- Manage the information of the company so as to edit the basic information such as email, sector, activity ,employee and location
- Manage the product information .
- Manage row material information
- Generate some reports, such as statistics reports about companies ,factories, and sector,

4.1.4 Functions for Administrator

- Manage the information of the company so as to edit the basic information such as email, sector, activity ,employee and location.
- Add and delete new company
- Manage the product information .
- Manage row material information
- Generate some reports, such as statistics reports about companies ,factories, and sector.

4.2 Use-Case View

The system functions can be described by using use-case diagrams graphically. Different users have different views for the system. All the system functions are divided into following modules:

- use-case view of the public user (see Figure 4.1)
- use-case view of operator (see Figure 4.2)
- use-case view of the company and staff (see Figure 4.3)
- use-case view of administrators (see Figure 4.4)

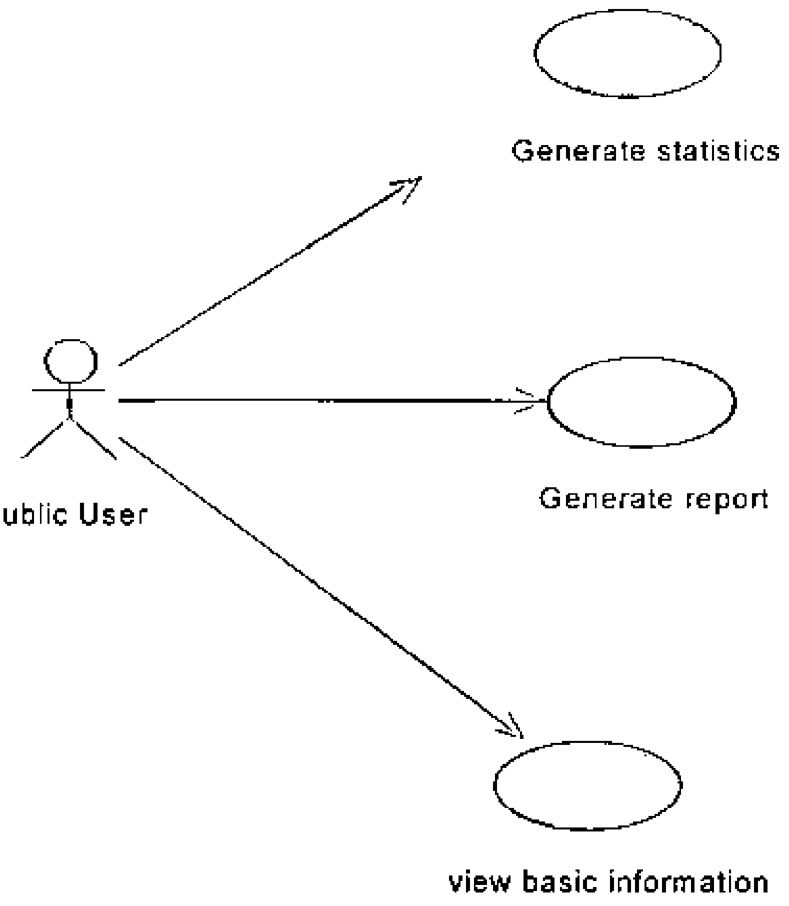

Figure 4.1 : Use case view of public user

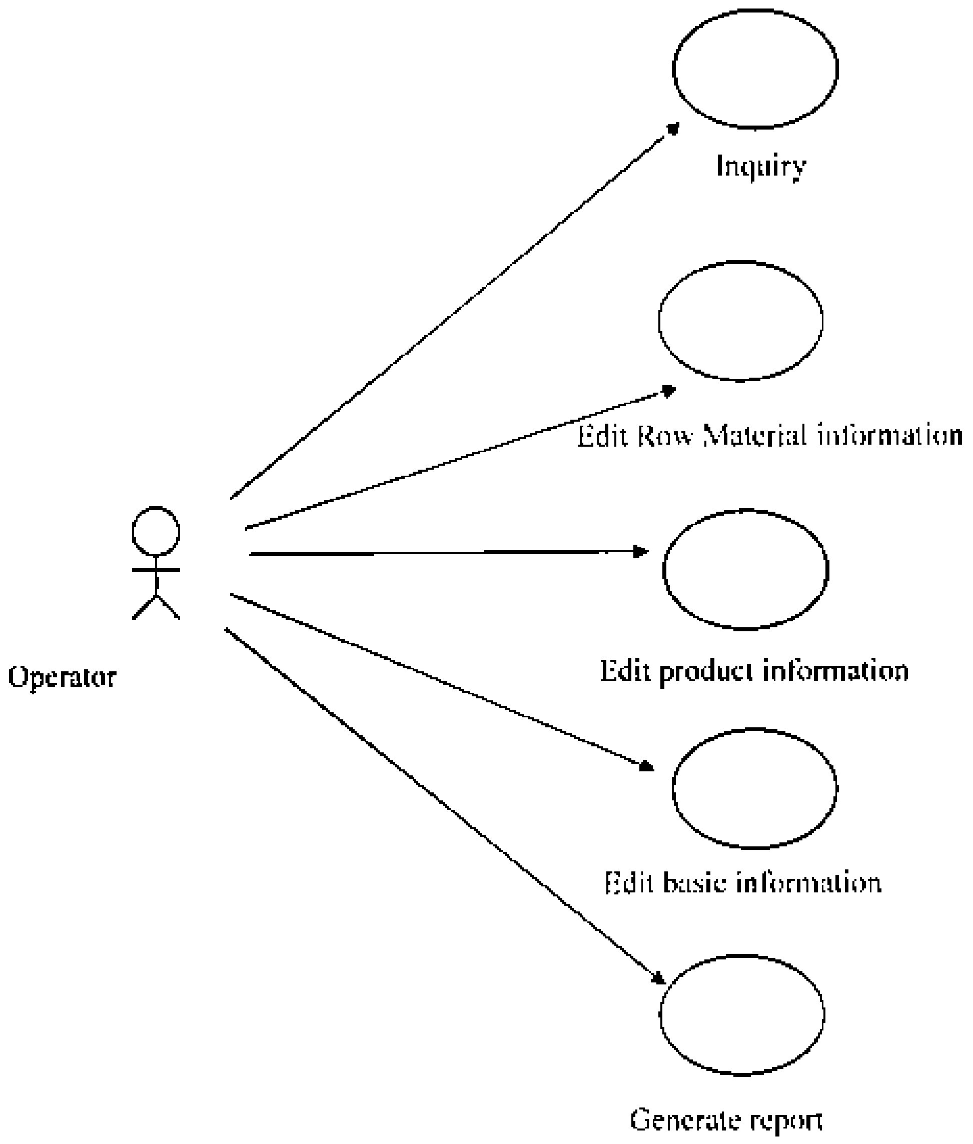

Figure 4.2. Use-case view of operator

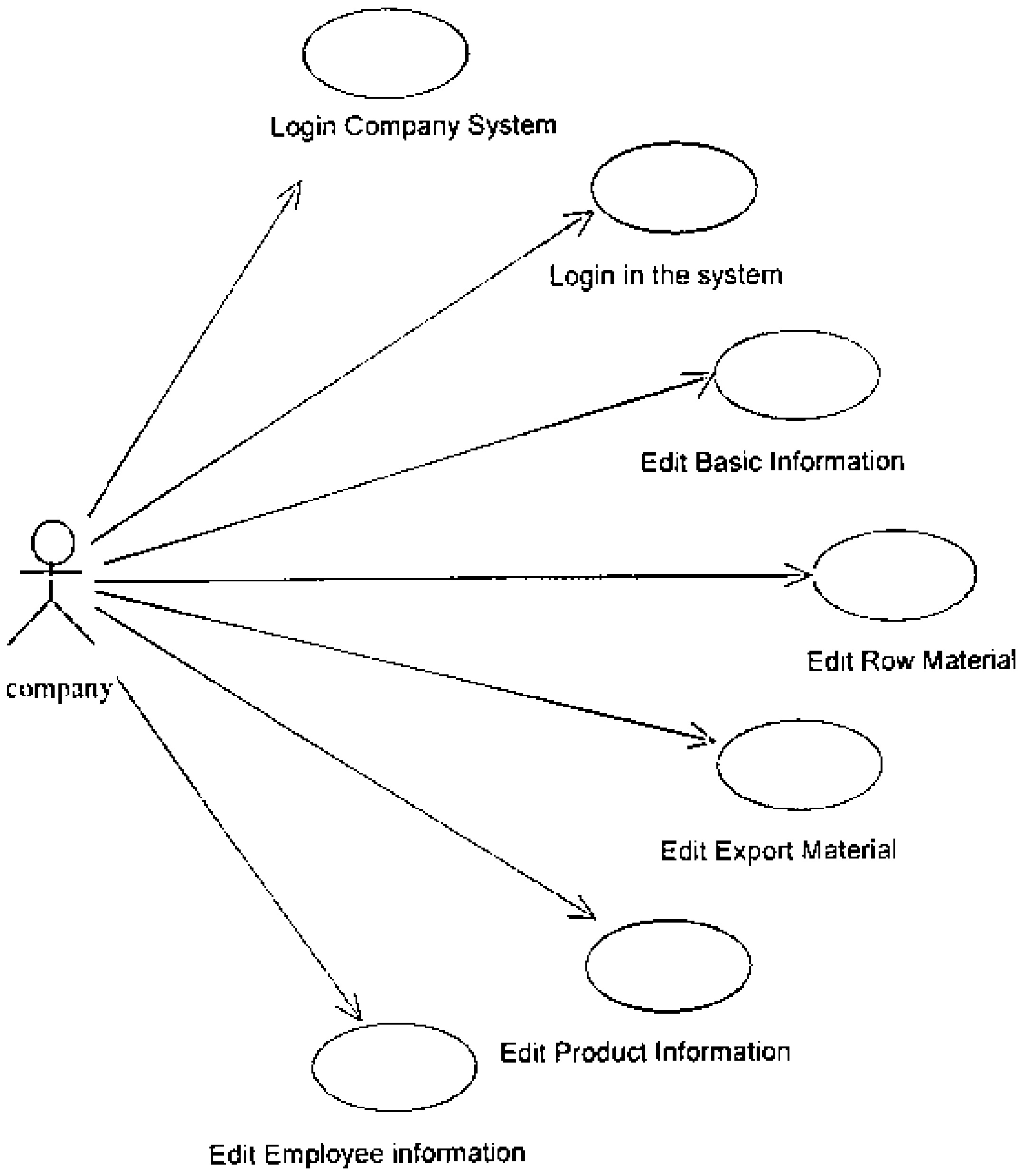

Figure 4.3. use-case view of the company and staff

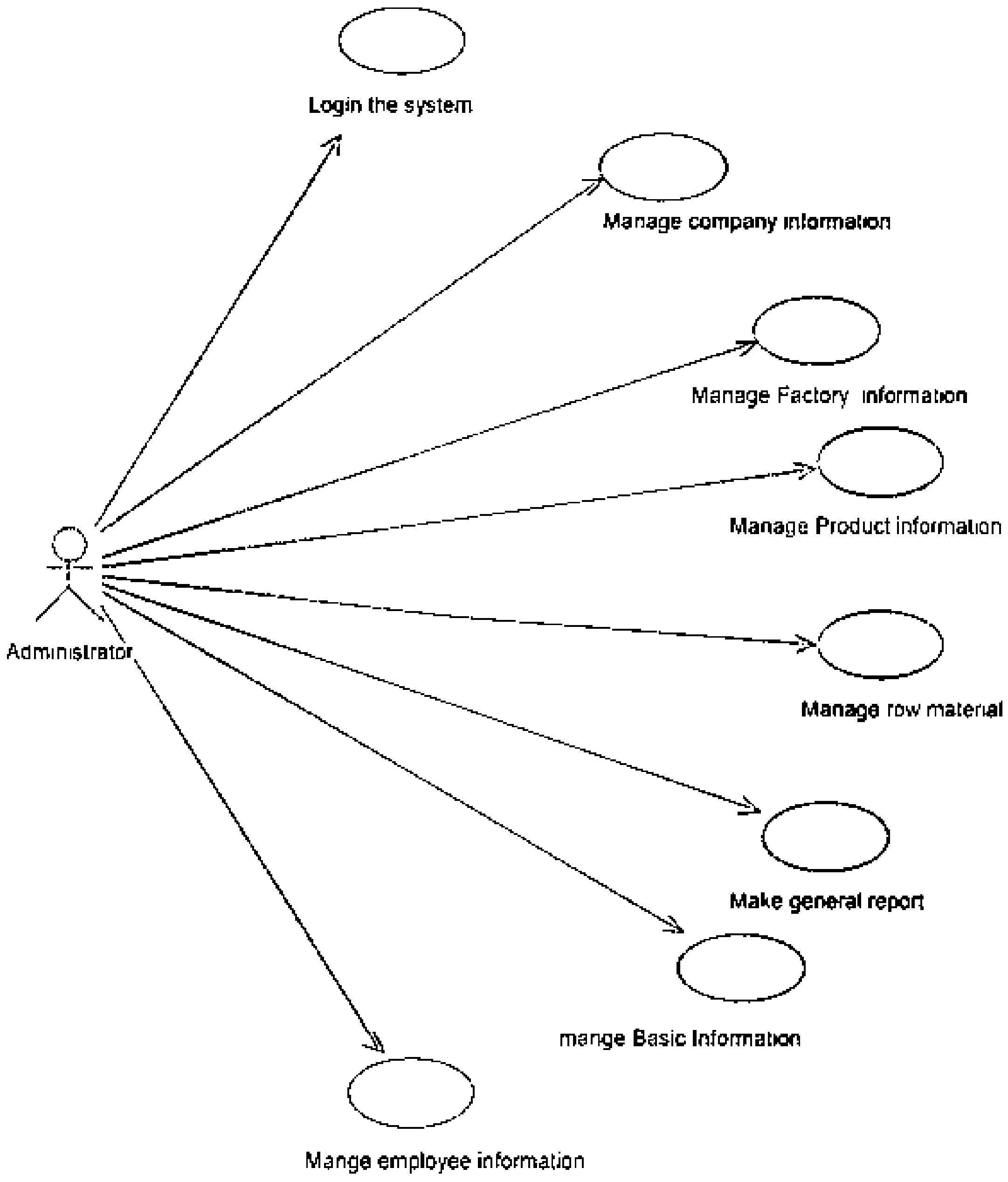
Login the system
Manage company information
Manage Factory information
Manage Product information
Administrator
Manage row material
Make general report
mange Basic Information
Mange employee information

Figure 4.4 use-case view of administrators

4.3 Database design

A database (DB) is an integrated collection of stored data that is centrally managed and controlled. A database typically stores information about dozens or hundreds of entity types or classes. The information stored includes entity or class attributes as well as relationships among the entities or classes. A database also stores descriptive information about data such as field names, restrictions on allowed values and access controls to sensitive data items [3].

A database is managed and controlled by a database management system (DBMS). A DBMS is a system software component that is generally purchased and installed separately from other system software components (for example, operating systems) Examples of modem database management systems include Microsoft Access, Oracle DB/2, ObjectStore, and Gemstone [13].

4.3.1 Database and database management system

Figure 4.5 illustrates the components of a typical database and its interaction with a DBMS, application programs, users, and administrators[13]. The database consists of two related information stores: the physical data store and the schema. The data store contains the raw bits and bytes of data that are created and used by the information system [14]. The schema contains descriptive information about the data stored in the physical data store, including the following:

- Access and content controls, including allowable values for specific data elements value dependencies among multiple data elements, and lists of users allowed to read or update data element contents.
- Relationships among data elements and groups of data elements (for example, a pointer from data describing a customer to orders made by that customer).
- Details of physical data store organization, including type and length of data elements, the locations of data elements, indexing of key data elements, and sorting of related groups of data elements [3].

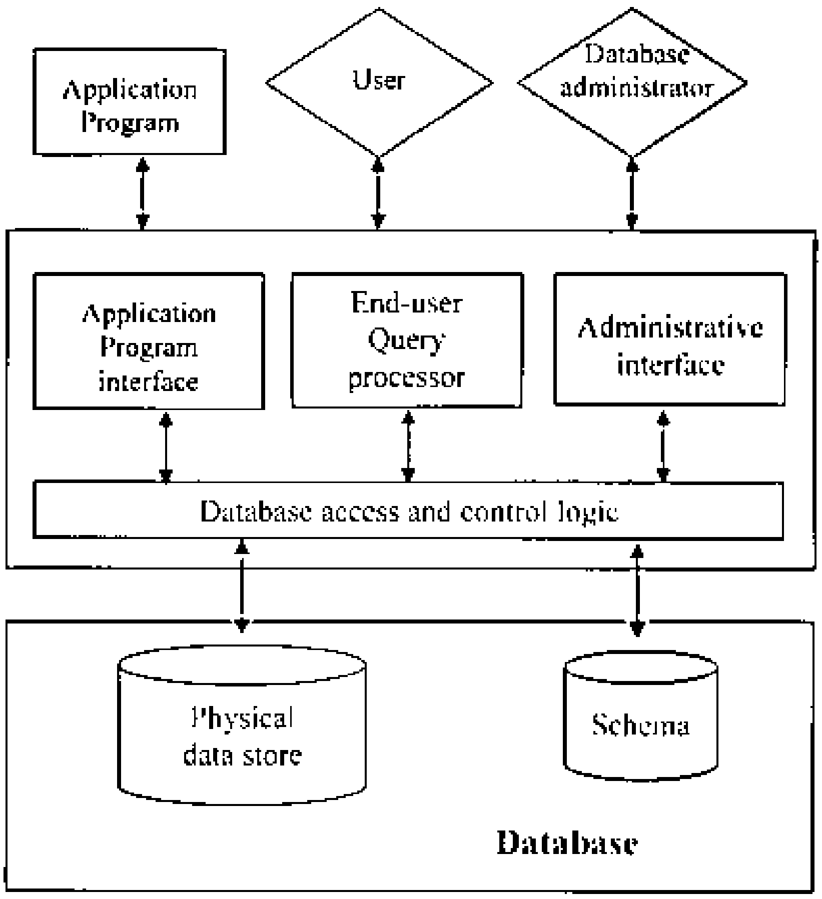

Figure 4.5 the components of a database and database management system

DBMSs have evolved through a number of technology stages since their introduction in the 1960s. The most significant change has been the type of model used to represent and access the content of the physical data store. Four such model types have been widely used:

- Hierarchical
- Network
- Relational
- Object-oriented.[3]

4.3.2 Object-oriented databases

Object database management systems (ODBMSs) are a direct extension of the OO design and programming paradigm. ODBMSs are designed specifically to store objects and to interface with object-oriented programming languages. Although it is possible to store objects in files or relational databases, there are many advantages to using a DBMS specifically designed for objects. These advantages include direct support for method storage, inheritance, nested objects, object linking, and programmer-defined data types [3].

ODBMSs are the database technology of choice for newly designed systems implemented with OO tools, especially for scientific and engineering applications. ODBMSs are expected to supplant RDBMSs in more traditional business applications gradually over the next decade as OO technology becomes more widely used and as better tools for interfacing object and relational databases are developed [14].

Because ODBMSs are relatively new, there are few widely accepted standards for specifying an object database schema. Some object database standards that are gaining wide acceptance are those proposed by the Object Database Management Group. One of these standards is the Object Definition language (ODL). ODL is a language for describing the structure and content of an object database [13].

4.3.2.1 Object model and class diagram

Objects can be classified into two broad types for purposes of data management. A transient object exists only during the lifetime of a program or process. Examples of transient objects include objects created to implement user-interface components (such as a view window or pop-up menu). Transient objects are created each time a program or process is executed and then destroyed when a program or process terminates [3].

A persistent object is not destroyed when the program or process that creates it ceases execution. Instead, the object continues to exist independently of any program or process. Storing the object state to persistent memory (such as a magnetic or optical disk) ensures that the object exists between process executions. Objects can be persistently stored within a file or database management system [3].

An object database schema includes a definition for each class that requires persistent storage. ODL class definitions derive from the corresponding UML class diagram. Thus, classes already defined in UML are simply reused for the database schema definition.

4.3.2.2 Representing relationships

Each object stored within an ODBMS is automatically assigned a unique object identifier. An object identifier may be a physical storage address or a reference that can be converted to a physical storage address at run time. In either case, each object has a unique identifier that can be stored within another object to represent a relationship.

The ODBMS uses attributes containing object identifiers to find objects that related to other objects. The process of extracting an object identifier from one object and using it to access another object is sometimes called navigation.

4.3.2.2.1 Include relationships

In UML modeling, an include relationship is a relationship in which one use case (the base use case) includes the functionality of another use case (the inclusion use case). The include relationship supports the reuse of functionality in a use case model. Include relationships usually do not have names. If you name an include relationship, the name is displayed beside the include connector in the diagram.

Include relationship is displayed in the diagram editor as a dashed line with an open arrow pointing from the base use case to the inclusion use case. The keyword «include» is attached to the connector, as in Figure 4.6 [2]

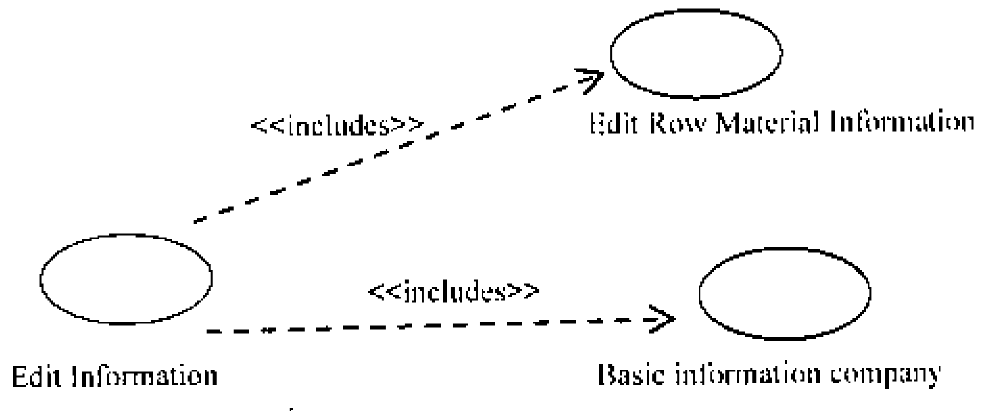

Figure 4.6 an example of an <<includes>> relationships (UML use case diagram)

4.3.2.2.2 Extend relationships

In UML modeling, we can use an extend relationship to specify that one use case (extension) extends the behavior of another use case (base). This type of relationship reveals details about a system or application that are typically hidden in a use case.

The extend relationship specifies that the incorporation of the extension use case is dependent on what happens when the base use case executes. The extension use case owns the extend relationship. You can specify several extend relationships for a single base use case.

4.3.2.2.3 Associations relationships

The association relationship is the most common relationship in a class diagram. The association shows the relationship between instances of classes. For example, the class company is associated with the class Factory. The multiplicity of the association denotes the number of objects that can participate in then relationship. For example, a Company object can be associated to one or more Factory , but a factory can be associated to zero or more, as in Figure 4.7

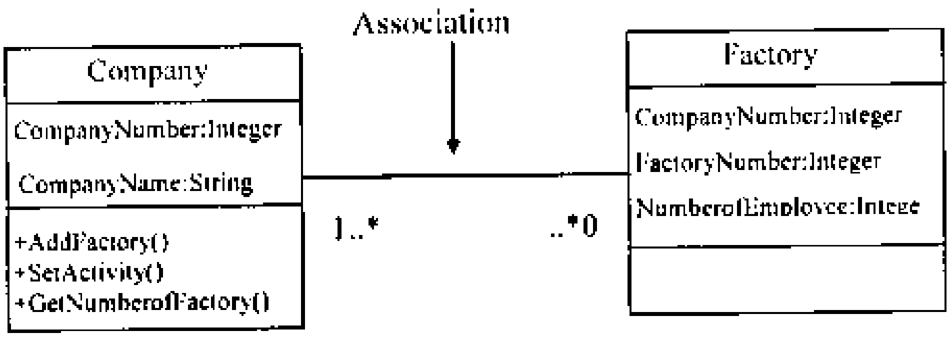

Figure 4.7 an example of a association relationship (UML use case diagram)

An association represents a structural relationship among objects. In many modeling situations, it's important for us to state how many objects may be connected across an instance of an association. This "how many" is called the multiplicity of an association's role, and is written as an expression that evaluates to a range of values or an explicit value as in table 4.1. When you state a multiplicity at one end of an association, you are specifying that, for each object of the class at the opposite end, there must be that many objects at the near end. we can show a multiplicity of exactly one (1), zero or one (0..1), many (0..*), or one or more (1..*). we can even state an exact number (for example, 3), as figure 4.8

Company can have many employees. An employee can only work for one company.

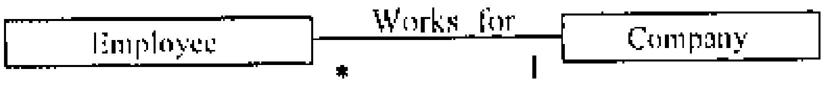

Figure 4.8 an example of multiplicity

Table 4.1 Some examples of specifying multiplicity

Multiplicities	Meaning
0..1	Optional (0 or 1)
0..*	Zero or more
1..1	Exactly one 1
1..*	at least one instance

One of the most important details we can specify for a classifier's attributes and operations its visibility. The visibility of a feature specifies whether it can be used by other classifiers. In the UML, we can specify any of three levels of visibility.

A classifier can see another classifier if it is in scope and if there is an explicit or implicit relationship to the target, show table 4.2

+ public: any class can use the feature (attribute or operation);
protected: any descendant of the class can use the feature;
- private: only the class itself can use the feature.

Table 4.2 visibility

4.3.2.2.4 Generalization relationships

Generalization/specialization relationship are mechanism for reducing the complexity of a model. A use case can specialize a more general one by adding more detail. Generalization relationships are used in class, component, deployment, and use case diagrams, as figure 4.9.

A generalization is used when two classes are similar, but have some differences.

In figure 4.10 the classes PublicOwnedCompany , PrivateCompany and PrivateandPublic have some similarities such as name and number, but each class has some of its own attributes and operations. The class Company is a general form of the PublicOwnedCompany , PrivateCompany and PrivateandPublic classes. This allows the designers to just use the Company class for modules and do not require in-depth representation of each type of Company.

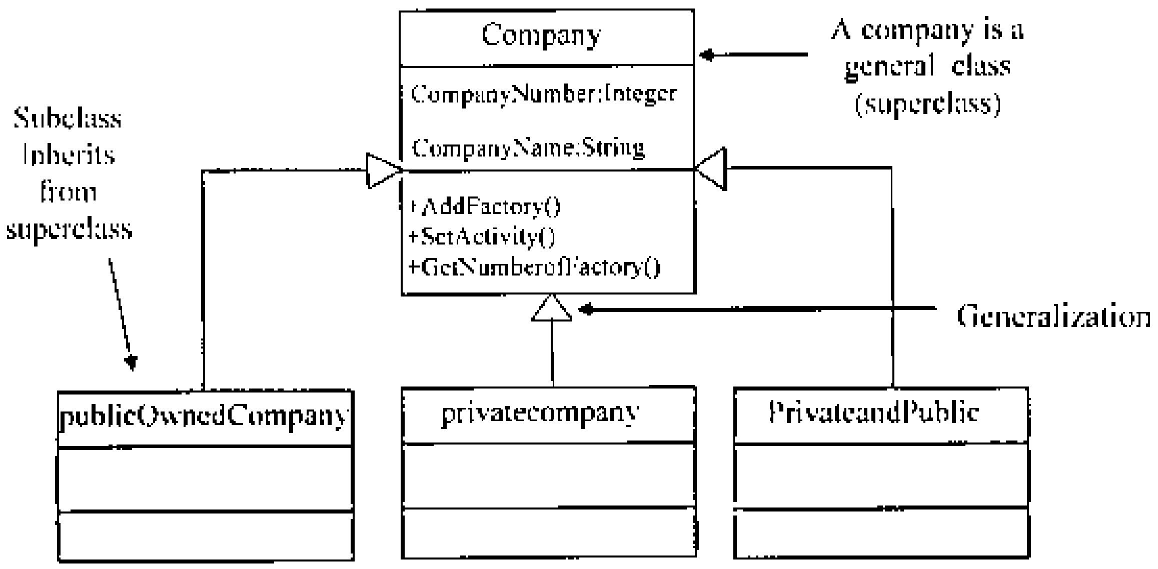

Figure 4.9 an example of a generalization relationship (UML class diagram)

4.3.2.2.5 Aggregation relationships

A plain association between two classes represents a structural relationship between peers, meaning that both classes are conceptually at the same level, no one more important than the other. Sometimes, you will want to model a "whole/part" relationship, in which one class represents a larger thing (the "whole"), which consists of smaller things (the "parts"). This kind of relationship is called aggregation, which represents a "has-a" relationship, meaning that an object of the whole has objects of the part. Aggregation is really just a special kind of association and is specified by adorning a plain association with an open diamond at the whole end, as shown in Figure 4.11

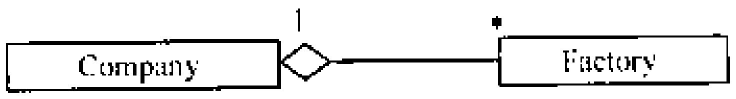

Figure 4.10 aggregation

A composition relationship represents a whole-part relationship and is a type of aggregation. A composition relationship specifies that the lifetime of the part classifier is dependent on the lifetime of the whole classifier, as figure 4.12

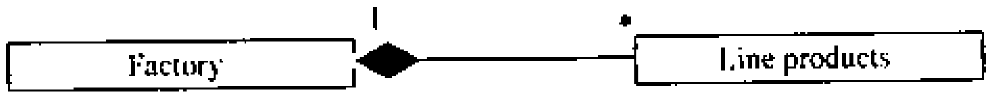

Figure 4.11 composition (a factory composed of Line products)

4.4 Module interface specification

4.4.1 User interface subsystem

The User interface (UI) subsystem is to present the graphic user interface to the system users. UI receives the input from users and sends the user requests to the server. This subsystem contains one class UI. (Figure 4.13)

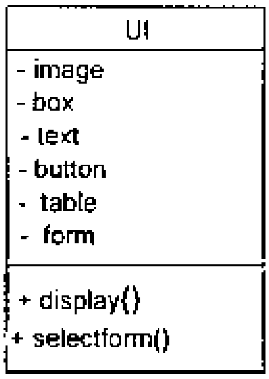

Figure 4.12 The Class diagram of UI

4.4.2 Database subsystem

This subsystem stores all tables specified in the database design.
Database subsystem contains one class DB. (Figure 4.14)

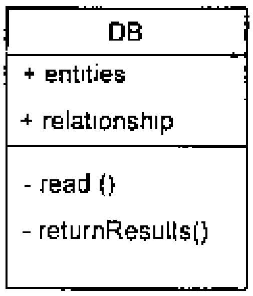

Figure 4.13 The Class Diagram of DB

4.4.3 System package

The package system contains five components: Public User, Operator, Company and staff, Administrator, and Monitor . These components are connected only at the database level, therefore each component can be developed separately.

4.4.3.1 PublicUser component

The PublicUser component provides functions of part of the system that is available to public users. It includes four classes..(Figure 4.15)

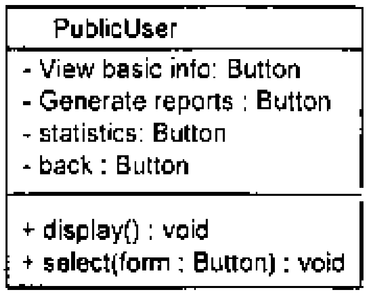
PublicUser
- View basic info: Button
- Generate reports : Button
- statistics: Button
- back : Button
+ display() : void
+ select(form : Button) : void

Figure 4.14 The Class Diagram of Public User

4.4.3.2 Operator component

The operator component provides functions of part of the system that is available to operator

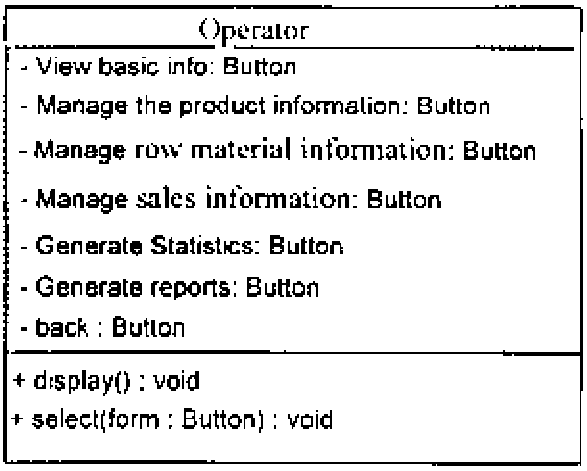
Operator
- View basic info: Button
- Manage the product information: Button
- Manage row material information: Button
- Manage sales information: Button
- Generate Statistics: Button
- Generate reports: Button
- back : Button
+ display() : void
+ select(form : Button) : void

Figure 4.15 The class diagram of operator

4.4.3.3 Company component

The company component provides all functions required for this part of the system . It contains the following classes:

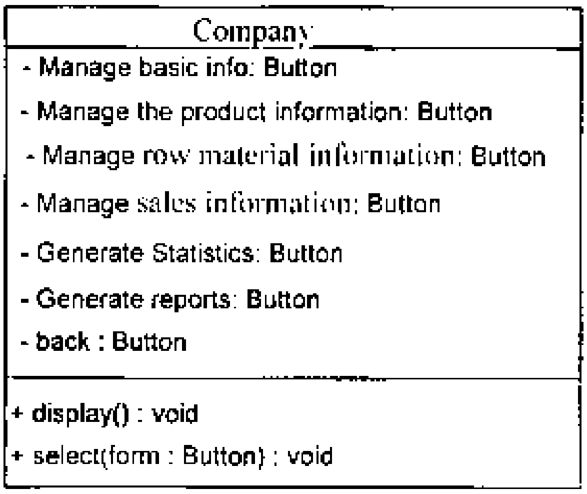
Company
- Manage basic info: Button
- Manage the product information: Button
- Manage row material information: Button
- Manage sales information: Button
- Generate Statistics: Button
- Generate reports: Button
- back : Button
+ display() : void
+ select(form : Button) : void

Figure 4.16 The class diagram of operator

4.4.3.4 Administrator component

This component provides all of the functions available to administrators

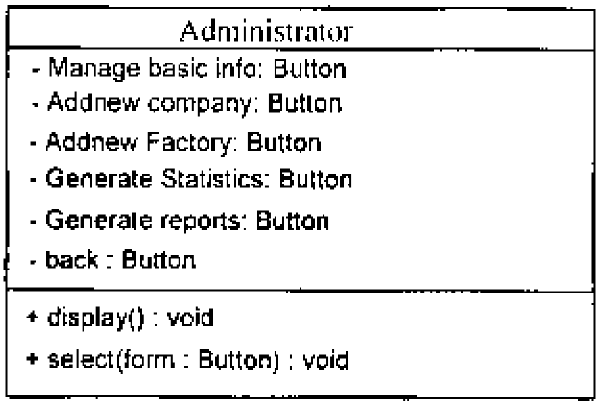
Administrator
- Manage basic info: Button
- Addnew company: Button
- Addnew Factory: Button
- Generate Statistics: Button
- Generate reports: Button
- back : Button
+ display() : void
+ select(form : Button) : void

Figure 4.17 The Class diagram of administrator

4.5 Application architecture

Application architecture defines what an application looks like at the end of the development process. Because of this, application architecture is very similar to the blueprints. In this study two-tier application architecture is used.

Two-tier application architecture is also called as client/server architecture and consists of two layers. One of the layers is client which request service and the other layer is the server which processes the requests that come from client. In this architecture, the client part is called as a fat client because all the presentation service and the business logic are running on this part. The server side only maintains the database operations.

The main advantage of the two tier application architecture is the centralized database.

4.5.1 Database implementation

In this study Microsoft SQL Server 2005 is chosen as a database management system.

Microsoft SQL Server supports client/server architecture. The main features of SQL Server are:

The main features of SQL Server are;

- It can be hosted by Win9x and NT operating systems.
- It is scalable. It can run both on small scale desktop computer and large server system that has 32 processors.
- It has replication ability.
- The OLAP (On-Line Analytical Processing) services can be given.
- Importing and exporting data can be possible by data transformation services.

4.5.2 Client side implementation

The client side implementation of the Industrial information system is developed on the Visual Basic .NET which is a component of the Visual Studio .NET.

Visual Studio .NET is the complete set of development tools for rapid development of enterprise-scale ASP web applications, high performance desktop applications, XML web services and mobile applications . It includes component-based development tools such as Visual C#, Visual Basic and Visual C++. Moreover, it includes additional tools in order to simplify the team-based design, development and deployment of solutions. All of the tools share the same integrated development environment (IDE), which allow them to share the tools and facilities to create the mixed-language solutions.

Chapter 5

Integration and system testing

Testing is the process of finding differences between the expected behavior specified by system models and the observed behavior of the system. testing Unit finds differences between the object design model and its corresponding component. Structural testing finds differences between the system design model and a subset of integrated subsystems. Functional testing finds differences between the use case model and the system. Finally, performance testing finds differences between nonfunctional requirements and actual system performance. When differences are found, developers identify the defect causing the observed failure and modify the system to correct it. In other cases, the system model is identified as the cause of the difference, and the model is updated to reflect the state of the system [1].

5.1 Levels of testing in software development

Testing is applied at different levels of software development in the life cycle, but the testing done is different in nature and has different objectives at each level. The focus of all of development testing is to find errors, but different types of error are looked for at each level [16].

5.1.1 Testing unit

The objective of testing unit is to find errors in individual modules. Testing unit is usually considered part of the coding process and usually requires a significant investment in scaffolding, as illustrated in Figure 5.1 [17].

Testing unit is often performed by the programmer who wrote the code, although this is not necessarily the most effective alternative. Tests can be derived from the detailed logic of the unit (the detailed design specification), with any additional structural tests derived from the physical design, as a result of coverage measurement [16].

Because a component is not a stand-alone program, driver and/or stub software must be developed for each unit test. In most applications a driver is nothing more than a "main program" that accepts test case data, passes such data to the component (to be tested), and prints relevant results. Stubs serve to replace modules that are subordinate to the component to be tested. A stub or "dummy subprogram" uses the subordinate module's interface, may do minimal data manipulation, provides verification of entry, and returns control to the module undergoing testing [12].

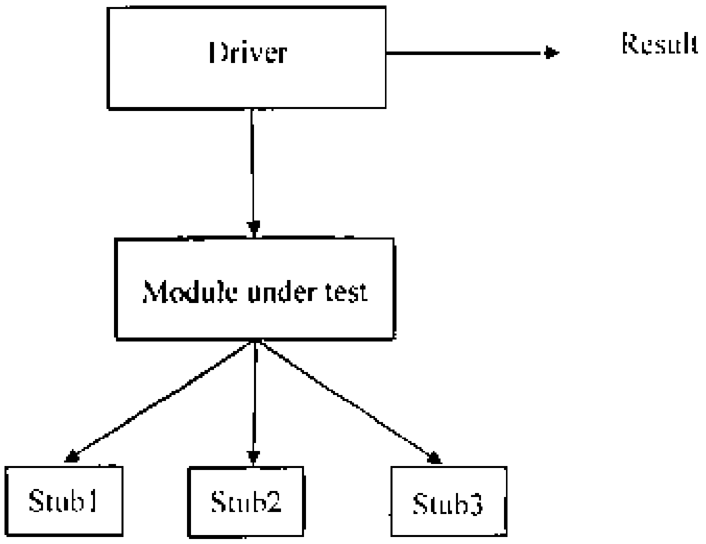

Figure 5.1 Testing unit environment

5.1.2 Integration testing

When two or more tested components are combined into a larger structure, the tests should look for errors in two ways: in the interfaces between the components, and in the functions which can be performed by a new group which was not assessable for the individual components or units forming the group [16].

The objective of integration testing is to find bugs related to interfaces between modules as they are integrated. One question is frequently asked: If all modules are testing unit, why is integration testing necessary? Here are some answers:

- One module can adversely affect another module.
- Sub functions, when combined, may not produce the desired major function.
- Individually acceptable imprecision in calculations may be magnified to unacceptable levels.
- Interfacing errors not detected in testing unit may appear.
- Timing problems (in real-time systems) are not detectable by unit testing.
- Unit testing cannot detect resource contention problems.

Integration testing covers a broad range of testing, beginning with the testing of a few modules and culminating with the testing of the complete system. Let us look briefly at different approaches to integration testing [17].

5.1.2.1 Top-down integration

Top-down integration testing is an incremental approach to construction of the software architecture. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program). Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner see figure 5.2 [12].

The integration process is performed in a series of five steps:

1. The main control module is used as a test driver, and stubs are substituted for all components directly subordinate to the main control module.
2. Depending on the integration approach selected (i.e., depth or breadth first),

subordinate stubs are replaced one at a time with actual components.

3. Tests are conducted as each component is integrated.
4. On completion of each set of tests, another stub is replaced with the real component.
5. Regression testing may be conducted to ensure that new errors have not been introduced [17].

5.1.2.2 Bottom-up integration

Bottom-up integration testing, as its name implies, begins construction and testing with atomic modules (i.e., components at the lowest levels in the program structure). Because components are integrated from the bottom up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated. A bottom-up integration strategy may be implemented with the following steps:

1. Low-level components are combined into *clusters* (sometimes called *builds)* that perform a specific software sub function.
2. A driver (a control program for testing) is written to coordinate test case input and output.
3. The cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure [12].

The advantage of bottom-up testing is that interface faults can be more easily found: When the developers substitute a test driver by a higher level component, they have a clear model of how the lower level component works and of the assumptions embedded in its interface. The disadvantage of bottom-up testing is that it tests the most important subsystems, namely the components of the user interface, last. First, faults found in the top layer may often lead to changes in the subsystem decomposition or in the subsystem interfaces of lower layers, invalidating previous tests. Second, tests of the top layer can be derived from the requirements model and, thus, are more effective in finding faults that are visible to the user [4].

The advantage of top-down testing is that it starts with user interface components.

The same set of tests, derived from the requirements, can be used in testing the increasingly more complex set of subsystems. The disadvantage of top-down testing is that the development of test stubs is time consuming and prone to error. A large number of stubs is usually required for testing nontrivial systems, especially when the lowest level of the system decomposition implements many methods [4].

5.1.3 Validation testing

The objective of software validation testing is to determine if the software meets its requirements as defined in the software requirement specification (SRS) and other relevant documents. Achieving that objective is based on the premise that software development and software validation tests are both based on the SRS.

The software validation testing process can be broken down into five phases:

1. Test planning.
2. Test development (informal validation).
3. Software validation readiness review.
4. Test execution (formal validation).
5. Test completion criteria.

5.1.4 System testing

Unit and integration testing focus on finding faults in individual components and the interfaces between the components. Once components have been integrated, system testing ensures that the complete system complies with the functional and nonfunctional requirements of the system. There are several system testing activities that are performed:

5.1.4.1 Functional testing

Functional testing, also called requirements testing, finds differences between the functional requirements and the system. System testing is a blackbox technique: Test cases are derived from the use case model. In systems with complex functional requirements, it is usually not possible to test all use cases for all valid and invalid inputs. The goal of the tester is to select those tests that are relevant to the user and have a high probability of uncovering a failure [4].

5.1.4.2 Performance testing

Performance testing finds differences between the design goals selected during system design and the system. Because the design goals are derived from the nonfunctional requirements, the test cases can be derived from the SDD or from the RAD. The following tests are performed during performance testing.

- Stress testing checks if the system can respond to many simultaneous requests.
- Volume testing attempts to find faults associated with large amounts of data. such as static limits imposed by the data structure, or high-complexity algorithms, or high disk fragmentation.
- Security testing attempts to find security faults in the system.
- Timing tests attempts to find behaviors that violate timing constraints described by the nonfunctional requirements
- Recovery tests evaluate the ability of the system to recover from errors. such as the unavailability of resources, a hardware failure, or a network failure[4].

5.1.4.3 Pilot testing

During the pilot test, also called the field test, the system is installed and used by a selected set of users. Users exercise the system as if it had been permanently installed. No explicit guidelines or test scenarios are given to the users [4].

5.1.4.4 Acceptance testing

There are three ways the client evaluates a system during acceptance testing. In a benchmark test, the client prepares a set of test cases that represent typical conditions under which the system should operate. Benchmark tests can be performed with actual users or by a special test team exercising the system functions, but it is important that the testers be familiar with the functional and nonfunctional requirements so they can evaluate the system.[4]

5.1.4.5 Installation testing

After the system is accepted, it is installed in the target environment. A good system testing plan allows the easy reconfiguration of the system from the development environment to the target environment. The desired outcome of the installation test is that the installed system correctly addresses all requirements [4].

Abstract

In this thesis, we will present an example of industrial information system design and implementation. The most common software engineering design steps will be applied to the different design stages. We will be going through the life cycle of software system development. We start by a study of system requirement and end with testing and delivering system, going by system design and coding, program integration and system integration step. The most modern software design tools available will be used in the design, Unified Modeling Language (UML), system modeling, SQL sever side application, uses case analysis, design and testing as applied to information processing systems. The system will be designed to perform tasks specified by the client with real data. By the end of the implementation of the system, default or user defined acceptance policy to provide an overall score as an indication of the system performance will be used. To test the reliability of the designed system, it will be tested in different environment and different work burden such as multi user environment.

الملخص

استخدام هندسة البرمجيات الشيئية لتصميم وتنفيذ نظم المعلومات الصناعية

تقدم هذه الأطروحة مثال لتصميم وتنفيذ نظام للمعلومات الصناعية ، وتم فيها تطبيق خطوات تصميم برامج تصميم البرمجيات في مختلف مراحل تصميم نظامنا.

تم دراسة دورة حياة تطوير النظام بداية من تحديد متطلبات النظام وانتهاءً باختبار وتسليم النظام مرورا بتصميم وكتابة شفرة النظام وتكامل البرنامج وخطوات تكامل النظام.

تم استخدام أدوات تصميم النظام الأكثر إتاحة مثل لغة النمذجة الموحدة والتي تعرف اختصارا (UML) والتي تستخدم كلغة نمذجة وترميز باستخدام حالة الاستخدام في مرحلة تحليل وتصميم النظام .

النظام تم تصميمه لتأدية وظائف محددة من قبل الزبون لتوظيف بيانات حقيقية ، وفي نهاية تطبيق النظام تم اختباره من قبل الزبون وفي بيئة مختلفة باستخدام شبكة متعددة المستخدمين .

العنوان:	Object oriented sottwore engineering approach to industriol intormotion system design and implementation
المؤلف الرئيسي:	مانيطة، عيسى حسين أحمد
مؤلفين آخرين:	الفقيه، إدريس ساسد(مشرف)
التاريخ الميلادي:	2008
موقع:	سرت
الصفحات:	1 - 78
رقم MD:	859171
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة التحدي
الكلية:	كلية العلوم
الدولة:	ليبيا
قواعد المعلومات:	Dissertations
مواضيع:	نظم المعلومات، هندسة البرمجيات، البرمجيات الشيئية، نظم المعلومات الصناعية
رابط:	https://search.mandumah.com/Record/859171

Table of contents

Object oriented software engineering approach to industrial information system design and implementation

Prepared By :

Issa Hessin Manita

Supervised By :

Dr. Idris S. El-Feghi

This thesis submitted in partial fulfillment of the requirements for the degree of master in computer science

Al-Tahadi university

Faculty of science

Department of computer science

Academic year Fall 2008

الجماهيرية العربية الليبية الشعبية الاشتراكية العظمى

جامعة التحدي

سرت

إن الدراسة ليست غاية في حد ذاتها
وإنما الغاية هي خلق الإنسان النموذجي الجديد

التاريخ : ....

الموافق : 2008/12/21 مسيحي ....

الرقم الإشاري : ج ت س/ك ع/2008/1/812

Faculty of Science

Department Of Computer Science

Title of Thesis

Object Oriented Software Engineering Approach to Industrial Information System Design and Implementation

By

Issa Hessin Manita

Approved by:

Dr. Idris S. El-Feghi (Supervisor) .... [illegible]

Dr. Faraj A. El-Mouadib (External examiner) .... [illegible]

Dr. Zakaria Suliman Zubi (Internal examiner) 2008/11/27

Countersigned by:

Dr. Ahmed Farag Mhgoub

(Dean of faculty of science)

[illegible]

جامعة التحدي كلية العلوم

WWW.altahadi.edu.ly

Abstract

In this thesis, we will present an example of industrial information system design and implementation. The most common software engineering design steps will be applied to the different design stages. We will be going through the life cycle of software system development. We start by a study of system requirement and end with testing and delivering system, going by system design and coding, program integration and system integration step. The most modern software design tools available will be used in the design, Unified Modeling Language (UML), system modeling, SQL sever side application, uses case analysis, design and testing as applied to information processing systems. The system will be designed to perform tasks specified by the client with real data. By the end of the implementation of the system, default or user defined acceptance policy to provide an overall score as an indication of the system performance will be used. To test the reliability of the designed system, it will be tested in different environment and different work burden such as multi user environment.

Dedication

To my family, my wife and my friends I dedicate this work

Acknowledgment

I would like first to thank Allah (SWT) for his guidance. Without his support, this work would have never been completed.

Secondly, I would like to express my sincere thanks and appreciation to my supervisors, Dr. Idris. S. El-Feghi for his invaluable advices, guidance, and constant encouragement and support through the progress of this thesis.

I would like also to thank my friends and colleagues in the Computer Science Department at Al-Tahdi University for happy memories we had during my work on this thesis. My thanks also go to the staff at the Computer Science department at al-Tahdi University for their help and encouragement.

Finally, I would like also to thank the Industrial Information Center's manager Eng. Abdulhamed Alshareif for his continuous encouragement and help.

Last but not least, my thanks go to my family for their encouragement and support.

Table of contents

List of figures

List of table

Chapter 1

Introduction

One of the largest application areas for computers is the storage and retrieval of data. We call this class of systems "information-based systems" or simply "information systems" because the primary purpose of such systems is managing information. Examples of information systems are banking systems, library-cataloguing systems, and personal information systems. At the heart of such systems is a database against which we apply transaction to create, retrieve, update, or delete items. Many systems provide a web interface to operate on the information[1].

Information systems have gained importance because of the increasing volume of information as resources. The data that these systems manage is often the most valuable resource of an enterprise. Such data concern both the processes and the resources internal to the enterprise-plants, goods, people, etc... as well as information on external sources-competitors, suppliers, clients, etc [1].

Information systems are data oriented that can be characterized on the basis of the way they treat data. The following are some of the qualities that characterize the information system

- **Data integrity**. Under what circumstances will the data be corrupted when the system malfunctions?
- **Security**. To which extent does the system protect the data from unauthorized access?
- **Data availability**. Under what conditions will the data become unavailable and for how long?

- **Transaction performance.** Because the goal of information systems is to support transactions against data, the performance of such systems can be uniformly characterized in terms of the number of transactions carried out per unit of time [1].

1.1 Types of information system

The applications of information systems in the real world can be classified in several different ways. For example, several types of information systems can be classified as either operations or management information systems. Figure 1.1 illustrates this conceptual classification of information systems applications.

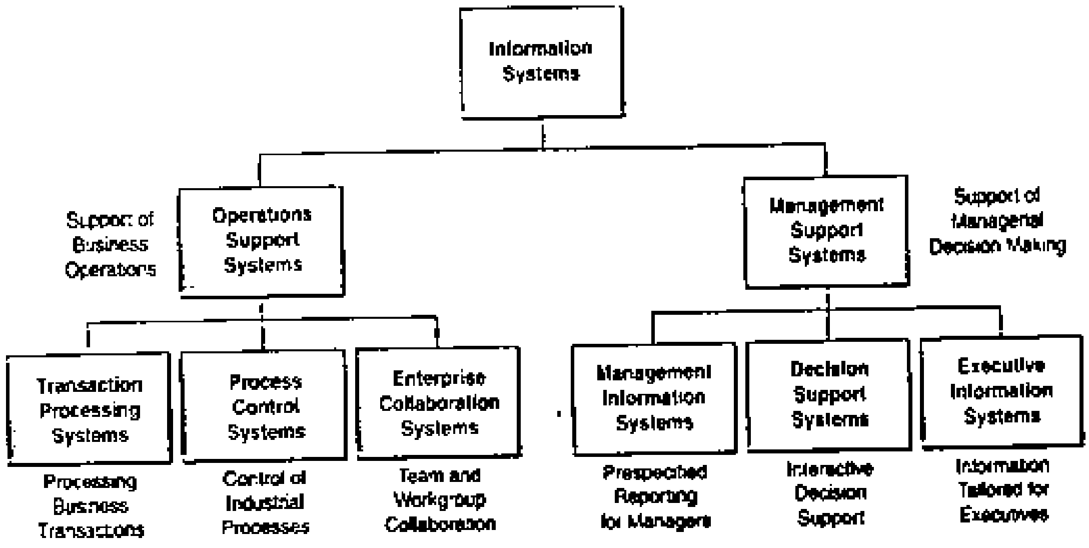
Information Systems
Support of Business Operations
Operations Support Systems
Management Support Systems
Support of Managerial Decision Making
Transaction Processing Systems
Process Control Systems
Enterprise Collaboration Systems
Management Information Systems
Decision Support Systems
Executive Information Systems
Processing Business Transactions
Control of Industrial Processes
Team and Workgroup Collaboration
Prespecified Reporting for Managers
Interactive Decision Support
Information Tailored for Executives

Figure 1.1 Operation and management classification of information systems

Information systems have always been needed to process data generated by, and used in, business operations. Such **operations support systems** produce a variety of information products for internal and external use. However, they do not emphasize producing the specific information products that can best be used by managers.

Further processing by management information systems is usually required. The role of a business firm's operations support systems is to efficiently process business transactions, control industrial processes, support enterprise communications and collaboration, and update corporate databases.

Transaction Processing System (TPS) are an important example of operations support systems that record and process data resulting from business transactions. They process transactions in two basic ways. In batch processing , transaction a data are accumulated over a period of time and processed periodically. In *real-time* (or online) processing, data are processed immediately after a transaction occurs. For example, Point-Of-Sale (POS) systems at many retail stores use electronic cash register terminals to electronically capture and transmit sales data over telecommunications links to regional computer centers for immediate (real-time) or nightly (batch) processing.

Process control systems monitor and control physical processes. For example, a petroleum refinery uses electronic sensors linked to computers to continually monitor chemical processes and make instant (real-time) adjustments that control the refinery process.

Enterprise collaboration systems enhance team and workgroup communications and productivity, and sometimes called office automation systems.

When information system applications focus on providing information and support for effective decision making by managers, they are called **management support systems**. Providing information and support for decision making by all types of managers and business professionals is a complex task. Conceptually, several major types of information systems support a variety of decision-making responsibilities: (1) management information systems, (2) decision support systems, and (3) executive information systems [2].

Management information systems that take information captured by transaction processing systems and produce reports that management needs for planning and controlling the business. Management information systems are possible because the information has been captured by the transaction processing systems and placed in organizational databases [3].

Decision support systems give direct computer support to managers during the decision-making process. For example, advertising managers may use an electronic spreadsheet program to do what if analysis as they test the impact of alternative advertising budgets on the forecasted sales of new products.

Executive information systems provide critical information from a wide variety of internal and external sources in easy-to-use displays to executives and managers. For example, top executives may use touchscreen terminals to instantly view text and graphics displays that highlight key areas of organizational and competitive performance [2].

1.2 Two approaches of system development

System development is done in many different ways. this diversity can confuse new employees when they works a system developers. sometimes it seems that every company develops information systems has its own approach. sometimes different development groups within the same company uses different approaches, and each person in the company may have his or her own way of developing the system.

All system developers should be familiar with two general approaches to system development, because they form the basis of virtually all methodologies, the traditional approach and the object-oriented approach.

1.2.1 The traditional approach

The traditional approach includes many variations based on techniques used to develop information systems with structured and modular programming. This approach is often referred to as structured system development. A refinement to structured development approach, is called Information Engineering (IE), a popular variation.

- **structured system development approach**

Structured analysis, structured design, and structured programming are the three techniques that make up the structured approach. Sometimes these techniques are collectively referred to as the Structured Analysis and Design Technique (SADT) , the structured programming technique developed in the 1960s. was the first attempt to provide guidelines to improve the quality of computer programs. The structured design technique was developed in the 1960s, to makes it possible to combine separate programs into more complex information systems. the structured analysis technique evolved in the early 1980s is to help clarify the requirements for the computer system to the developers before they designed the programs.

- **Structured Programming**

High-quality programs are not only to produce the correct outputs each time the program runs but also make it easy for other programmers to read and modify the program later. it programs need to be modified all the time. A structured program is one that has one beginning and one end, and each step in the program execution consists of one of three programming constructs:

1. A sequence of program statements.
2. A decision where one set of statements or another set of statements to the execute.
3. A repetition of a set of statements [3].

A complex system may be divided into simpler pieces called modules. A system that is composed of modules is called modular. The main benefit of modularity is that it allows the principle of separation of concerns to be applied in two phases: when dealing with the details of each module in isolation (and ignoring details of other modules) and when dealing with the overall characteristics of all modules and their relationships in order to integrate them into a coherent system. If the two phases are executed in sequence first by concentrating on modules and then on their composition. then we say that the system is designed bottom up; the converse when we decompose modules first and then concentrate on individual module design is top down design [1].

- **Structured design**

As information systems continued to become increasingly complex through the 1970s, each system involved many different functions. Each function performed by the system might be made up of dozens of separate programs. The structure design technique was developed to provide some guidelines for deciding what the set of programs should what each program should accomplish, and how the programs should be organized into a hierarchy.

Two main principles of structured design are that program modules should be designed so they are (1) loosely coupled and (2) highly cohesive. Loosely coupled means that each module is as independent of the other modules as possible, which allows each module to be designed and later modified without interfering with the performance of the other modules. Highly cohesive means that each module accomplishes one clear task. That way, it is easier to understand what each module does and to ensure that if changes to the module are required, none will accidentally affect other modules [2].

Information engineering is a refinement to structured development that begins with overall strategic planning to define all of the information systems that the organization needs to conduct its business (the application architecture plan). The plan also includes a definition of the business functions and activities that the systems need to support, the data entities about which the systems need to store information, and the technological infrastructure that the organization plans to use to support the information systems.

Each new system project begins by using the defined activities and data entities created during strategic systems planning. The activities and data are refined in the project progresses. At each step, the project team creates models of the processes, the data, and the ways they are integrated.

The information engineering approach refines many of the concepts of the structure approach into a rigorous and comprehensive methodology. Both approaches define information systems requirements, design of information systems, and construct information systems by looking at processes, data, and the interaction of the two.

This text merges key concepts from these two approaches into one, which we will refer to hereafter to traditional approach. The traditional approach. in one version or another, is still widely for information system development. although many information systems project. Using Object-Oriented (OO) technology-which requires a completely different approach [6].

1.2.2 The object-oriented approach

An entirely different approach to information systems. the object-oriented approach view an information system as a collection of interacting objects that work together to accomplish tasks [2]. This means that objects in a computer system, like objects in the world around us, are view as thing. These things have certain features, or attributes, and they can exhibit certain beavers [7]. There are no processes or programs; there are no data entities or files. The system consists of objects, an object is a thing in the computer system that is capable of responding to messages, this radically different view of a computer system requires a different approach to systems analysis, systems design, and programming.

Because the object-oriented approach views information systems as collections of interacting Object-Oriented Analysis (OOA) defines all of the types of objects that do the work in the system and shows how the objects interact to complete tasks. Object-Oriented Design (OOD) defines all of the additional types of objects necessary to communicate with people and devices in the system and refines the definition of each type of object so it can be implemented with a specific language or environment. Object-Oriented programming (OOP) consists of writing statements in a programming language to define what each type of object does [3].

What is different about the object-oriented approach? The object-oriented approach is based on a view of computer systems that is fundamentally different from that of the structured approach. Once the business events involving the system have been identified. defining what the user requires means defining all of the types of objects that are part of the user's work environment (object-oriented analysis).

Second, to design a computer system means to define all the additional types of objects involving the user interface and operating environment of the computer system and the ways they interact with objects in the user's work environment (object-oriented design). Third, programmers must write statements that define all types of objects, including their attributes and behaviors (object-oriented programming). Therefore, in some ways, everything is different with the object-oriented approach [7].

1.3 Research objectives

This research have several objectives which can be summirized as follows:.

- Understanding of large system design such industrial information systems for which all software design of large scale are applied.
- Understanding and applying the different phases of system developments life cycle as applied to large scale system development and implementation.
- Use the object oriented software engineering to provide a system requirement definition and system requirement specification using some evolutionary and throw-away prototyping models to satisfy user needs.
- Useing Unified Modeling Language(UML) in requirement specification and in system design and implementation.
- Study of life cycle of software engineering project and applying knowledge that has been learned in the field.
- Compare the design approach using OOD and OOP with that obtained using structural programming and how the new system approach can be used to replace the existing system with as less changes as possible.
- Among the different software life cycles, the most appropriate life cycle that fits the needs for such systems will be selected and used to provide a reasonable reasoning for selecting such a model.

1.4 Thesis Organization

The thesis consist of 6 chapters. Chapter 1 includes introduction to the information management system.

In chapter 2 litrature review of object-oriented analysis and design with UML.

In chapter 3 system analysis and speification of industrial information system.

In chapter 4 system designing of the industrial information system.

Im Chapter 5 about coding testing unites of the system as well as the integrated testing of the system and implementation.

Finally Chapter 6 discusses the conclusions and future work that can carried out on industrial information system.

Chapter 2

Literature review

2.1 Overview

The System Development Life Cycle (SDLC) is a widely used framework for organizing and managing the process. A system development life cycle typically defines phases that are completed by the project team as they move from the beginning to the end of the development project. The term life cycle is used because every information development project has a beginning and eventually an ending. Between these points in time. the project "lives" in one form or another.[7]

The object modeling approach prescribes the use of methodologies and diagramming notations that are completely different from the ones used for data modeling and process modeling. In the late 80s and early 90s many object-oriented methods were being used throughout industry. The existence of so many methods and associated modeling techniques was a major problem for the object-oriented. system development industry. It was not uncommon for a developer to have to learn several object modeling techniques depending on what was being used at the time. This use of different object modeling techniques limited the sharing of models across projects (reduced reusability) and development teams. It hampered communication between team members and the users, which led to many errors being introduced into the project. These problems and others led to the effort to design a standard modeling language[13].

The Unified Modeling Language (UML) is a set of modeling conventions that is used to specify or describe a software system in terms of objects [6].

The UML does not prescribe a method for developing systems-only a notation that is now widely accepted as a standard for object modeling. The Object Management Group (OMG), an industry standards body, adopted the UML in November 1997 [8].

Because the object-oriented approach views information systems as collections of interacting Object, Object-oriented Analysis (OOA) defines all of the types of objects that do the work in the system and shows how the objects interact to complete tasks. Object-Oriented Design (OOD) defines all of the additional types of objects necessary to communicate with people and devices in the system and refines the definition of each type of object so it can be implemented with a specific language or environment. Object-Oriented Programming (OOP) consists of writing statements in a programming language to define what each type of object does [3].

2.2 The unified doftware development process

The Unified Software Development Process (in short, Unified Process, or UP) has its roots in the industrial experience within Ericsson manufacture, A manufacturer of telecommunication equipment, in the late 1960s. Subsequently, that experience led to successor methodologies developed by two other companies: Objectory and Rational. gained the status of a widely adopted industrial standard [1].

The Rational Unified Process (RUP) is an object-oriented system development methodology offered by Rational Software. RUP is their attempt to define a complete methodology that, in addition to providing several unique features, uses UML for system models. In RUP, the term development process is synonymous with development methodology [3].

UP uses the unified modeling language throughout the software life cycle. As we have observed, the rich collection of languages that constitute UML provides specific notations to specify, analyze, visualize, construct, and document the artifacts that are developed in the life cycle of a software system. Software engineering is the notations to produce standardized blueprints that contain a number of different diagrams, each enlightening a certain aspect of the application being developed. UML is, however, an important step in direction of a standardized language that would allow all software engineers to communicate and interact on rigorous grounds while developing or evolving a given software system [1].

2.3 The phases of the unified process

UP consist of four phases: Inception , Elaboration, Construction, and Transition (see figure 2.1).

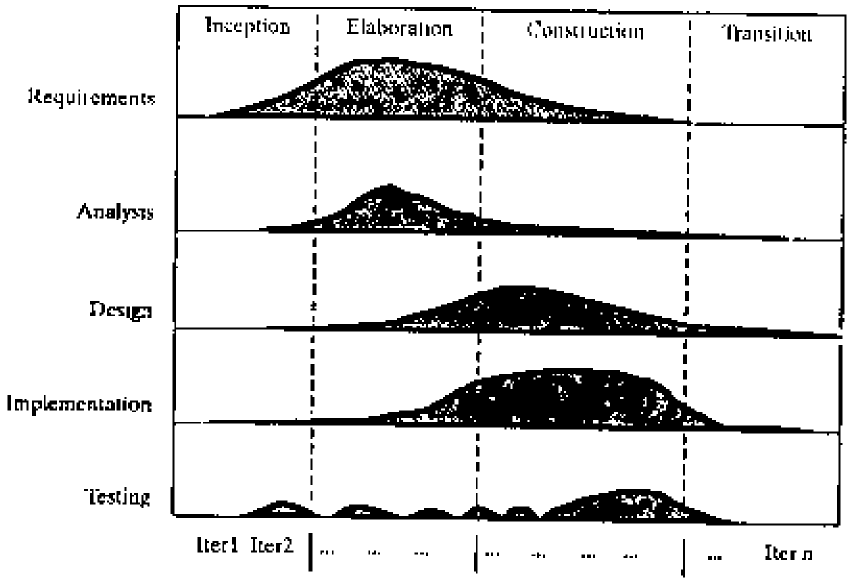

Figure 2.1 Activity (workflows), with phases and iteration.

2.3.1 The inception phase

The purpose of the inception phase is to determine whether it is worthwhile to develop the target information system. In other words, the primary aim of this phase is to determine whether the proposed information system is economically viable [6]. The inception phase defines the scope of the project by specifying use cases (which are similar to the user stories) as with any OO development approach. The project team also completes a feasibility study to determine whether resources should be invested in the project [3] .

Chapter 6

Conclusion and future work

6.1 Conclusion

The development of a new system requires an organized, step-by-step approach. We call this approach the systems development life cycle. The SDLC defines the overall approach to system development. The SDLC is divided into four development phases and one support phase. For development, the phases are project planning, analysis, design, and implementation. Each of these phases is further divided into activities, and the activities are divided into individual tasks. In this way, analysts can identify and schedule the individual work tasks that make up the project.

System requirements define the services the system is to provide and prescribe constraints for its operation. In documenting the system requirements for a new information system, an analyst will likely identify dozens of unique requirements. To simplify the presentation of the requirements and to make them more readable, understandable, and traceable, requirements are often categorized as functional versus nonfunctional.

There are seven common fact-finding techniques.

- The sampling of existing documents and files can provide many facts and details with little or no direct personal communication being necessary. The analyst should collect historical documents, business operations manuals and forms, and information systems documents.

- Research is an often-overlooked technique based on the study of other similar applications. It now has become more convenient with the Internet and World Wide Web. Site visits are a special form of research.
- Observation is a fact-finding technique in which the analyst studies people doing their jobs.

To ferret out the requirements, analysts must work with various stakeholders in the new system. We categorize stakeholders into three groups: (1) the users, those who will actually use the system day to day, (2) the clients, those who pay for and own the system, and (3) the technical staff, the people who must ensure that the system operates within the computing environment of the organization. One of the most important first steps in determining systems requirements is to identify these various system stakeholders.

The traditional approach uses the entity-relationship diagram to show data entities, attributes of data entities, and relationships. The object-oriented approach uses the class diagram to show the classes, attributes, and methods of the class, and associations among classes. Two additional concepts are used in class diagrams (although they are sometimes used in entity-relationship diagrams, too): generalization/specialization hierarchies, which allow inheritance from a super class to a subclass, and whole-part hierarchies, which allow a collection of objects to be associated as a whole and its parts.

A relational database is a collection of data stored in tables. A relational database scheme is normally developed from an entity-relationship diagram. Each entity is represented as a separate table. One-to-many relationships are represented by embedding foreign keys in entity tables. Many-to-many relationships are represented by creating additional tables containing foreign keys of the related entities.

An object database stores data as a collection of related objects. The design class diagram starting point for developing an object database schema. The database schema defines each and the ODBMS stores each object as an instance of a particular class. Each object is assigned a unique object identifier. Relationships among objects are represented by storing the object identifier an object within related objects.

6.2 *Future work*

There are many aspects of the work that can be carried further. The use of UML was used in the design of the system, however, the test for automatic code generation using the UML software can be further developed to measure the speed of software development process. Dedicated studies to software measurements can be further developed and recommendation for the best software life cycle that can fit different types of software can be classified.

In this research, it was difficult to measure the different software quality assurance such as compatibility, portability, adaptability, and learnability which was due to the fact that there are no benchmarks for software quality assurance measurements. We recommend that a new study to the quality assurance of software should be carried out in the future.

There are many human factors which are used to test any software such as good GUI design and use of screens. Such factors depend of personal preferences. It would be helpful if a new study to the user preferences is performed and different form for the preferred designs are studied and measured by interviewing large number of people involved in the software development or use.

العنوان:	Object oriented sottwore engineering approach to industriol intormotion system design and implementation
المؤلف الرئيسي:	مانيطة، عيسى حسين أحمد
مؤلفين آخرين:	الفقيه، إدريس ساسد(مشرف)
التاريخ الميلادي:	2008
موقع:	سرت
الصفحات:	1 - 78
رقم MD:	859171
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة التحدي
الكلية:	كلية العلوم
الدولة:	ليبيا
قواعد المعلومات:	Dissertations
مواضيع:	نظم المعلومات، هندسة البرمجيات، البرمجيات الشيئية، نظم المعلومات الصناعية
رابط:	https://search.mandumah.com/Record/859171

References

[1]	Carlo ghezzi , Mehdi jazayeri, Dino mandrioli, Fundamentals of software engineering, 2003
[2]	James A. O'Brien, McGraw-hill Irwin, Introduction to information systems Essential for the e-business Enterprise, 2003
[3]	John W.Satzinger, Southwest Missouri State University, Robert B. Jackson, Brigham Young University, Steohen D. Burd, University of New Mexico, " System Analysis and Design, in a changing world ", course Technology, 2002.
[4]	Bernd Bruegge & Allen H. Dutoit, object-Oriented Software Engineering, Conquering Complex and Changing System, Technical University of Munich, Department of Computer Science, Munich, Germany, 2000
[5]	Pfleeger S. "Software Engineering Theory and Practice", Upper Saddle River, Nj, Prentice Hall, 2001
[6]	Stephen R. Schach, Vanderbilt University , Introduction to Object-Oriented Analysis And Design With UML And Unified Process, (2004)
[7]	John w. satzinger , tore u. orvik , the object-Oriented approach concepts, system development, and modeling with uml, course Technology, 2001
[8]	Stephen R. Schach , Vanderbilt university, "object-oriented and classical software engineering " , McGraw-hill Higher Education , 2005
[9]	Christopher fox, "Introduction to software engineering design processes, principles, and patterns with UML2, " Pearson Addison Wesley, 2006
[10]	Booch G atl. OMG Unified Modeling Language Specification, Rational Software Corporation, 2003
[11]	Dustin E. Effective software testing , 50 specific ways to improve your testing Boston, MA, Pearson Education, 2003
[12]	Roger S. Pressman , "Software engineering a practitioner's approach " , McGraw-hill International edition , 2005
[13]	Jeffery L. Whitten, Lonnie D. Bentley, Kevin C. Dittman, "systems analysis and design methods", 5th edition, McGraw-hill Higher Education , 2000

[14]	David avison, Guy fitzgerald, "information systems development methodologies, techniques & tools". 4th edition, McGraw-hill companies , 2006
[15]	Jon Holt , "UML for systems engineering", The institution of Electrical Engineers , 2001
[16]	Jon j. marciniak, Editor-in-chief , "encyclopedia of software engineering", 1994
[17]	Steven R. Rakitin, "software verification and validation A practitioner's guide", library of congress cataloging in publication data, 1997
[18]	Gareth Carter , " Support Tools For Object Oriented Software Development", thesis for the degree of master of science, national university of Ireland, maynooth , 2005
[19]	Muhammad K. Bashar Molla, " an overview object oriented design heuristics" thesis for the degree of Master Thesis , Department of Computer Science, Umeå University, Sweden , 2005
[20]	Raman Ramsin, " The Engineering of an Object-Oriented Software Development Methodology " , thesis for the degree of Doctor of Philosophy Department of Computer Science, University of YORK , 2006

Object oriented software engineering approach to industrial information system design and implementation

Prepared By :

Issa Hessin Manita

Supervised By :

Dr. Idris S. El-Feghi

This thesis submitted in partial fulfillment of the requirements for the degree of master in computer science

Al-Tahadi university

Faculty of science

Department of computer science

Academic year Fall 2008

استعمال هندسة البرمجيات الشيئية لتصميم وتنفيذ نظم المعلومات الصناعية

إعداد الطالب

عيسى حسين أحمد مانيطة

إشراف

د. إدريس ساسي الفقيه

قدمت هذه الرسالة لاستكمال متطلبات الحصول على شهادة
الدرجة العليا الماجستير في علوم الحاسوب

جامعة التحدي - كلية العلوم

قسم الحاسوب

خريف 2008