Unit I: Information System Development (12 Hrs.)

1.1.Fundamentals of System Analysis and Design: System, Information System, System analysis and design and its importance

What is System?

A system is a collection of components (i.e. subsystems) that work together to realize some objective. For example, the **library system** contains librarians, books, and periodicals as components to provide knowledge for its members.

The word System is derived from Greek word Systema, which means an organized relationship between any set of components to achieve some common cause or objective.

A system is "an orderly grouping of interdependent components linked together according to a plan to achieve a specific goal."

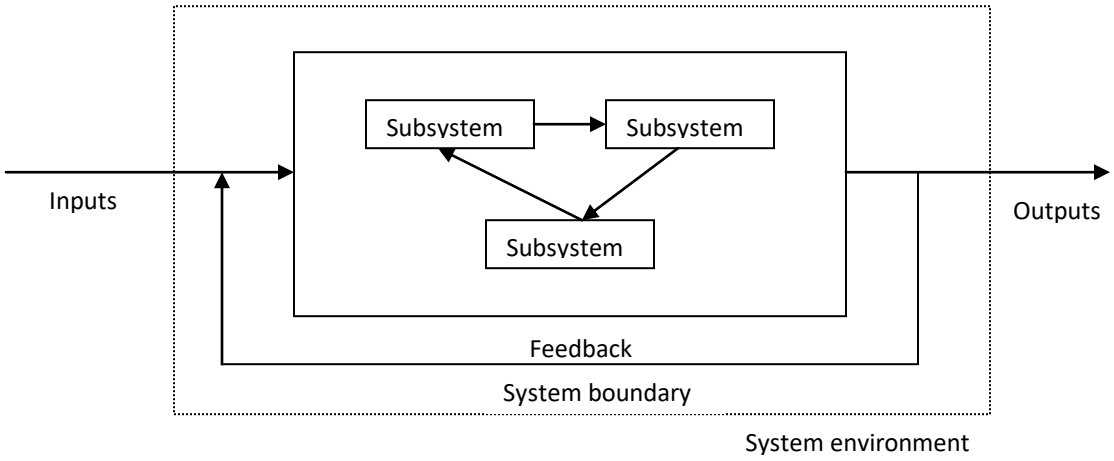


Fig: Basic System Model

Theoretical approaches to systems have introduced many generalized principles. **Goal setting** is one such principle. It defines exactly what the system is supposed to do. There are principles concerned with system structure and behavior. **System boundary** is one such a principle. This defines the components that make up the system and these components can be changed during system design. Anything outside the system boundary is known as **system environment**. A system can be made up of any number of **subsystems**. Each subsystem carries out part of the system function i.e. part of the system goal. The subsystems communicate by passing messages between themselves.

There are also a number of principles concerned with system behavior. One such important principle is **feedback**. It is the idea of monitoring the current system output and

comparing it to the system goal. Any variation from the goal are then fed back in to the system and used to adjust it to ensure that it meets its goal.

The **inputs** to the system are anything to be captured by the system from its environment. **Outputs** are the every thing produced by the system and sent into its environment. The **interface** is the media or channel through which the system and its environment interact.

Constraints of a System

A system must have three basic constraints −

- A system must have some structure and behavior which is designed to achieve a predefined objective.
- Interconnectivity and interdependence must exist among the system components.
- The objectives of the organization have a higher priority than the objectives of its subsystems.

For example, traffic management system, payroll system, automatic library system, human resources information system.

Properties of a System

A system has the following properties −

Organization

Organization implies structure and order. It is the arrangement of components that helps to achieve predetermined objectives.

Interaction

It is defined by the manner in which the components operate with each other.

For example, in an organization, purchasing department must interact with production department and payroll with personnel department.

Interdependence

Interdependence means how the components of a system depend on one another. For proper functioning, the components are coordinated and linked together according to a specified plan. The output of one subsystem is the required by other subsystem as input.

Integration

Integration is concerned with how a system components are connected together. It means that the parts of the system work together within the system even if each part performs a unique function.

Central Objective

The objective of system must be central. It may be real or stated. It is not uncommon for an organization to state an objective and operate to achieve another.

The users must know the main objective of a computer application early in the analysis for a successful design and conversion.

What is an Information System?

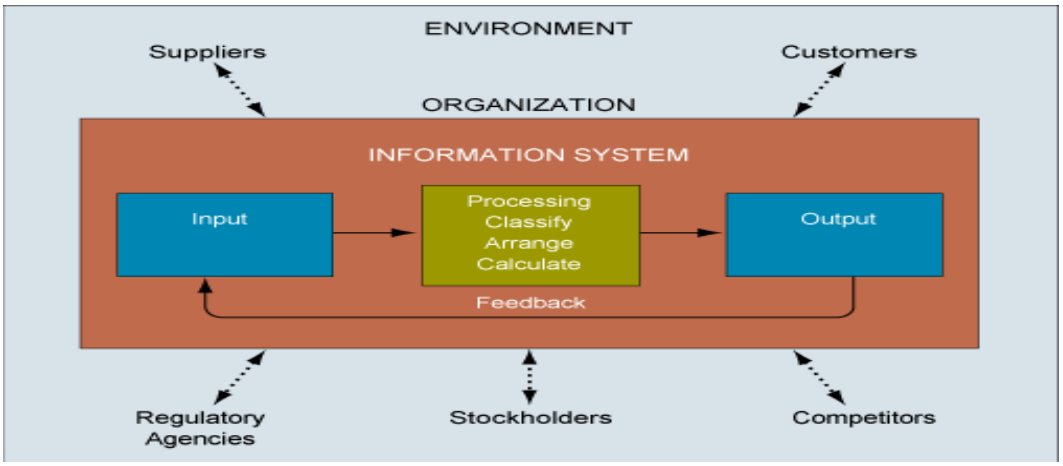
A system that provides information to people in an organization is called **information system (IS)**. An information system is a collection of interrelated components working together to *collect*, *process*, *store*, and *disseminate* information to support *decision making*, *coordination*, *control*, *analysis*, and *visualization* in an organization.

Information systems in organizations capture and manage data to produce useful information that supports an organization and its employees, customers, suppliers and partners. So, many organizations consider information system to be the essential one.

Information systems produce information by using data about significant *people*, *places*, and *things* from within the organization and/or from the external environment to *make decisions*, *control operations*, *analyze problems*, and *create new products or services*. **Information** is the data shaped into a meaningful form. **Data**, on the other hand, are the collection of raw facts representing events occurring in organizations or the environment before they have been organized and arranged into a form that people can understand and use.

The three activities to produce information in an information system are *input*, *processing*, and *output*. **Input** captures or collects row data from within the organization or from its external environment for processing. **Processing** converts these row data into the meaningful information. **Output** transfers this information to the people who will use it or to the activities for which it will be used. Information systems also require **feedback**, which is output that is returned to the appropriate members of the organization to help them evaluate or correct input.

Fig: Activities and Functions of an Information System



The two types of information systems are **formal** and **informal**. **Formal information systems** are based on accepted and fixed definitions of data and procedures for collecting, storing, processing, disseminating, and using these data with predefined rules. **Informal information systems**, in contrast, relay on unstated rules.

Formal information systems can be **manual** as well as **computer based**. **Manual information systems** use paper-and-pencil technology. In contrast, **computer-based information systems (CBIS)** relay on computer hardware and software for processing and disseminating information.

Why Information Systems?

Today, information systems are essential because most organizations cannot survive or prosper without information systems. In addition, information systems can help companies extend their reach to faraway locations, offer new products and services, reshape job and work flows, and perhaps extremely change the way they conduct business.

- **The Competitive Business Environment and the Digital Firm:** There are four trends in the global (or competitive) business environment that have made information systems so important. These trends are described below:
  - **Emergence of the Global Economy**: Companies, both export and import goods and services. They also distribute core business functions to locations in other countries where the work can be performed more cost effectively. The success of firms today and in the future depends on their ability to operate globally.

    Information systems provide the communication and analytic power that firms need for conducting trade and managing business on a global scale.

    Globalization and information technology also bring new threats to domestic business firms. To become competitive participants in international markets, firms need powerful information and communication systems.
  - **Transformation of Industrial Economics**: Most companies are being transformed from industrial economies to knowledge-and information-based service economies, whereas manufacturing has been moving to low wage-countries. In knowledge-and information-based service economy, knowledge and information are key ingredients in creating wealth.

    Knowledge and information intense products such as computer games require a great deal of knowledge and information to produce. In knowledge-and information-based economy, information technology and systems take on great importance. Information systems are needed to optimize the flow of information and knowledge within the organization and to help management to maximize the firm's knowledge resources.

  - **Transformation of the Business Enterprise:** The traditional business firm was – and still is – a hierarchical, centralized, structured arrangement of specialists that typically relied on a fixed set of standard operating procedures to deliver a mass-produced product (or service). Furthermore, the traditional management group also relied – and still relies – on formal plans, a rigid division of labor, and formal rules.

    The new style of business firm is a flattened (less hierarchical), decentralized, a flexible arrangement of generalists. Furthermore, the modern management group relies on informal plans, a flexible arrangement of teams and individuals working in a task force, and informal rules. This type of management is possible with information technology and information systems.
  - **The Emerging Digital Firm:** A digital firm is one where nearly all significant business processes and relationships with customers, suppliers, and employees are digitally enabled, and key corporate assets are managed through digital means. In a digital firm, any piece of information required to support key business decisions is available at anytime and anywhere in the firm. In addition, the digital firms sense and respond to their environment far more rapidly than traditional firms. Digital firms use information technology and information systems to organize and manage.

Key business processes are accomplished through digital networks spanning the entire organization or linking multiple organizations. **Business processes** refer to the unique manner in which work is organized, coordinated, and focused to produce a valuable product or service.

There are four major systems that help to define the digital firm. **Supply chain management systems** automate the relationship between a firm and its suppliers in order to optimize the planning, sourcing, manufacturing, and delivery of products and services. **Customer relationship management systems** automate the relationship between a firm and its customers. **Enterprise systems** are the integrated enterprise-wide information systems that coordinate key internal processes of the firm, integrating data from manufacturing and distribution, sales, finance, and human resources. Finally, **knowledge management systems** create, capture, store, and disseminate firm expertise and knowledge.

System analysis, design, and its importance

Systems development is systematic process, which includes phases such as planning, analysis, design, deployment, and maintenance. Here, in this tutorial, we will primarily focus on −

- Systems analysis
- Systems design

a. System analysis

It is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components.

System analysis is conducted for studying a system or its parts in order to identify its objectives. It is a problem solving technique that improves the system and ensures that all the components of the system work efficiently to accomplish their purpose.

Analysis specifies what the system should do.

b. Systems Design

It is a process of planning a new business system or replacing an existing system by defining its components or modules to satisfy the specific requirements. Before planning, you need to understand the old system thoroughly and determine how computers can best be used in order to operate efficiently. System Design focuses on how to accomplish the objective of the system.

System Analysis and Design (SAD) mainly focuses on −

- Systems
- Processes
- Technology design

Importance of system analysis and design

System designing in terms of software engineering has its own value and importance in the system development process as a whole. To mention it may though seem as simple as anything or simply the design of systems, but in a broader sense it implies a systematic and rigorous approach to design such a system which fulfills all the practical aspects including flexibility, efficiency and security. Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development.

Before there is any further discussion of system design, it is important that some points be made clear. As it goes without saying that nothing is created that is not affected by the world in which it's made. So, the systems are not created in a vacuum.

They are created in order to meet the needs of the users. They are not only intended to solve the existing problems, but they also come up with acceptable solutions to the problems that may arise in the future. The whole process of system development, from blueprint to the actual product, involves considering all the relevant factors and taking the required specifications and creating a useful system based on strong technical, analytical and development skills of the professionals.

Let's get back to our discussion about what the system design phase is and the importance of system design in the process of system development. Being another important step in the system development process, system designing phase commences after the system analysis phase is completed. It's appropriate to mention that the output or the specifications taken through the phase of system analysis become an input in the system design phase which in turn leads to workout based on the user defined estimations.

The importance of this phase may be understood by reason of the fact that it involves identifying data sources, the nature and type of data that is available. For example, in order to design a salary system, there is a need for using inputs, such as, attendance, leave details, additions or deductions etc. This facilitates understanding what kind of data is available and by whom it is supplied to the system so that the system may be designed considering all the relevant factors. In addition, system designing leads to ensure that the system is created in such a way that it fulfills the need of the users and keep them at ease being user-oriented. In terms of the flexibility, one of the main objectives of this phase is that it is intended to design such a system which can be dynamic in nature and responsive to the changes if required. Another important objective is that the phase of system designing is concerned with creating the system which can work efficiently providing the required output and being responsive to the time within a given time limit. The aspect of reliability and physical security of data cannot be ignored. With this respect, the system designing phase ensures security measures of the system effectively and efficiently.

1.2.Process of System Development, Capability Maturity Model (CMM) Level

Introduction

Work is getting underway at SoundStage Entertainment Club for the systems analysis and design of their member services information system. But the more Bob Martinez learns about the system, the more confused he gets. Bob can recall some of his programming assignments in college. Most of them were just a page or two of bulleted points listing required features. It was pretty easy to get your head around that. But the new SoundStage system will involve tracking member contacts and purchase requirements, promotions, sales, shipments, inventory, multiple warehouses, Web sites, and more. Bob wonders how they will even list all the requirements, let alone keep them straight. How will they know what data they need to track? How will they know what every piece of programming needs to do? He mentioned that to his boss, Sandra. She said it was all about following "the methodology." He remembered something about methodology from a systems analysis class. At the time it seemed like a lot of unnecessary work. But he is starting to see now that on a large project, following an established method may be the only path that is safe to travel.

The Process of Systems Development

systems development process a set of activities, methods, best practices, deliverables, and automated tools that stakeholders (from Chapter 1) use to develop and continuously improve information systems and software (from Chapters 1 and 2).

This chapter introduces a focus on information systems development. We will examine a **systems development process.** Notice we did not say "the" process—there are as many variations on the process as there are experts and authors. We will present one representative process and use it consistently throughout this book. The chapter home page shows an expanded number of phases compared to the home page of Chapter 1. This is because we have factored the high-level phases such as system analysis and system design from Chapter 1 into multiple phases and activities. We have also refined the size and place of the stakeholder roles to reflect "involvement" as opposed to emphasis or priority. And we have edited and enhanced the building blocks to indicate deliverables and artifacts of system development. All of these modifications will be explained in due time.

Why do organizations embrace standardized processes to develop information systems? Information systems are a complex product. Recall from Chapter 2 that an information system includes data, process, and communications building blocks and technologies that must serve the needs of a variety of stakeholders. Perhaps this explains why as many as 70 percent or more of information system development projects have failed to meet expectations, cost more than budgeted, and are delivered much later than promised. The Gartner Group suggests that "consistent adherence to moderately rigorous methodology guidelines can provide 70 percent of [systems development] organizations with a productivity improvement of at least 30 percent within two years."[1]

Increasingly, organizations have no choice but to adopt and follow a standardized systems development process. First, using a consistent process for systems development creates efficiencies that allow management to shift resources between projects. Second, a consistent methodology produces consistent documentation that reduces lifetime costs to maintain the systems. Finally, the U.S. government has mandated that any organization seeking to develop software for the government must adhere to certain quality management requirements. A consistent process promotes quality. And many other organizations have aggressively committed to total quality management goals in order to increase their competitive advantage. In order to realize quality and productivity improvements, many organizations have turned to project and process management frameworks such as the Capability Maturity Model, discussed in the next section.

[1]Richard Hunter, "AD Project Portfolio Management," *Proceedings of the Gartner Group ITS98 Symposium/Expo* (CD-ROM). The Gartner Group is an industry watchdog and research group that tracks and projects industry trends for IT managers.

> The Capability Maturity Model

It has been shown that as an organization's information system development process matures, project timelines and cost decrease while productivity and quality increase. The Software Engineering Institute at Carnegie Mellon University has observed and measured this phenomenon and developed the **Capability Maturity Model (CMM)** to assist all organizations to achieve these benefits. The CMM has developed a wide following, both in industry and government. Software evaluation based on CMM is being used to qualify information technology contractors for most U.S. federal government projects.

Capability Maturity Model (CMM) a standardized framework for assessing the maturity level of an organization's information systems development and management processes and products. It consists of five levels of maturity.

The CMM framework for systems and software is intended to help organizations improve the maturity of their systems development processes. The CMM is organized into five maturity levels (see Figure 3-1):

- *Level 1—Initial:* This is sometimes called anarchy or chaos. At this level, system development projects follow no consistent process. Each development team uses its own tools and methods. Success or failure is usually a function of the skill and experience of the team. The process is unpredictable and not repeatable. A project typically encounters many crises and is frequently over budget and behind schedule. Documentation is sporadic or not consistent from one project to the next, thus creating problems for those who must maintain a system over its lifetime. Almost all organizations start at Level 1.
- *Level 2—Repeatable:* At this level, project management processes and practices are established to track project costs, schedules, and functionality. The focus is on project management. A system development process is always followed, but it may vary from project to project. Success or failure is still a function of the skill and experience of the project team; however, a concerted effort is made to repeat earlier project successes. Effective project management practices lay the foundation for standardized processes in the next level.

FIGURE 3-1
The Capability Maturity Model (CMM)

TABLE 3-1 Impact of System Development "Process" on Quality

CMM Project Statistics for a Project Resulting in 200,000 Lines of Code

Organization's CMM Level	Project Duration (months)	Project Person-Months	Number of Defects Shipped	Median Cost ($ millions)	Lowest Cost ($ millions)	Highest Cost ($ millions)
1	30	600	61	5.5	1.8	100+
2	18.5	143	12	1.3	.96	1.7
3	15	80	7	.728	.518	.933

Source: Master Systems, Inc.

system development methodology a formalized approach to the systems development process; a standardized process that includes the activities, methods, best practices, deliverables, and automated tools to be used for information systems development.

- *Level 3—Defined:* In this level, a standard **system development** process (sometimes called a *methodology*) is purchased or developed. All projects use a tailored version of this process to develop and maintain information systems and software. As a result of using the standardized process for all projects, each project results in consistent and high-quality documentation and deliverables. The process is stable, predictable, and repeatable.
- *Level 4—Managed:* In this level, measurable goals for quality and productivity are established. Detailed measures of the standard system development process and product quality are routinely collected and stored in a database. There is an effort to improve individual project management based on this collected data. Thus, management seeks to become more proactive than reactive to systems development problems (such as cost overruns, scope creep, schedule delays, etc.). Even when a project encounters unexpected problems or issues, the process can be adjusted based on predictable and measurable impacts.
- *Level 5—Optimizing:* In this level, the standardized system development process is continuously monitored and improved based on measures and data analysis established in Level 4. This can include changing the technology and best practices used to perform activities required in the standard system development process, as well as adjusting the process itself. Lessons learned are shared across the organization, with a special emphasis on eliminating inefficiencies in the system development process while sustaining quality. In summary, the organization has institutionalized continuous systems development process improvement.

It is very important to recognize that each level is a prerequisite for the next level.

Currently, many organizations are working hard to achieve at least CMM Level 3 (sometimes driven by a government or organizational mandate). A central theme to achieving Level 3 (Defined) is the use of a standard process or methodology to build or integrate systems. As shown in Table 3-1, an organization can realize significant improvements in schedule and cost by institutionalizing CMM Level 3 process improvements.[2]

system life cycle the factoring of the lifetime of an information system into two stages, (1) systems development and (2) systems operation and maintenance—first you build it; then you use and maintain it. Eventually, you cycle back to redevelopment of a new system.

> Life Cycle versus Methodology

The terms *system life cycle* and *system development methodology* are frequently and incorrectly interchanged. Most system development processes are derived from a natural **system life cycle.** The system life cycle just happens. Figure 3-2 illustrates two

[2] White Paper, "Rapidly Improving Process Maturity: Moving Up the Capability Maturity Model through Outsourcing" (Boston: Keane, 1997, 1998, p.11).

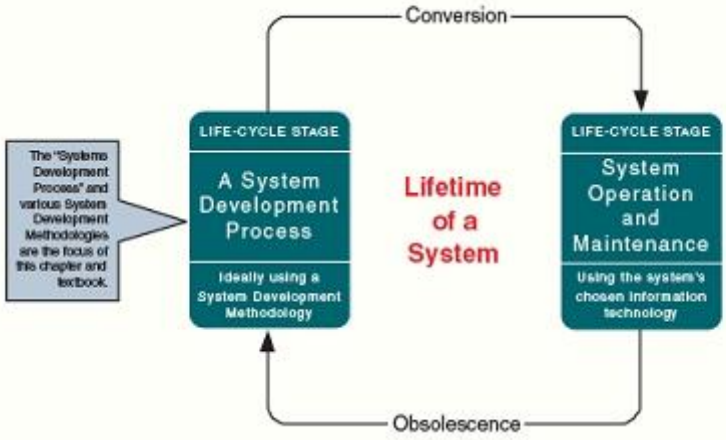


FIGURE 3-2
The System Life Cycle

life-cycle stages. Notice that there are two key events that trigger a change from one stage to the other:

- When a system cycles from development to operation and maintenance, a *conversion* must take place.
- At some point in time, *obsolescence* occurs (or is imminent) and a system cycles from operation and maintenance to redevelopment.

Actually, a system may be in more than one stage at the same time. For example, one version may be in *operation and support* while the next version is in *development.*

So how does this contrast with a systems development methodology? A systems development methodology "executes" the systems development stage of the system life cycle. Each individual information system has its own system life cycle. The methodology is the standard process to build and maintain that system and all other information systems through their life cycles. Consistent with the goals of the CMM, methodologies ensure that:

- A consistent, reproducible approach is applied to all projects.
- There is reduced risk associated with shortcuts and mistakes.
- Complete and consistent documentation is produced from one project to the next.
- Systems analysts, designers, and builders can be quickly reassigned between projects because all use the same process.
- As development teams and staff constantly change, the results of prior work can be easily retrieved and understood by those who follow.

Methodologies can be purchased or homegrown. Why purchase a methodology? Many information system organizations can't afford to dedicate staff to the development and continuous improvement of a homegrown methodology. Methodology vendors have a vested interest in keeping their methodologies current with the latest business and technology trends. Homegrown methodologies are usually based on generic methodologies and techniques that are well documented in books and on Web sites. Examples of system development methodologies are listed in the margin on the following page. You should be able to research most of them on the Web. Many of their underlying methods will be taught in this textbook.

Throughout this book, we will use a methodology called ***FAST,*** which stands for *F*ramework for the *A*pplication of *S*ystems *T*hinking. *FAST* is not a real-world commercial methodology. We developed it as a composite of the best practices we've

FAST a hypothetical methodology used throughout this book to demonstrate a representative systems development process. The acronym's letters stand for *F*ramework for the *A*pplication of *S*ystems *T*hinking.

REPRESENTATIVE SYSTEM DEVELOPMENT METHODOLOGIES

Architected Rapid Application Development (Architected RAD)
Dynamic Systems Development Methodology (DSDM)
Joint Application Development (JAD)
Information Engineering (IE)
Rapid Application Development (RAD)
Rational Unified Process (RUP)
Structured Analysis and Design (old, but still occasionally encountered)
eXtreme Programming (XP)
Note: There are many commercial methodologies and software tools (sometimes called *methodware*) based on the above general methodologies.

encountered in many commercial and reference methodologies. Unlike many commercial methodologies, *FAST* is not prescriptive. That is to say, *FAST* is an agile framework that is flexible enough to provide for different types of projects and strategies. Most important, *FAST* shares much in common with both the book-based and the commercial methodologies that you will encounter in practice.

> Underlying Principles for Systems Development

Before we examine the *FAST* methodology, let's introduce some general principles that should underlie all systems development methodologies.

Principle 1: Get the System Users Involved Analysts, programmers, and other information technology specialists frequently refer to "my system." This attitude has, in part, created an "us versus them" conflict between technical staff and their users and management. Although analysts and programmers work hard to create technologically impressive solutions, those solutions often backfire because they don't address the real organization problems. Sometimes they even introduce new organization problems. For this reason, system user involvement is an absolute necessity for successful systems development. Think of systems development as a partnership between system users, analysts, designers, and builders. The analysts, designers, and builders are responsible for systems development, but they must engage their owners and users, insist on their participation, and seek agreement from all stakeholders concerning decisions that may affect them.

Miscommunication and misunderstandings continue to be a significant problem in many systems development projects. However, owner and user involvement and education minimize such problems and help to win acceptance of new ideas and technological change. Because people tend to resist change, information technology is often viewed as a threat. The best way to counter that threat is through constant and thorough communication with owners and users.

Principle 2: Use a Problem-Solving Approach A system development methodology is, first and foremost, a problem-solving approach to building systems. The term *problem* is broadly used throughout this book to include (1) real problems, (2) opportunities for improvement, and (3) directives from management. The classical problem-solving approach is as follows:

1. Study and understand the problem, its context, and its impact.
2. Define the requirements that must be met by *any* solution.
3. Identify candidate solutions that fulfill the requirements, and select the best solution.
4. Design and/or implement the chosen solution.
5. Observe and evaluate the solution's impact, and refine the solution accordingly.

Systems analysts should approach all projects using some variation of this problem-solving approach.

Inexperienced or unsuccessful problem solvers tend to eliminate or abbreviate one or more of the above steps. For example, they fail to completely understand the problem, or they prematurely commit to the first solution they think of. The result can range from (1) solving the wrong problem, to (2) incorrectly solving the problem, (3) picking the wrong solution, or (4) picking a less-than-optimal solution. A methodology's problem-solving process, when correctly applied, can reduce or eliminate these risks.

Principle 3: Establish Phases and Activities All methodologies prescribe phases and activities. The number and scope of phases and activities vary from author to author, expert to expert, methodology to methodology, and business to business. The chapter home page at the beginning of this chapter illustrates the eight phases of our *FAST* methodology in the context of your information systems framework. The phases

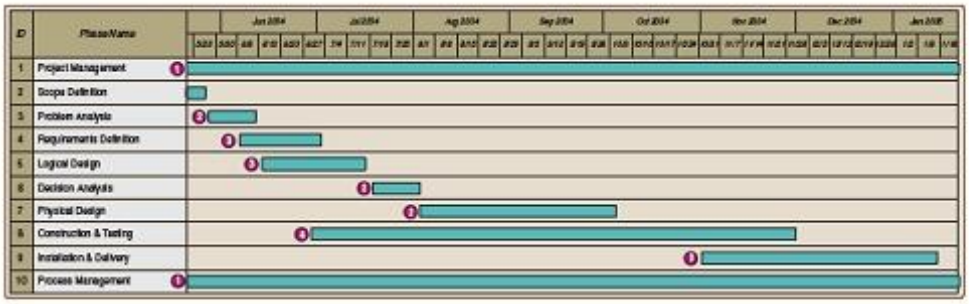


FIGURE 3-3 Overlap of System Development Phases and Activities

are listed on the far right-hand side of the illustration. In each phase, the focus is on those building blocks and on stakeholders that are aligned to the left of that phase.

The phases are: scope definition, problem analysis, requirements analysis, logical design, decision analysis, physical design and integration, construction and testing, and installation and delivery. Each of these phases will be discussed later in this chapter. These phases are not absolutely sequential. The phases tend to overlap one another, as illustrated in Figure 3-3. Also, the phases may be customized to the special needs of a given project (e.g., deadlines, complexity, strategy, resources). In this chapter, we will describe each customization as alternative routes through the methodology and problem-solving process.

Principle 4: Document throughout Development When do you document the programs you write? Be honest. We must confess that, like most students, we did our documentation after we wrote the programs. We knew better, but we postdocumented anyway. That just does not work in the business world. In medium to large organizations, system owners, users, analysts, designers, and builders come and go. Some will be promoted; some will have extended medical leaves; some will quit the organization; and still others will be reassigned. To promote good communication between constantly changing stakeholders, documentation should be a working byproduct of the entire systems development effort.

Documentation enhances communications and acceptance. Documentation reveals strengths and weaknesses of the system to multiple stakeholders. It stimulates user involvement and reassures management about progress. At the same time, some methodologies have been criticized for expecting too much documentation that adds little value to the process or resulting system. Our *FAST* methodology advocates a balance between the value of documentation and the effort to produce it. Experts call this *agile modeling*.

Principle 5: Establish Standards In a perfect world, all information systems would be integrated such that they behave as a single system. Unfortunately, this never happens because information systems are developed and replaced over a very long period of time. Even organizations that purchase and install an enterprise resource planning (ERP) product usually discover that there are applications and needs that fall outside the ERP system. Systems integration has become critical to the success of any organization's information systems.

To achieve or improve systems integration, organizations turn to standards. In many organizations, these standards take the form of enterprise information technology architecture. An IT architecture sets standards that serve to direct technology solutions and information systems to a common technology vision or configuration.

An information technology architecture typically standardizes on the following (note: it is not important that you know what all these sample technologies are):

- *Database technology*—What database engine(s) will be used (e.g., *Oracle*, IBM *DB2*, Microsoft *SQL Server*)? On what platforms will they be operated (e.g., *UNIX, Linux, Windows XP, MVS*)? What technologies will be used to load data into online transaction processing (OLTP) databases, operational data stores, and data warehouses (i.e., Extract Transform and Load [ETL])?
- *Software technology*—What application development environment(s)/language(s) will be used to write software (e.g., IBM's *Websphere* with *Java*, Microsoft's *Visual Studio .NET* with *Visual Basic .NET, Visual C++*, and/or *Visual C#*; Syebase's *Powerbuilder*, Oracle's *Oracle Forms*)?
- *Interface technology*—How will user interfaces be developed—with *MS Windows* components or Web languages and components (e.g., an *xHTML* editor such as Macromedia's *Dreamweaver*, a portal engine such as IBM's *Websphere*)? How will data be exchanged between different information systems (e.g., a data broker such as IBM's *MQ Messaging*, an XML-based data exchange, or a custom programmed interface)?

Notice how these architectural questions closely correspond to the technology drivers in your information system model.

In the absence of an IT architecture, each information system and application may be built using radically different technologies. Not only does this make it difficult to integrate applications, but it creates resource management problems—IT organizations cannot as easily move developers between projects as priorities change or emergencies occur because different teams are staffed with technical competencies based on the various technologies used and being used to develop information systems. Creating an enterprise IT architecture and driving projects and teams to that architecture make more sense.

As new technologies emerge, an IT architecture must change. But that change can be managed. The chief technology officer (CTO) in an organization is frequently charged with technology exploration and IT architecture management. Given that architecture, all information systems projects are constrained to implement new systems that conform to the architecture (unless otherwise approved by the CTO).

Principle 6: Manage the Process and Projects Most organizations have a system development process or methodology, but they do not always use it consistently on projects. Both the process and the projects that use it must be managed. **Process management** ensures that an organization's chosen process or management is used consistently on and across all projects. Process management also defines and improves the chosen process or methodology over time. **Project management** ensures that an information system is developed at minimum cost, within a specified time frame, and with acceptable quality (using the standard system development process or methodology). Effective project management is essential to achieving CMM Level 2 success. Use of a repeatable process gets us to CMM Level 3. CMM Levels 4 and 5 require effective process management. Project management can occur without a standard process, but in mature organizations all projects are based on a standardized and managed process.

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

project management the process of scoping, planning, staffing, organizing, directing, and controlling a project to develop an information system at minimum cost, within a specified time frame, and with acceptable quality.

Process management and project management are influenced by the need for quality management. Quality standards are built into a process to ensure that the activities and deliverables of each phase will contribute to the development of a high-quality information system. They reduce the likelihood of missed problems and requirements, as well as flawed designs and program errors (bugs). Standards also make the IT organization more agile. As personnel changes occur, staff can be relocated between projects with the assurance that every project is following an understood and accepted process.

Principle 7: Justify Information Systems as Capital Investments Information systems are capital investments, just like a fleet of trucks or a new building. System owners commit to this investment. Notice that the initial commitment occurs early in a project, when system owners agree to sponsor and fund the project. Later (during the phase called *decision analysis*), system owners recommit to the more costly technical decisions. In considering a capital investment, two issues must be addressed:

1. For any problem, there are likely to be several possible solutions. The systems analyst and other stakeholders should not blindly accept the first solution suggested. The analyst who fails to look at alternatives may be doing the business a disservice.
2. After identifying alternative solutions, the systems analyst should evaluate each possible solution for feasibility, especially for **cost-effectiveness.** Cost-effectiveness is measured using a technique called *cost-benefit analysis.*

cost-effectiveness the result obtained by striking a balance between the lifetime costs of developing, maintaining, and operating an information system and the benefits derived from that system. Cost-effectiveness is measured by cost-benefit analysis.

Like project and process management, cost-benefit analysis is performed throughout the system development process.

A significant advantage of the phased approach to systems development is that it provides several opportunities to reevaluate cost-effectiveness, risk, and feasibility. There is often a temptation to continue with a project only because of the investment already made. In the long run, canceled projects are usually much less costly than implemented disasters. This is extremely important for young analysts to remember.

Most system owners want more from their systems than they can afford or are willing to pay for. Furthermore, the scope of most information system projects increases as the analyst learns more about the business problems and requirements as the project progresses. Unfortunately, most analysts fail to adjust estimated costs and schedules as the scope increases. As a result, the analyst frequently and needlessly accepts responsibility for cost and schedule overruns.

Because information systems are recognized as capital investments, system development projects are often driven by enterprise planning. Many contemporary information technology business units create and maintain a **strategic information systems plan.** Such a plan identifies and prioritizes information system development projects. Ideally, a strategic information systems plan is driven by a **strategic enterprise plan** that charts a course for the entire business.

strategic information systems plan a formal strategic plan (3 to 5 years) for building and improving an information technology infrastructure and the information system applications that use that infrastructure.

strategic enterprise plan a formal strategic plan (3 to 5 years) for an entire business that defines its mission, vision, goals, strategies, benchmarks, and measures of progress and achievement. Usually, the strategic enterprise plan is complemented by strategic business unit plans that define how each business unit will contribute to the enterprise plan. The information systems plan (above) is one of those unit-level plans.

Principle 8: Don't Be Afraid to Cancel or Revise Scope There is an old saying: "Don't throw good money after bad." In other words, don't be afraid to cancel a project or revise scope, regardless of how much money has been spent so far—cut your losses. To this end, we advocate a **creeping commitment** approach to systems development.[3] With the creeping commitment approach, multiple feasibility checkpoints are built into any systems development methodology. At each checkpoint feasibility is reassessed. All previously expended costs are considered sunk (meaning not recoverable). They are, therefore, irrelevant to the decision. Thus, the project should be reevaluated at each checkpoint to determine if it remains feasible to continue investing time, effort, and resources into the project. At each checkpoint, the analyst should consider the following options:

- Cancel the project if it is no longer feasible.
- Reevaluate and adjust the costs and schedule if project scope is to be increased.
- Reduce the scope if the project budget and schedule are frozen and not sufficient to cover all project objectives.

creeping commitment a strategy in which feasibility and risks are continuously reevaluated throughout a project. Project budgets and deadlines are adjusted accordingly.

The concept of sunk costs is more or less familiar to most financial analysts, but it is frequently forgotten or not used by the majority of systems analysts, most system users, and even many system owners.

[3] Thomas Gildersleeve, *Successful Data Processing Systems Analysis*, 2nd ed. (Englewood Cliffs, NJ: Prentice Hall, 1985), pp. 5–7.

risk management the process of identifying, evaluating, and controlling what might go wrong in a project before it becomes a threat to the successful completion of the project or implementation of the information system. Risk management is driven by risk analysis or assessment.

In addition to managing feasibility throughout the project, we must manage risk. **Risk management** seeks to balance risk and reward. Different organizations are more or less averse to risk, meaning that some are willing to take greater risks than others in order to achieve greater rewards.

Principle 9: Divide and Conquer Whether you realize it or not, you learned the divide-and-conquer approach throughout your education. Since high school, you've been taught to outline a paper before you write it. Outlining is a divide-and-conquer approach to writing. A similar approach is used in systems development. We divide a system into subsystems and components in order to more easily conquer the problem and build the larger system. In systems analysis, we often call this *factoring*. By repeatedly dividing a larger problem (system) into more easily managed pieces (subsystems), the analyst can simplify the problem-solving process. This divide-and-conquer approach also complements communication and project management by allowing different pieces of the system to be communicated to different and the most appropriate stakeholders.

The building blocks of your information system framework provide one basis for dividing and conquering an information system's development. We will use this framework throughout the book.

Principle 10: Design Systems for Growth and Change Businesses change over time. Their needs change. Their priorities change. Accordingly, information systems that support a business must change over time. For this reason, good methodologies should embrace the reality of change. Systems should be designed to accommodate both growth and changing requirements. In other words, well-designed information systems can both scale up and adapt to the business. But regardless of how well we design systems for growth and change, there will always come a time when they simply cannot support the business.

System scientists describe the natural and inevitable decay of all systems over time as *entropy*. As described earlier in this section, after a system is implemented it enters the *operations and maintenance* stage of the life cycle. During this stage the analyst encounters the need for changes that range from correcting simple mistakes, to redesigning the system to accommodate changing technology, to making modifications to support changing user requirements. Such changes direct the analyst and programmers to rework formerly completed phases of the life cycle. Eventually, the cost of maintaining the current system exceeds the costs of developing a replacement system—the current system has reached entropy and becomes obsolete.

But system entropy can be managed. Today's tools and techniques make it possible to design systems that can grow and change as requirements grow and change. This book will teach you many of those tools and techniques. For now, it's more important to simply recognize that flexibility and adaptability do not happen by accident—they must be built into a system.

We have presented 10 principles that should underlie any methodology. These principles are summarized in the margin and can be used to evaluate any methodology, including our *FAST* methodology.

PRINCIPLES OF SYSTEMS DEVELOPMENT

- Get the System Users Involved.
- Use a Problem-Solving Approach.
- Establish Phases and Activities.
- Document throughout Development.
- Establish Standards.
- Manage the Process and Projects.
- Justify Information Systems as Capital Investments.
- Don't Be Afraid to Cancel or Revise Scope.
- Divide and Conquer.
- Design Systems for Growth and Change.

A Systems Development Process

In this section we'll examine a logical process for systems development. (Reminder: *FAST* is a hypothetical methodology created for learning purposes.) We'll begin by studying types of system projects and how they get started. Then we'll introduce the eight *FAST* phases. Finally, we'll examine alternative variations, or "routes" through the phases, for different types of projects and development strategies.

1.3. System Life Cycle Vs. Development, Underlying Principles for System Development, System Development Lifecycle (SDLC): Planning and Selection, Analysis, Design, Implementation and Operation, Cross Life Cycle Activities

1.4. Alternate Approaches to Development: Rapid Application Development

Rapid application development (RAD) describes a method of software development which heavily emphasizes rapid prototyping and iterative delivery. The RAD model is, therefore, a sharp alternative to the typical waterfall development model, which often focuses largely on planning and sequential design practices. First introduced in 1991 in James Martin's book by the same name, rapid application development has become one of the most popular and powerful development methods, which falls under the parental category of agile development techniques.

RAD model is Rapid Application Development model. It is a type of incremental model. In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype. This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.

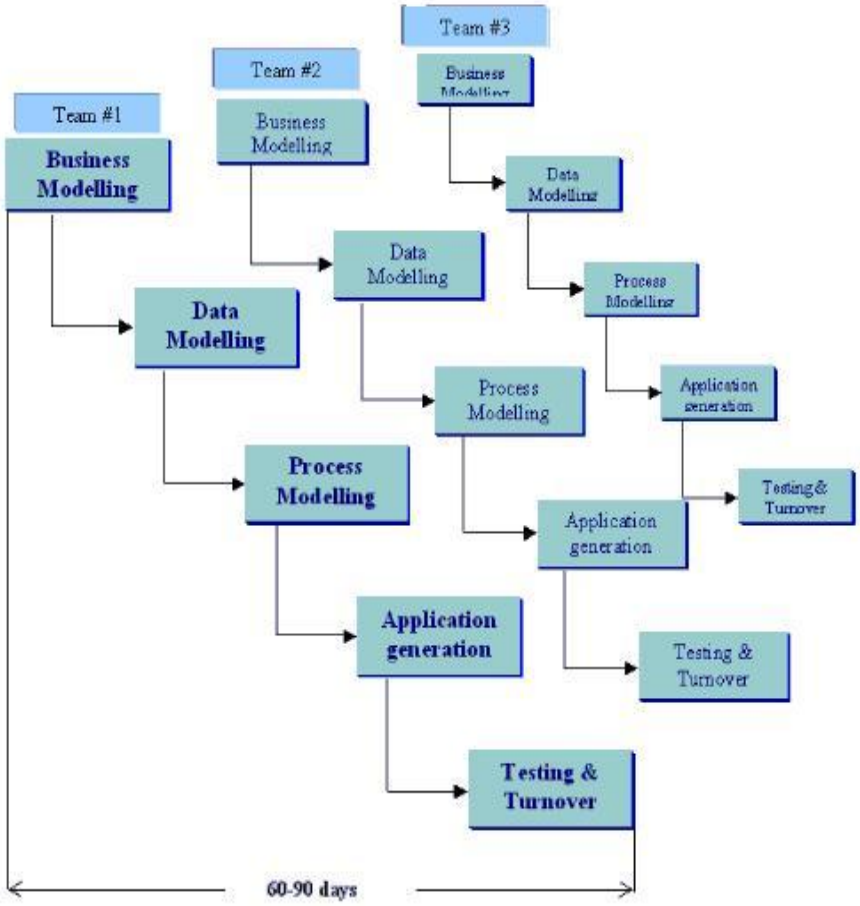


Figure 1.5 – RAD Model

The phases in the rapid application development (RAD) model are:

Business modeling: The information flow is identified between various business functions.
Data modeling: Information gathered from business modeling is used to define data objects that are needed for the business.
Process modeling: Data objects defined in data modeling are converted to achieve the business information flow to achieve some specific business objective. Description are identified and created for CRUD of data objects.
Application generation: Automated tools are used to convert process models into code and the actual system.
Testing and turnover: Test new components and all the interfaces.

Advantages of the RAD model:

- Reduced development time.
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of **integration issues**.

Disadvantages of RAD model:

- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularized can be built using RAD
- Requires highly skilled developers/designers.
- High dependency on modeling skills
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

When to use RAD model:

- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD **SDLC model** should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

For more detail please visit : https://airbrake.io/blog/sdlc/rapid-application-development

Agile Methodology

Agile software development methodology is an process for developing software (like other software development methodologies – Waterfall model, V-Model, Iterative model etc.) However, Agile methodology differs significantly from other methodologies. In English, Agile means 'ability to move quickly and easily' and responding swiftly to change – this is a key aspect of Agile software development as well.

Brief overview of Agile Methodology

In traditional software development methodologies like Waterfall model, a project can take several months or years to complete and the customer may not get to see the end product until the completion of the project.

At a high level, non-Agile projects allocate extensive periods of time for Requirements gathering, design, development, testing and User Acceptance Testing, before finally deploying the project.
In contrast to this, Agile projects have Sprints or iterations which are shorter in duration (Sprints/iterations can vary from 2 weeks to 2 months) during which pre-determined features are developed and delivered.

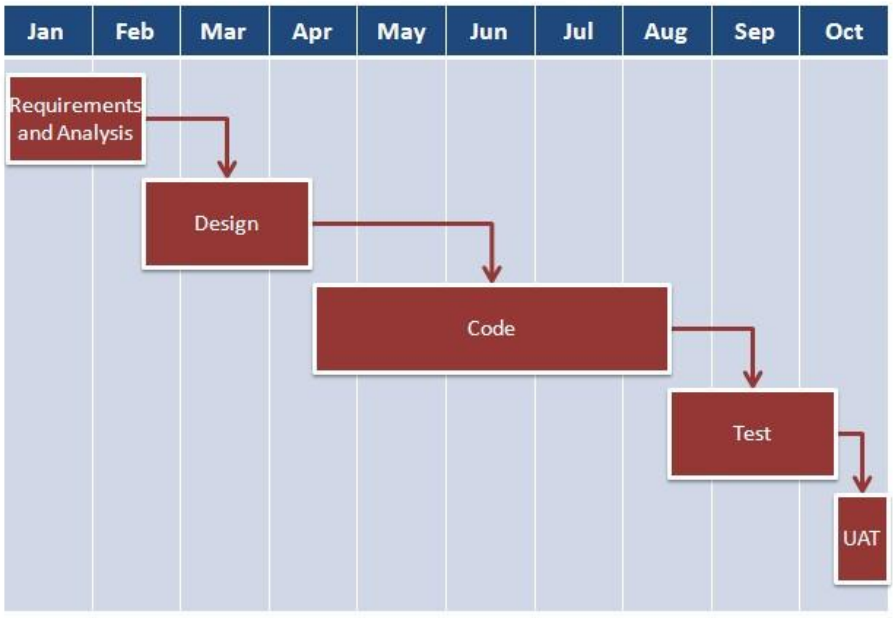
Agile projects can have one or more iterations and deliver the complete product at the end of the final iteration.

Example of Agile software development

Example: Google is working on project to come up with a competing product for MS Word, that provides all the features provided by MS Word and any other features requested by the marketing team. The final product needs to be ready in 10 months of time. Let us see how this project is executed in traditional and Agile methodologies.

In traditional Waterfall model –

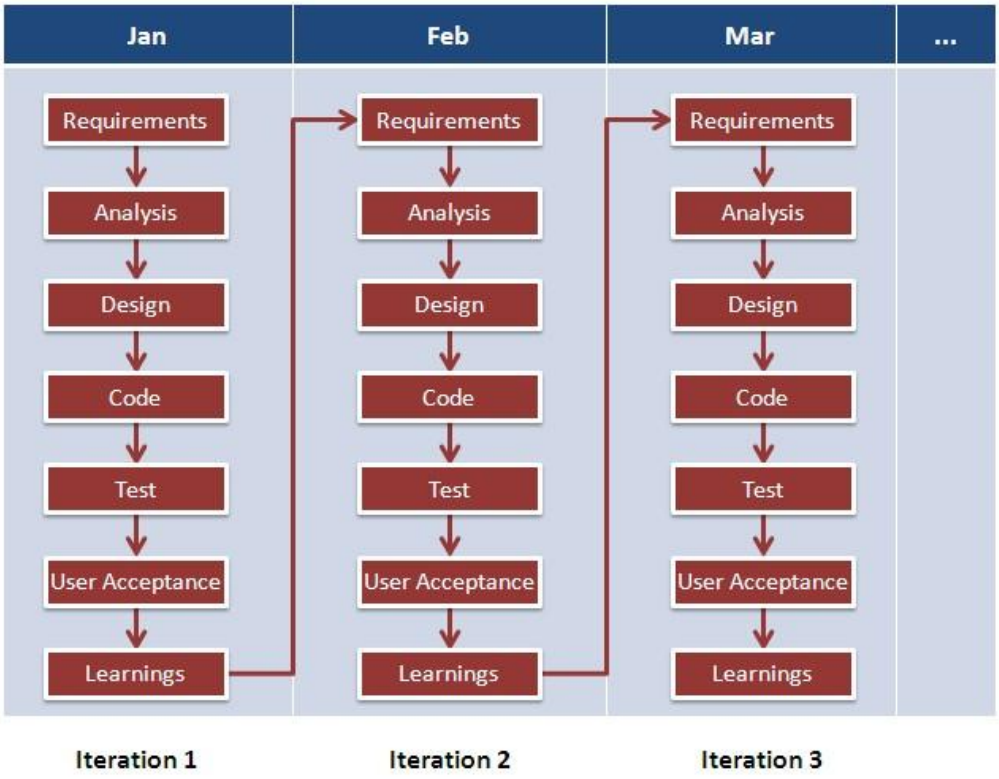
At a high level, the project teams would spend 15% of their time on gathering requirements and analysis (1.5 months)
20% of their time on design (2 months)
40% on coding (4 months) and unit testing
20% on System and Integration testing (2 months).
At the end of this cycle, the project may also have 2 weeks of User Acceptance testing by marketing teams.
In this approach, the customer does not get to see the end product until the end of the project, when it becomes too late to make significant changes.
The image below shows how these activities align with the project schedule in traditional software development.



With Agile development methodology –

- In the Agile methodology, each project is broken up into several 'Iterations'.
- All Iterations should be of the same time duration (between 2 to 8 weeks).
- At the end of each iteration, a working product should be delivered.
- In simple terms, in the Agile approach the project will be broken up into 10 releases (assuming each iteration is set to last 4 weeks).
- Rather than spending 1.5 months on requirements gathering, in Agile software development, the team will decide the basic core features that are required in the product and decide which of these features can be developed in the first iteration.
- Any remaining features that cannot be delivered in the first iteration will be taken up in the next iteration or subsequent iterations, based on priority.
- At the end of the first iterations, the team will deliver a working software with the features that were finalized for that iteration.
- There will be 10 iterations and at the end of each iteration the customer is delivered a working software that is incrementally enhanced and updated with the features that were shortlisted for that iteration.

The iteration cycle of an Agile project is shown in the image below.


Jan
Feb
Mar
...
Requirements
Analysis
Design
Code
Test
User Acceptance
Learnings
Requirements
Analysis
Design
Code
Test
User Acceptance
Learnings
Requirements
Analysis
Design
Code
Test
User Acceptance
Learnings
Iteration 1
Iteration 2
Iteration 3

This approach allows the customer to interact and work with functioning software at the end of each iteration and provide feedback on it. This approach allows teams to take up changes more easily and make course corrections if needed. In the Agile approach, software is developed and released incrementally in the iterations. An example of how software may evolve through iterations is shown in the image below.

Agile methodology gives more importance to collaboration within the team, collaboration with the customer, responding to change and delivering working software.

Agile development has become common place in IT industry. In a recent survey over 52% of respondents said that their company practiced Agile development in one form or another. Irrespective of your role in the organization, it has become essential to understand how Agile development works and how it differs from other forms of software development.

In traditional approach each job function does its job and hands over to the next job function. The previous job functions have to signoff before it is handed over the next job function authenticating that the job is full and complete in all aspects. For example, Requirement gathering is completed and handed over to design phase and it is subsequently handed over to development and later to testing and rework. Each job function is a phase by itself.

In Agile way of working, each feature is completed in terms of design, development, code, testing and rework, before the feature is called done. There are no separate phases and all the work is done in single phase only.

Advantages of Agile Methodology

- In Agile methodology the delivery of software is unremitting.
- The customers are satisfied because after every Sprint working feature of the software is delivered to them.
- Customers can have a look of the working feature which fulfilled their expectations.
- If the customers has any feedback or any change in the feature then it can be accommodated in the current release of the product.
- In Agile methodology the daily interactions are required between the business people and the developers.
- In this methodology attention is paid to the good design of the product.
- Changes in the requirements are accepted even in the later stages of the development.

Disadvantages of the Agile Methodology

- In Agile methodology the documentation is less.
- Sometimes in Agile methodology the requirement is not very clear hence it's difficult to predict the expected result.
- In few of the projects at the starting of the software development life cycle it's difficult to estimate the actual effort required.
- The projects following the Agile methodology may have to face some unknown risks which can affect the development of the project.

Commercial Off The Components (COTS)

Short for commercial off-the-shelf, an adjective that describes software or hardware products that are ready-made and available for sale to the public. For example, Microsoft Office is a COTS product that is a packaged software solution for businesses. COTS products are designed to be implemented easily into existing systems without the need for customization.

Commercial-off-the-shelf (COTS) software and services are built and delivered usually from a third party vendor. COTS can be purchased, leased or even licensed to the general public.

COTS can be obtained and operated at a lower cost over in-house development[citation needed], and provide increased reliability and quality over custom-built software as these are developed by specialists within the industry and are validated by various independent organizations, often over an extended period of time.

Maintenance and Reengineering

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

Market Conditions - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.

Client Requirements - Over the time, customer may ask for new features or functions in the software.

Host Modifications - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.

Organization Changes - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

Types of maintenance

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

Corrective Maintenance - This includes modifications and updating done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.

Adaptive Maintenance - This includes modifications and updating applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.

Perfective Maintenance - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.

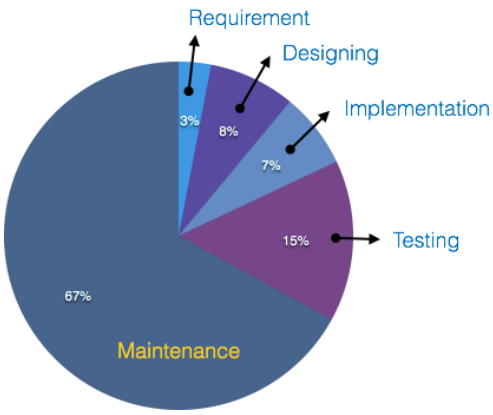
Preventive Maintenance - This includes modifications and updating to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Cost of Maintenance

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.

Maintenance Cost Chart

On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:



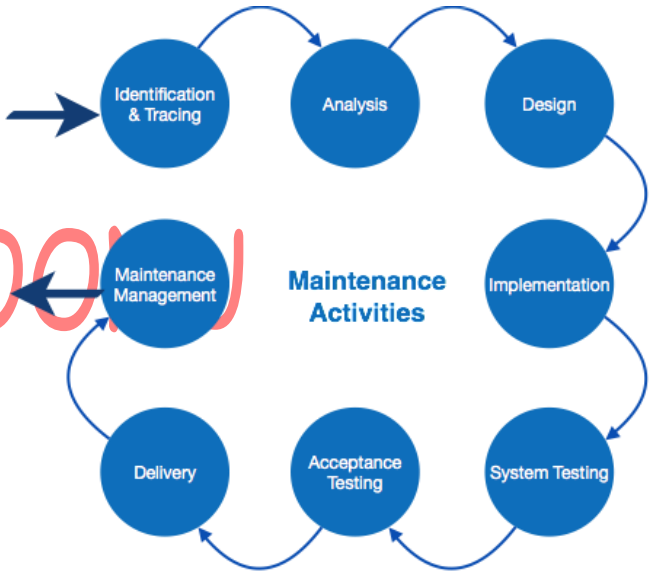
On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

Real-world factors affecting Maintenance Cost

- The standard age of any software is considered up to 10 to 15 years.
- Older softwares, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced softwares on modern hardware.
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

Software-end factors affecting Maintenance Cost

- Structure of Software Program
- Programming Language
- Dependence on external environment
- Staff reliability and availability
- Maintenance Activities
- IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.



These activities go hand-in-hand with each of the following phase:

Identification & Tracing - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages. Here, the maintenance type is classified also.

Analysis - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.

Design - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.

Implementation - The new modules are coded with the help of structured design created in the design step.Every programmer is expected to do unit testing in parallel.

System Testing - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.

Acceptance Testing - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.

Delivery - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

Training facility is provided if required, in addition to the hard copy of user manual.

Maintenance management - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

1.5. Automated Tools and Technology:

CASE Tools

CASE stands for Computer Aided Software Engineering. It means development and maintenance of software projects with help of various automated software tools.

CASE Tools

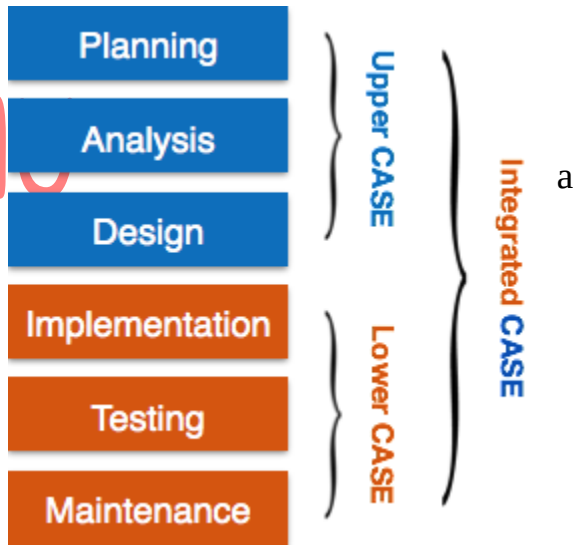
CASE tools are set of software application programs, which are used to automate SDLC activities. CASE tools are used by software project managers, analysts and engineers to develop software system.

There are number of CASE tools available to simplify various stages of Software Development Life Cycle such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools are to name a few.

Use of CASE tools accelerates the development of project to produce desired result and helps to uncover flaws before moving ahead with next stage in software development.

Components of CASE Tools

CASE tools can be broadly divided into the following parts based on their use at a particular SDLC stage:



- **Central Repository -** CASE tools require a central repository, which can serve as a source of common, integrated and consistent information. Central repository is a central place of storage where product specifications, requirement documents, related reports and diagrams, other useful information regarding management is stored. Central repository also serves as data dictionary.
- **Upper Case Tools -** Upper CASE tools are used in planning, analysis and design stages of SDLC.
- **Lower Case Tools -** Lower CASE tools are used in implementation, testing and maintenance.
- **Integrated Case Tools -** Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation.

CASE tools can be grouped together if they have similar functionality, process activities and capability of getting integrated with other tools.

Scope of Case Tools

The scope of CASE tools goes throughout the SDLC.

Case Tools Types

Now we briefly go through various CASE tools

Diagram tools

These tools are used to represent system components, data and control flow among various software components and system structure in a graphical form. For example, Flow Chart Maker tool for creating state-of-the-art flowcharts.

Process Modeling Tools

Process modeling is method to create software process model, which is used to develop the software. Process modeling tools help the managers to choose a process model or modify it as per the requirement of software product. For example, EPF Composer

Project Management Tools

These tools are used for project planning, cost and effort estimation, project scheduling and resource planning. Managers have to strictly comply project execution with every mentioned step in software project management. Project management tools help in storing and sharing project information in real-time throughout the organization. For example, Creative Pro Office, Trac Project, Basecamp.

Documentation Tools

Documentation in a software project starts prior to the software process, goes throughout all phases of SDLC and after the completion of the project.

Documentation tools generate documents for technical users and end users. Technical users are mostly in-house professionals of the development team who refer to system manual, reference manual, training manual, installation manuals etc. The end user documents describe the functioning and how-to of the system such as user manual. For example, Doxygen, DrExplain, Adobe RoboHelp for documentation.

Analysis Tools

These tools help to gather requirements, automatically check for any inconsistency, inaccuracy in the diagrams, data redundancies or erroneous omissions. For example,

Accept 360, Accompa, CaseComplete for requirement analysis, Visible Analyst for total analysis.

Design Tools

These tools help software designers to design the block structure of the software, which may further be broken down in smaller modules using refinement techniques. These tools provides detailing of each module and interconnections among modules. For example, Animated Software Design

Configuration Management Tools

An instance of software is released under one version. Configuration Management tools deal with

- Version and revision management
- Baseline configuration management
- Change control management
- CASE tools help in this by automatic tracking, version management and release management. For example, Fossil, Git, Accu REV.

Change Control Tools

These tools are considered as a part of configuration management tools. They deal with changes made to the software after its baseline is fixed or when the software is first released. CASE tools automate change tracking, file management, code management and more. It also helps in enforcing change policy of the organization.

Programming Tools

These tools consist of programming environments like IDE (Integrated Development Environment), in-built modules library and simulation tools. These tools provide comprehensive aid in building software product and include features for simulation and testing. For example, Cscope to search code in C, Eclipse.

Prototyping Tools

Software prototype is simulated version of the intended software product. Prototype provides initial look and feel of the product and simulates few aspect of actual product.

Prototyping CASE tools essentially come with graphical libraries. They can create hardware independent user interfaces and design. These tools help us to build rapid

prototypes based on existing information. In addition, they provide simulation of software prototype. For example, Serena prototype composer, Mockup Builder.

Web Development Tools

These tools assist in designing web pages with all allied elements like forms, text, script, graphic and so on. Web tools also provide live preview of what is being developed and how will it look after completion. For example, Fontello, Adobe Edge Inspect, Foundation 3, Brackets.

Quality Assurance Tools

Quality assurance in a software organization is monitoring the engineering process and methods adopted to develop the software product in order to ensure conformance of quality as per organization standards. QA tools consist of configuration and change control tools and software testing tools. For example, SoapTest, AppsWatch, JMeter.

Maintenance Tools

Software maintenance includes modifications in the software product after it is delivered. Automatic logging and error reporting techniques, automatic error ticket generation and root cause Analysis are few CASE tools, which help software organization in maintenance phase of SDLC. For example, Bugzilla for defect tracking, HP Quality Center.

Application Development Environments

An application development environment (ADE) is the hardware, software and/or the computing resources required for building software applications. ADE is the composite set of computing resources that provides an interface or application development, testing, deployment, integration, troubleshooting and maintenance services.

An application development environment is primarily the manufacturing unit or the software engineering platform of a software application. Its primary objective is to ensure that it provides comprehensive application development resources, can be integrated with underlying hardware architecture and includes application-wide reporting features.

Generally, ADE is used to build enterprise-wide applications that need to be executed and tested on an array of different computing devices and back-end or integrated platforms. ADE may include the basic hardware infrastructure, such as servers, computers and

handheld devices, that will host the application. These are combined with the software engineering resources, such as a programming language's integrated development environment (IDE), reporting and analysis software, troubleshooting tools, and other performance evaluation software utilities.

Lab Work

- **Discuss the use of CASE Tools**

CASE (Computer-Aided Software Engineering) packages are software packages that include many tools that can be helpful when it comes to database design. The main goal of these packages is to give designers a way of representing systems that are too complex to understand in their source code or schema-based forms. They help automate software development and maintenance tasks and usually contain tools for system analysis, project management, and design.

Uses in Databases

CASE tools can serve many functions in database design, including:

- Collecting and analyzing data
- Designing a data model
- Feasibility analysis
- Requirements definition
- Implementing the database
- Prototyping
- Data conversion
- Generating application code
- Generating reports
- Programming and testing
- Maintenance

Advantages and Disadvantage

CASE tools can provide many advantages when used in database design, including:

- Improved productivity in development
- Improved quality through automated checking
- Automatic preparation and update of documentation

- Encouragement of prototyping and incremental development
- Automatic preparation of program code from requirements definition
- Reduced maintenance systems

However, there are also some disadvantages to using the tools in database design, including:

- Cost increase
- Need for specialized training
- Limitations in flexibility of documentation
- Inadequate standardization
- Slow implementation
- Unrealistic expectations

Common CASE tool packages

There are three different types of CASE tools:
upper, lower, and integrated. Upper CASE tools focus on concept-level products and tend to ignore design, lower CASE tools concentrate on details of design such as physical design and testing, and integrated CASE tools combine the two to support the entire development. The most useful type in database design is integrated CASE tools. Some of the common packages on the market today include:

- IEF (Information Engineering Facility)
- IEW (Information Engineering Workbench)
- Oracle Designer

PHASE 2 :

SYSTEM ANALSYSI

LESSON 2

FEASIBILITY ANALYSIS AND THE SYSTEM PROPOSAL

INTRODUCTION

In the previous lesson we have completed three main activities involved during system analysis phase; requirement gathering, data and process modeling. Next, in this lesson Six, the system development team should perform the feasibility analysis to make a decision either to continue or cancel the project. The final decision on which solution to select would be made by the steering committee that was overseeing the project. This lesson consists of six sections:

- overview of feasibility and the system proposal
- tests for feasibility

- cost-benefits analysis techniques
- techniques for assessing economic feasibility
- feasibility analysis of candidate system
- the system proposal

LEARNING OUTCOMES

At the end of this lesson, students should be able to :

- identify and explain three feasibility checkpoints in the system's life cycle
- differentiate six types of feasibility testing
- perform various cost-benefit analyses
- use the techniques to assess the economic feasibility
- analyze the candidate system
- write suitable system proposal reports and plan for formal presentation

TERMINOLOGY

No	Word	Definition
1	Cultural feasibility	A measure of how the end users feel about the proposed system
2	Economic Feasibility	A measure of the cost-effectiveness of a project
3	Feasibility	The measure of how beneficial or practical is the system
4	Legal Feasibility	A measure of how well a solution can be implemented within existing legal and contractual obligations
5	Net present value	A technique that compares the annual costs

		and benefits of alternative solutions
6	Operational Feasibility	A measure of how well a solution meets the identified system to the organization to solve the problems requirements
7	Pay back analysis	A technique is a popular technique for determining if and when an investment will pay for itself.
8	Return on investment	An analysis technique compares the lifetime profitability of alternative solutions
9	Schedule Feasibility	A measure of how reasonable a project timetable is
10	System proposal	A report or presentation of a recommend system
11	Technical Feasibility	A measure of practicality of a technical solution and the availability of technical resources and expertise

6.1 OVERVIEW OF FEASIBILITY AND THE SYSTEM PROPOSAL

Feasibility analysis should be conducted during the analysis phase before the decision is made by the top management. Feasibility is a measure of how beneficial or practical an information system will be to the organization. There are three feasibility checkpoints during the system analysis phase of the system development life cycle. At the end of this checkpoints, the decision will be made either the system development works will be continued or cancelled. In every checkpoint, there are six tests that can be used for feasibility analysis; operational feasibility, cultural feasibility, technical feasibility, schedule feasibility

and economic feasibility.

Scope definition checkpoint

The first feasibility checkpoint where a measurement is more on answering questions such as; do the problems and opportunity warrant the cost of a detailed study of current system? At this checkpoint, the problems and opportunities are measured in terms of accuracy. This feasibility is not focus from the aspects of development costs, time and other. After estimating the real problems and opportunities, then, the system analyst will estimates the development cost.

Problem analysis checkpoint

The second feasibility checkpoint where occurs after the detailed measurement on problem analysis of the current system. After the problems and opportunities are defined, it's easy for the system analyst to come out with good development cost estimation. At this stage, the minimum development cost is equal with solving the problems. But the real development cost is done did the requirement analysis.

Decision analysis checkpoint

The third feasibility checkpoint where major feasibility analysis will be conducted after completing the user requirements. At this point, several alternative solutions are available and define its input/output methods, data storage, software and hardware requirements and other. This checkpoint is where all the detailed estimation of development cost is available.

6.2 SIX TESTS FOR FEASIBILITY

Feasibility can be viewed from different perspectives. During the system planning phase, we have identified six types of feasibility; operational feasibility, technical feasibility, economic feasibility, schedule feasibility, cultural feasibility and legal feasibility. We will explain more details all this feasibility in this sub topic.

6.2.1 Operational Feasibility

Operational Feasibility is feasibility that measure of how well a solution meets the system requirements in order to solve the problems and take advantage from the opportunities identified during the scope definition and analysis activity. It measure on how it satisfies the system requirements that have been identified in the requirements analysis activity. It's also concerns on what is the current problem and with cost of the solution provided; either the problem is still there or not.

6.2.2 Technical Feasibility

Technical feasibility is a measure of the practically of a technical solution and the availability of technical resources and expertise. Normally, technical feasibility addresses three major issues :

- Are the proposed system and the technology used practical enough?
- Do we currently posses the necessary technology?
- Do we have the necessary technical expertise?

6.2.3 Economic Feasibility

Economic feasibility is a measure to identify the financial benefits and costs related with the development project. Lots of people focus more on economic feasibility. At the early stage of system development project, the cost analysis amount is too little. It's impossible to estimate the cost at the early stage of the

system development because the system requirement has not been identified and the development did not started yet. Normally, the cost estimation should be worth it with the benefits got from the system. However, as soon as specific requirement and the proposed system have identified, the system analyst can weigh the cost and benefits of each alternative. This is referred to cost-benefit analysis.

6.2.4 Schedule Feasibility

Schedule feasibility measure of how reasonable a project timetable is. We ask the technical expertise, is the project can be completed within the deadlines? During the schedule feasibility analysis, it's important to define the deadlines for every phase if there is any.

6.2.5 Cultural Feasibility

Cultural feasibility also refers as political feasibility. This is related with operational feasibility. But, the operational feasibility deals more on how well the solution meets the system requirement but the cultural feasibility deals with how the end users feel about the proposed system. We can say that operational feasibility concerns with whether the system can work in solving the problem and cultural feasibility concerns with whether the system can be adapted in the organizational environment.

The following questions can help the team when doing cultural feasibility :

- Does the management support the system?
- How do the end users feel about their role in the proposed system?
- What end users or managers may resist or not use the system?
- How will the working environment of the end users can change with the proposed system?

6.2.6 Legal Feasibility

Information system has a legal impact. Legal feasibility is a measure of how well a solution can be implemented within existing legal and organization's policy. It's also regarding copyright issues. For example, if we need certain software in develop the system; we have to make sure that we use the licensed software.

6.2.7 Exercises

Answer TRUE or for FALSE for each of the questions below.

1. Feasibility is the measure of how beneficial or practical the development of an information system will be to an organization. **TRUE**
2. Operational feasibility is a measure of how well a solution meets the identified system requirements. **TRUE**
3. Cultural feasibility is a measure of how well the solution will be accepted in the organizational surroundings. **TRUE**
4. Operational feasibility is a measure of the practically of a technical solution and the availability of technical resources and expertise. **FALSE**
5. Schedule flexibility is a measure of how reasonable a project timetable is. **true**

6.3 COST-BENEFITS ANALYSIS TECHNIQUES

Economic feasibility has been defined as a cost benefit analysis. It's a way to estimate the benefits and costs in the system development. In order to determine this, we need to do a comparison between this.

6.3.1 Costs

An information system can have tangible costs and intangible costs. Tangible costs refer to items that you can easily measure in terms of money and with certainty. During the system development, tangible cost includes hardware costs,

labor costs, and others. Intangible costs refers to cost derived from the system, but it can't be measured in terms of money and with certainty. Examples of intangible costs are loss of customer goodwill, and employee morale.

In system development life cycle, there are two types of costs; costs of developing the system and costs of operating the system. The cost of developing the system refers to one time costs which it will not recur after the project has been completed. Several categories of costs that need to be considered such as :

- Personnel costs – salaries for the person involved such as system analyst, programmer, system designer, data entry personnel, manager and others.
- Computer usage – computer time for the activities such as programming, installation, data loading, storage and others.
- Training – the cost of training that will be provided for users who will use the system.
- Supply, duplication and equipment costs
- New hardware and software costs

There are two types of costs, fixed cost and variable cost. Fixed cost is a cost that occurs at a regular interval and at a relatively fixed rate such as salaries. Variable cost is a cost that occurs in proportion to some usage factors such as printer toner, paper and others.

6.3.2 Benefits

Similar with costs, an information system can provide a lot of benefits to an organization. It can be divided as tangible benefits and intangible benefits. A tangible benefit refers to benefits that can be measured in terms of money and with certainty. An example of tangible benefits are reduces the cost, increase the profits. An intangible benefit refers to benefits derived from the system, but it

can't be measured in terms of money and with certainty. An example of intangible benefits are competitive necessity, improved the organizational reputation, faster decision making. Normally, benefits can increase the organizations' profit and can decrease the costs, either in short term or long term.

6.3.3 Exercises

Answer TRUE or for FALSE for each of the questions below.

1. A cost-benefit analysis is a way to estimate and compare the benefits and costs in the system development. **TRUE**
2. Purchasing a new hardware is a tangible cost. **TRUE**
3. The cost of printer paper is not a variable cost. **FALSE**
4. When estimating the system development costs, the cost of buying computer cannot be included. **FALSE**
5. An example of intangible benefits is improving the organizational reputation. **TRUE**

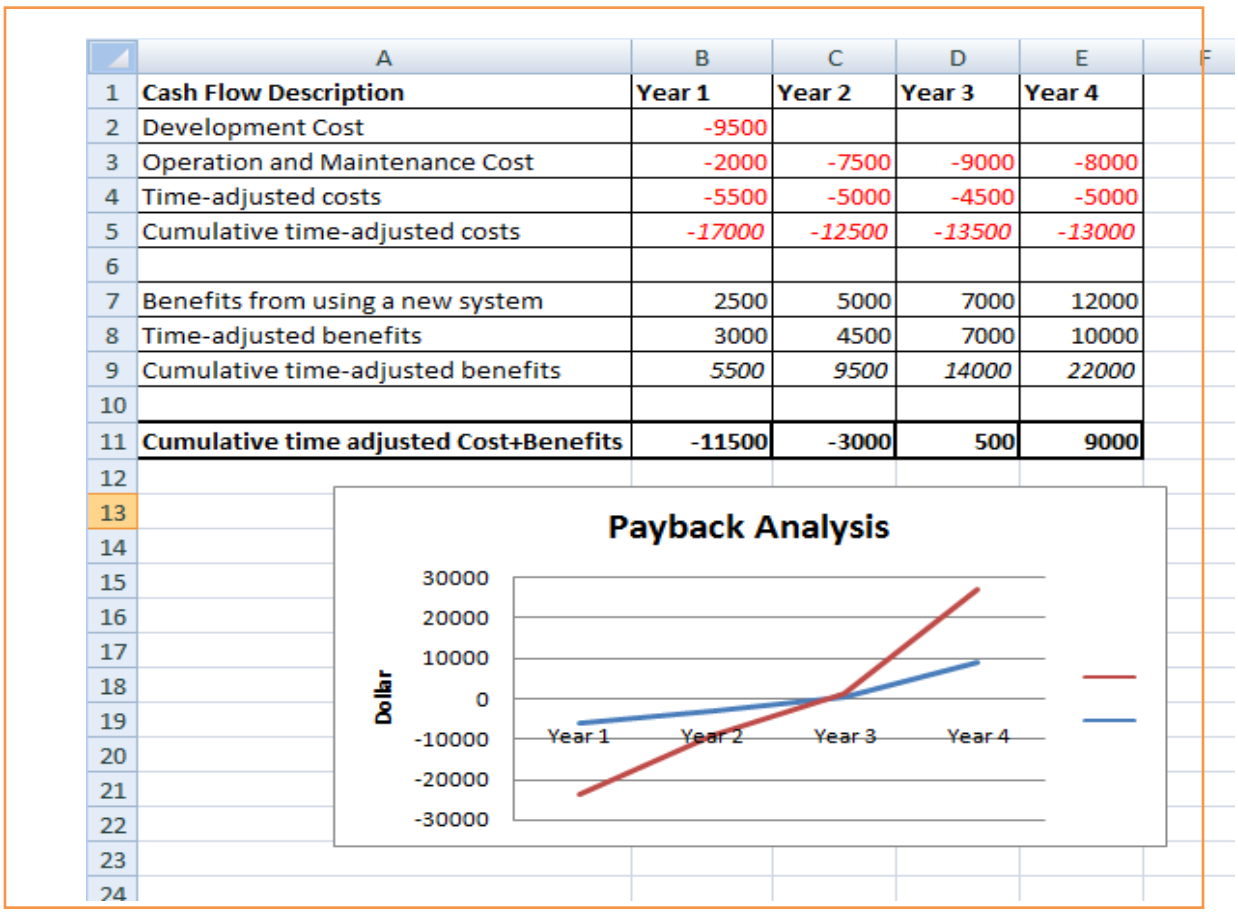
6.4 TECHNIQUES FOR ASSESSING ECONOMIC FEASIBILITY

There are three popular techniques can be used for assessing the economic feasibility; payback analysis, return on investment and net present value. We will discuss these three techniques in this following section.

6.4.1 Payback Analysis

Payback analysis technique is a popular technique for determining if and when an investment will pay for itself. As mentioned earlier, system development needs more costs during the earlier phases we need more time to get the benefits to overtake the costs. At the early phases of system development, most of the costs spend on analysis, design and implementation. After implementation, the cost is needed for the operating expenses that must be recovered. Payback

analysis determines how much time needed before benefits overtake the costs needed. This period is refer payback period. Figure 6-1 shows that before entered the year three, the benefits can overtake the costs spend.



	A	B	C	D	E
1	**Cash Flow Description**	**Year 1**	**Year 2**	**Year 3**	**Year 4**
2	Development Cost	-9500			
3	Operation and Maintenance Cost	-2000	-7500	-9000	-8000
4	Time-adjusted costs	-5500	-5000	-4500	-5000
5	Cumulative time-adjusted costs	*-17000*	*-12500*	*-13500*	*-13000*
6					
7	Benefits from using a new system	2500	5000	7000	12000
8	Time-adjusted benefits	3000	4500	7000	10000
9	Cumulative time-adjusted benefits	*5500*	*9500*	*14000*	*22000*
10					
11	**Cumulative time adjusted Cost+Benefits**	**-11500**	**-3000**	**500**	**9000**

Figure 6-1: Payback Analysis

6.4.2 Return On Investment

Return on Investment (ROI) analysis technique compares the lifetime profitability of alternative solutions. The ROI of the solution is a percentage rate that measures the relationship between the amounts the business get back from the investment and the amount invested.

Lifetime ROI = (Estimated lifetime benefits – Estimated lifetime costs)/

Estimated lifetime costs

As for example from Figure 6-1 above:

22000 – 13000 = 9000

Therefore the lifetime ROI is

Lifetime ROI = 9000/13000 = .692 = 70%

6.4.3 Net Present Value

Net present value is a technique that compares the annual costs and benefits of alternative solutions. We define the costs and benefits for each years of the system's lifetime. Using a same example as Figure 6-1, we can determine the net present value as in Figure 6-2 below.

	A	B	C	D	E	F
1	**Cash Flow Description**	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Total**
2	Development Cost	-9500				
3	Operation and Maintenance Cost	-2000	-7500	-9000	-8000	
4	Time-adjusted costs	-5500	-5000	-4500	-5000	
5	Present value of costs	*-17000*	*-12500*	*-13500*	*-13000*	
6	Total present value of lifetime costs					*-13000*
7						
8	Benefits from using a new system	2500	5000	7000	12000	
9	Time-adjusted benefits	3000	4500	7000	10000	
10	Present value of benefits	*5500*	*9500*	*14000*	*22000*	
11	Total present value of lifetime benefits					***22000***
12						
13	**Net present value**	**-11500**	**-3000**	**500**	**9000**	**9000**
14						

Figure 6-2: Net Present Value

6.4.4 Exercises

Answer TRUE or for FALSE for each of the questions below.

1. Payback analysis is a technique used for determining if and when an investment will pay for itself. **TRUE**
2. In payback analysis the period of time determine how much time needed before benefits overtake the costs needed is refer to payback period. **TRUE**
3. It's abnormal if the costs spend during the early phases is more than the late phase. **FALSE**
4. Return on investment is a measurement of calculation computes a percentage. **TRUE**
5. Net present value is a technique that compares the annual costs and benefits of alternative solutions. **TRUE**

6.5 FEASIBILITY ANALYSIS OF CANDIDATE SYSTEM

At the third checkpoint of the feasibility analysis, the team identifies all candidate system solutions and then analyzes each of them. We do a comparison for each of the candidate to choose which candidate is the best solution to be applied. There are two can be used to make a comparison and make a recommendation. These two matrices are candidate system matrix and feasibility analysis matrix.

6.5.1 Candidate System Matrix

Candidate system matrix allows us to make a comparison between all the candidate systems that available. It is a tool used to compare the similarities and differences between candidate systems based on certain characteristic. Table 6-1 shows an example of candidate systems matrix.

Table 6-1: Candidate Systems Matrix Template

Characteristics	Candidate 1	Candidate 2	Candidate 3

Portion of the system			
Benefits			
Software needed			
Input devices			
Output devices			

6.5.2 Feasibility Analysis Matrix

Feasibility analysis matrix is similar with candidate system matrix, but the different is it comes with an analysis and ranking of the candidate systems. It has same columns as in candidate system matrix with an additional column named ranking column. Table 6-2 shows an example of feasibility analysis matrix.

Table 6-2: Feasibility Analysis Matrix Template

	Weighting	Candidate 1	Candidate 2	Candidate 3
Description				
Operational Feasibility				
Technical Feasibility				
Economic Feasibility				
Schedule Feasibility				
Legal Feasibility				
Cultural				

Feasibility				
Weighted score				

6.5.3 Exercises

Answer TRUE or for FALSE for each of the questions below.

1. At the third checkpoint of the feasibility analysis, the team identifies all candidate system solutions and then analyzes each of them. **TRUE**
2. Each of the candidate systems can be compared using two tools; candidate systems matrix and feasibility analysis matrix. **TRUE**
3. Feasibility analysis matrix a tool used to compare the similarities and differences between candidate systems based on certain characteristic. **FALSE**
4. Candidate system matrix allows us to make a comparison between all the candidate systems that available. **TRUE**
5. The feasibility analysis matrix is a tool used to rank candidate systems. **TRUE**

6.6 THE SYSTEM PROPOSAL

Notice that during this decision analysis task, it involves identifying all candidate solutions, analyzing all candidate solutions using the matrix discussed earlier, ended up with selecting the best candidate to be implemented. Next, the team will continue with the preparing the system proposal. System proposal can be in a format of a report or a formal presentation of a recommended solution. Normally, both methods; report and formal presentation will be used.

6.6.1 Report

A report about the recommended system is prepared by the team to the several categories of audiences such as top management, clerk, manager and end users. The report consists of both primary and secondary elements. Primary element is the actual information that the report is intended to convey while secondary elements is information about the recommended system. Main basic and important element in this report is introduction, methods and procedures and conclusion. It's important to write the report in a proper way, well organized, using simple but meaningful words, so that it can deliver effectively to the target audience.

6.6.2 Formal Presentation

Formal presentation is another way in how the team presents the findings of the recommended system to solve the problem to the audience. The way how to conduct the presentation will effect the audience' confident towards the system. Effective and successful presentation requires significant preparation. Using presentation is good because the audience can respond during the session, and a good body language will convey the message which is difficult to show in written report. Visual aids can be used to support and convey the ideas.

6.6.3 Exercises

Answer TRUE or for FALSE for each of the questions below.

1. System proposal is a time when top management decides either to proceed or cancelled the project. **TRUE**
2. The team must choose either to write a report or present the system proposal. **FALSE**
3. When writing a system proposal we should include less information for higher-level management. **TRUE**

4. The systems proposal is a deliverable that is usually a formal written report or oral presentation intended for system users. **TRUE**
5. During the presentation, using visual aids is important in order to support the presentation. **TRUE**

SUMMARY

This is the end of lesson Six. In this lesson, we have learned :

- feasibility and the system proposal
- tests for feasibility
- cost-benefits analysis techniques
- techniques for assessing economic feasibility
- feasibility analysis of candidate system
- the system proposal

In the next lesson, we will discuss several types of development strategies that should be considered in the system development.

SELF ASSESSMENT

Fill in with the correct answer

1. _____ is the measure of how beneficial or practical the development of an information system will be to an organization. **Feasibility**

2. _____ is a measure of how well a solution meets the identified system requirements. **Operational feasibility**

3. _____ is a measure of how well a proposed system

solves the problems and takes advantages of the opportunities envisioned for the system. **Technical feasibility**

4. _____ feasibility is a measure of how reasonable the project timetable is

5. The three techniques for assessing economic feasibility are _____, _____, and _____ **payback analysis, return on investment, net present value**

6. In _____ the period of time that will elapse before accrued benefits overtake accrued and continuing costs is called the payback period. **payback analysis**

7. _____ are those benefits believed to be difficult or impossible to quantify. **Intangible benefits**

8. A _____ calculation computes a percentage. **return-on-investment**

9. The _____ complements the feasibility analysis matrix with an analysis and ranking of the candidate systems. **candidate systems matrix**

10. When writing a _____ you should include less information for higher-level management. **system proposal**

BOONU

Unit III: Determining System Requirement (12 Hrs)

3.1. Requirement Discovery, System Requirements: Functional and non-functional requirements

Requirements discovery is the process of distilling business needs into capabilities desired from an end solution. Requirements discovery leads to

technology strategy definition and eventually, into solution perspectives such as solution architecture, solution strategy and quality attribute concerns.

Requirements discovery is also often referred to as requirements elicitation, requirements gathering, requirements analysis, and requirements definition. We prefer to use the term because it's a little less high-brow, more meaningful, and more appropriate to the activity it is intended to describe.

Analysis is the first SDLC phase where we begin to understand, in depth, the needs for system. System analysis involves a substantial amount of effort and cost and is therefore undertaken only after management had decided that the systems development project under consideration has merit and should be pursed through this phase. Most observers would agree than many of the errors in developed system are directly traceable to inadequate efforts in the analysis and design phases of the life cycle. Industry studies show that 56% of systems problems are based on poor requirements definition, as opposed to 7% that are caused by poor coding. In the maintenance arena, 82% of the effort is due to poor requirements as opposed to 1% for poor coding. However, for many reasons, it is difficult to obtain a correct and complete set of requirements

As analysis can be divided into three main activities: Requirement determination, Requirements Structuring, and Alternative generation and selection. The third one is usually regarded as the automatic result from the first two processes. So, in this article, I focus on requirements determination and requirements structuring.

Requirements determination is the beginning sub phase of analysis. In this sub phase, analysts should gather information on what the system should do from as many sources as possible. There are some traditional methods to help collecting system requirements, such as interviewing, survey, directly observing users, etc. Nowadays, some modern requirements colleting methods, such as JAD and prototyping, emerged.

Requirements structuring is the process to use some kind of systematical and standard, well-structured methods to model the real world. Traditionally, we use data flow diagram for process modeling, decision table or decision tree for logic modeling, and Entity-relationship diagram for data modeling. These modeling tools usually separately model only one face of the real world. So, when we try to show the integral picture of a system, we usually choose more than one of the above requirements structuring methods.

Rapid Application Development (RAD) is an approach that promises better and cheaper systems and more rapid deployment by having system developers and end users work tighter jointly in real-time to develop systems. Usually, RAD allows usable systems to be built in as little as 60-90 days. RAD is not a single methodology but is more a general strategy of developing information systems. It brings several system development components together. Nowadays, a lot of RAD tools are available, such as VB for windows application, MBBuilder for MapInfo MapBasic.

Object-oriented analysis and development is a brand new methodology. Although OOP has become popular in computer world, whether OOA is superior to traditional methods is still a question mark. However, from the view of OO world, OOA seems having an important role to play in the future.

The objectives of this article are to introduce some widely adopted basic requirements determination and requirements structuring methods, compare and contrast those methods and try to find a best way for system requirements analysis.

Requirements Determination

Collection of information is at the core of systems analysis. Information requirement determination (IRD) is frequently and convincingly presented as the most critical phase of information system (IS) development, and many IS failures have been attributed to incomplete and inaccurate information requirements. [13] System analysts must collect the information about the current system and how users would like to improve their performance with new information system. Accurately understanding the users' requirements will help the system developing team deliver a proper system to the end users in limited time and limited budget. If user just wants an "ant", definitely, an "elephant" is improper. There are many methods to collect information. This article will discuss some basic and widely adopted ones of them.

Interviewing is one of the primary ways to gather information about an information system. A good system analyst must be good at interviewing and no project can be conduct without interviewing. There are many ways to arrange an effectively interview and no one is superior to others. However, experience analysts commonly accept some following best practices for an effective interview:

Prepare the interview carefully, including appointment, priming question, checklist, agenda, and questions.

- Listen carefully and take note during the interview (tape record if possible)
- Review notes within 48 hours after interview
- Be neutral
- Seek diverse views

Questionnaires have the advantage of gathering information from many people in a relatively short time and of being less biased in the interpretation of their results. Choosing right questionnaires respondents and designing effective questionnaires are the critical issues in this information collection method. People usually are only use a part of functions of a system, so they are always just familiar with a part of the system functions or processes. In most situations, one copy of questionnaires obviously cannot fit to all the users. To conduct an effective survey, the analyst should group the users properly and design different questionnaires for different group. Moreover, the ability to build good questionnaires is a skill that improves with practice and experience. When designing questionnaires, the analyst should concern the following issues at least:

- The ambiguity of questions.
- Consistence of respondents' answers.
- What kind of question should be applied, open-ended or close-ended?
- What is the proper length of the questionnaires?

The third one is **directly observing users**. People are not always very reliable informants, even when they try to be reliable and tell what they think is the truth. People often do not have a completely accurate appreciation of what they do or how they do it. This I especially true concerning infrequent events, issues from the past, or issues for which people have considerable passion. Since people can not always be trusted to reliably interpret and report their own actions, analyst can supplement and corroborate what people say by watching what they do or by obtaining relatively objective measures of how people behave in work situation. However, observation can cause people to change their normal operation behavior. It will make the gathered information biased.

The fourth one is **analyzing procedures and other documents**. By examining existing system and organizational documentation, system analyst can find out details about current system and the organization these systems support. In documents analyst can find

information, such as problem with existing systems, opportunities to meet new needs if only certain information or information processing were available, organizational direction that can influence information system requirements, and the reason why current systems are designed as they are, etc.

However, when analyzing those official documentations, analysts should pay attention to the difference between the systems described on the official documentations and the practical systems in real world. For the reason of inadequacies of formal procedures, individual work habits and preferences, resistance to control, and other factors, the difference between so called formal system and informal system universally exists.

The fifth one is **Joint Application Design (JAD)**. JAD is a facilitated, team-based approach for defining the requirements for new or modified information systems. JAD is started at IBM in the late 1970s. The main idea behind JAD is to bring together the key users, managers, and system analysts involved in the analysis of a current system. The primary purpose of using JAD in the analysis phase is to collect systems requirements simultaneously from the key people involved with the system. The result is an intense and structured, but highly effective, process. Having all the key people together in one place at one time allows analysts to see where there are areas of agreement and where there are conflicts. [2,21]

The typical participants in a JAD are: JAD session leader, end users, business managers, sponsor, system analysts, IS staff, scribe, etc. The JAD team is a group of from six to sixteen individual who all have a stake in designing a high quality system. Approximately two thirds of the group members are functional experts the other one third are systems professionals. JAD sessions are usually conducted in a location other than the place where the people involved normally work, and are usually held in special purpose rooms where participants sit around horseshoe-shaped tables. Involving so many different kinds of people in one workshop makes how to effectively and efficiently organize the JAD session a big challenge.

When a JAD is completed, the final result is a set of documents that detail the workings of the current system related to study of a replacement system. These requirements definition document generally includes business activity model and definitions, data model and definition, data input and output requirements. It may also include interface requirements, screen and report layouts, ad hoc query specifications, menus, and security requirements. When used at a later point in the system development life cycle, a JAD session can also be used to refine a system prototype, develop new job profiles for system users, or develop an implementation plan.

However, to exploit full potential of JAD, the groupware tools should be applied in JAD workshop sessions. The use of groupware tools to support the joint Application

Development technique increases the value of this technique dramatically. When groupware tools are used in an automated JAD workshop, they greatly facilitate the generation, analysis, and documentation of information. This is particularly valuable for JAD workshops conducted to define and build consensus on the requirements for new systems.

The Sixth one is **Prototyping**. Prototyping is a means of exploring ideas before you invest in them. Most system developers believe that the benefits from early usability data are at least ten times greater than those from late usability data. Prototyping allow system analysts quickly show users the basic requirement into a working version of the desired information system. After viewing and testing the prototype, the users usually adjust existing requirements to new ones. The goal with using prototyping to support requirement determination is to develop concrete specification for the ultimate system, not to build the ultimate system from prototyping.

Prototyping is most useful for requirements determination when user requirements are not clear or well understood, one or a few users and other stakeholders are involved with the system, possible designs are complex and require concrete form to fully evaluate, communication problems have existed in the past between users and analysts, and Tools and data are readily available to rapidly build working systems, etc.

When adopting prototyping, analysts should concern about the potential problems about this requirements determination method, such as informal documentation, ignored subtle but important requirements, etc.

When we choose requirements determination method for a specific project, there seven characters of them we should consider. They are Information Richness, Time Required, Expense, Chance for Follow-up and probing, Confidentiality, Involvement of Subject, Potential Audience. Table 1 concludes these characters of previously discussed six requirements determination methods.

Table 1

Characteristic	Interviews	Questionnaires	Observation	Document analysis	JAD	Prototyping
Information Richness	High	Medium to low	High	Low (passive) and old	High	Medium to High
Time Required	Can be extensive	Low to moderate	Can be extensive	Low to moderate	Dedicated period of	Moderate and can

					time of all kinds of involved people	be extensive
Expense	Can be high	Moderate	Can be high	Low to moderate	High	High
Chance for Follow-up and probing	Good	Limited	Good	Limited	Good	Good
Confidentiality	Interviewee is known to interviewer	Respondent can be unknown	Observee is known to interviewer	Depends on nature of document	All the people know each other	Usually know each other
Involvement of Subject	Interviewee is involved and committed	Respondent is passive, no clear commitment	Interviewees may or may not be involved and committed depending on whether they know if they are being observed	None, no clear commitment	All kinds of people are involved and committed	Users are involved and committed
Potential Audience	Limited numbers, but complete responses from those interviewed	Can be quite large, but lack of response from some can bias results	Limited numbers and limited time of each	Potentially biased by which documents were kept or because document not created for this purpose	Potentially biased by the subordinator intentionally don't want to directly point out his superior's errors.	Limited numbers; it is difficult to diffuse or adapt to other potential users

Requirements Structuring:

In the last section, we discuss about the methods to determine information system requirements. In this section, we will focus on the widely adopted methods used to present the requirements. Traditionally, there are three basic methods for requirements structuring:

- Data flow diagrams, it is widely used for process modeling
- Structure English, decision tables, and decision trees, they are used for logic modeling
- Entity and relationship diagram, it is used for data modeling

First, a **data flow diagram (DFD)** is a graphical tool that allows analysts (and users) to depict the flow of data in an information system. DFD graphically representing the functions, or processes, which capture, manipulate, store and distribute data between a system and its environment and between components within a system. The final deliverables and outcomes for DFD are:

- Context data flow diagram, which defines the boundary of the system
- DFDs of current physical system, which is used to determine how to convert the current system into its replacement.
- DFDs of current logical system
- DFDs of new logical system
- Thorough descriptions of each DFD component

Functional System Requirement

In Software engineering and systems engineering, a functional requirement defines a function of a system or its component. A function is described as a set of inputs, the behavior, and outputs.

Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. Behavioral requirements describing all the cases where the system uses the functional requirements are captured in use cases. Functional requirements are supported by non-functional requirements (also known as quality requirements), which impose constraints on the design or implementation (such as performance requirements,

security, or reliability). Generally, functional requirements are expressed in the form "system must do <requirement>", while non-functional requirements are "system shall be <requirement>". The plan for implementing functional requirements is detailed in the system design. The plan for implementing non-functional requirements is detailed in the system architecture.

As defined in requirements engineering, functional requirements specify particular results of a system. This should be contrasted with non-functional requirements which specify overall characteristics such as cost and reliability. Functional requirements drive the application architecture of a system, while non-functional requirements drive the technical architecture of a system.

In some cases a requirements analyst generates use cases after gathering and validating a set of functional requirements. The hierarchy of functional requirements is: user/stakeholder request → feature → use case → business rule. Each use case illustrates behavioral scenarios through one or more functional requirements. Often, though, an analyst will begin by eliciting a set of use cases, from which the analyst can derive the functional requirements that must be implemented to allow a user to perform each use case.

Nonfunctional Requirements BOONU

Nonfunctional requirements refer to a whole slew (I've identified more than 30) of attributes including performance levels, security, and the various "ilities," such as usability, reliability, and availability. Invariably, requirements definers get wrapped up in how the product/system is expected to function and lose sight of these added elements.

When such factors are not addressed adequately, seemingly proper product/system functioning in fact fails to function. For example, a system may identify customers in such a slow, insecure, and difficult to use manner that it can cause mistakes which make data unreliable, provoke frustration-based attempted work-arounds that can create further problems, and ultimately lead to abandonment.

That's the recognized way in which nonfunctional requirements impact product/system success. Other often unrecognized issues also need to be appreciated.

Functional requirements

The definition of a functional requirement is:

Any requirement which specifies what the system should do.

In other words, a functional requirement will describe a particular behaviour of function of the system when certain conditions are met, for example: “Send email when a new customer signs up” or “Open a new account”.

A functional requirement for an everyday object like a cup would be: “ability to contain tea or coffee without leaking”.

Typical functional requirements include:

- Business Rules
- Transaction corrections, adjustments and cancellations
- Administrative functions
- Authentication
- Authorization levels
- Audit Tracking
- External Interfaces
- Certification Requirements
- Reporting Requirements
- Historical Data
- Legal or Regulatory Requirements

Non-functional requirements

The definition of a non-functional requirement is:

Any requirement which specifies how the system performs a certain function.

In other words, a non-functional requirement will describe how a system should behave and what limits there are on its functionality.

Non-functional requirements generally specify the system’s quality attributes or characteristics, for example: “Modified data in a database should be updated for all users accessing it within 2 seconds.”

A non-functional requirement for the cup mentioned previously would be: “contain hot liquid without heating up to more than 45 °C”.

Typical non-functional requirements include:

- Performance – for example: response time, throughput, utilization, static volumetric
- Scalability
- Capacity
- Availability
- Reliability
- Recoverability
- Maintainability
- Serviceability
- Security
- Regulatory
- Manageability
- Environmental
- Data Integrity
- Usability
- Interoperability

It is important to correctly state non-functional requirements since they'll affect your users' experience when interacting with the system.

One way to prevent missing out on important non-functional requirements is to use non-functional requirement groups as a guide for listing them down. This blog post provides an explanation of each of the four main non-functional requirements groups and how they are used.

3.2. The Process of Requirement Discovery:

a. Problem Discovery and Analysis

Though the requirements gathering or the discovery phase is an essential phase in any SDLC cycle, it is more often than not overlooked with not enough ground work done. Many experienced project managers would agree that if the requirements are identified correctly and early in the project cycle, that alone

would result in reducing your project scheduled budget. The detail and the extent of correctness of the requirements has a large role to play in the overall success of the project.

Unfortunately this importance is often overlooked due to our anxiety to save time. One cannot stress enough on the cost of this blunder. If the requirements capture is not done accurately it might lead to the project deliverables not meeting the business user or technology requirements. Reworking on the project and fixing the errors would add on both to the project cost and time unnecessarily.

Best practices for requirement analysis

- Problem identification, definition and capture: The requirement analyst should identify the problem along with the business users and define it accurately. The requirement definition should be able to provide information on:
    - The problems the solution is aimed to solve
    - The benefits expected from the solution
- The feasibility of the requirements
- High Level Description of solution: The solution planned to be developed should be described at a high level along with the business needs it caters to.
- Address the needs of all the stakeholders and the users: This is a very important part of the requirement analysis and a step, which needs to be meticulously followed before freezing the requirements. This would help in the deployment phase of the project too, by getting the users adaptable to the new process or application.
- Solution definition: The solution's dos and don'ts both should be captured in detail. This would help in clearing up any ambiguity whatsoever.
- Feature definition: The application's planned features need to be captured at length. The functional and non-functional requirements need to be captured in detail along with the details on how the project is going to be executed etc.

The documentation should capture the processes, workflow, dependencies, and hierarchies and so on. Documenting all the above-mentioned categories by following the documentation best practices would help in realizing detailed and correct requirement analysis deliverables increasing the probability of project success.

Following these best practices would not only result in reducing project costs but can also help in shortening the development cycle, reducing scope for needs mismatch, enhance team productivity and most importantly increase the application's usability.

b. Requirements Discovery

Requirements discovery is the process of distilling business needs into capabilities desired from an end solution. **Requirements discovery** leads to technology strategy definition and eventually, into solution perspectives such as solution architecture, solution strategy and quality attribute concerns.

c. Documenting and Analyzing Requirements

d. Requirements Management

3.3. Traditional Methods for determining requirements: interview, questionnaire, sampling, survey

3.4. Modern Methods for determining requirements: Joint Application Design, Using Prototypes for Requirement determination,

3.5. Documenting requirements using Use Case List

Lab Work

- **Practice use case diagrams by using CASE Tools**

Unit IV: Data Modeling (12 Hrs)

Data modeling is the process of documenting a complex software system design as an easily understood diagram, using text and symbols to represent the way data needs to flow. The diagram can be used as a blueprint for the construction of new software or for re-engineering a legacy application.

A data model is a description of how data should be used to meet the requirements given by the end user (Ponniah). Data modeling helps to understand the information requirements. Data modeling differs according to the type of the business, because the business processes or each sector is different, and it needs to be identified in the modeling stage. Initial step is the analyzing the situation, gather data. Data modeling process starts with requirement gatherings. When developing the proper data model it is important to communicate with the stakeholders about the requirements. Data modeling is the act of exploring data oriented structures. This can be used for variety of purposes. One of the important functions of data modeling is that, it helps to understand the information requirements. Especially this makes both developers and end users lives easier. As mentioned above, data modeling helps the end users to define their requirements, and the developers are able to develop a system to meet those specified requirements.

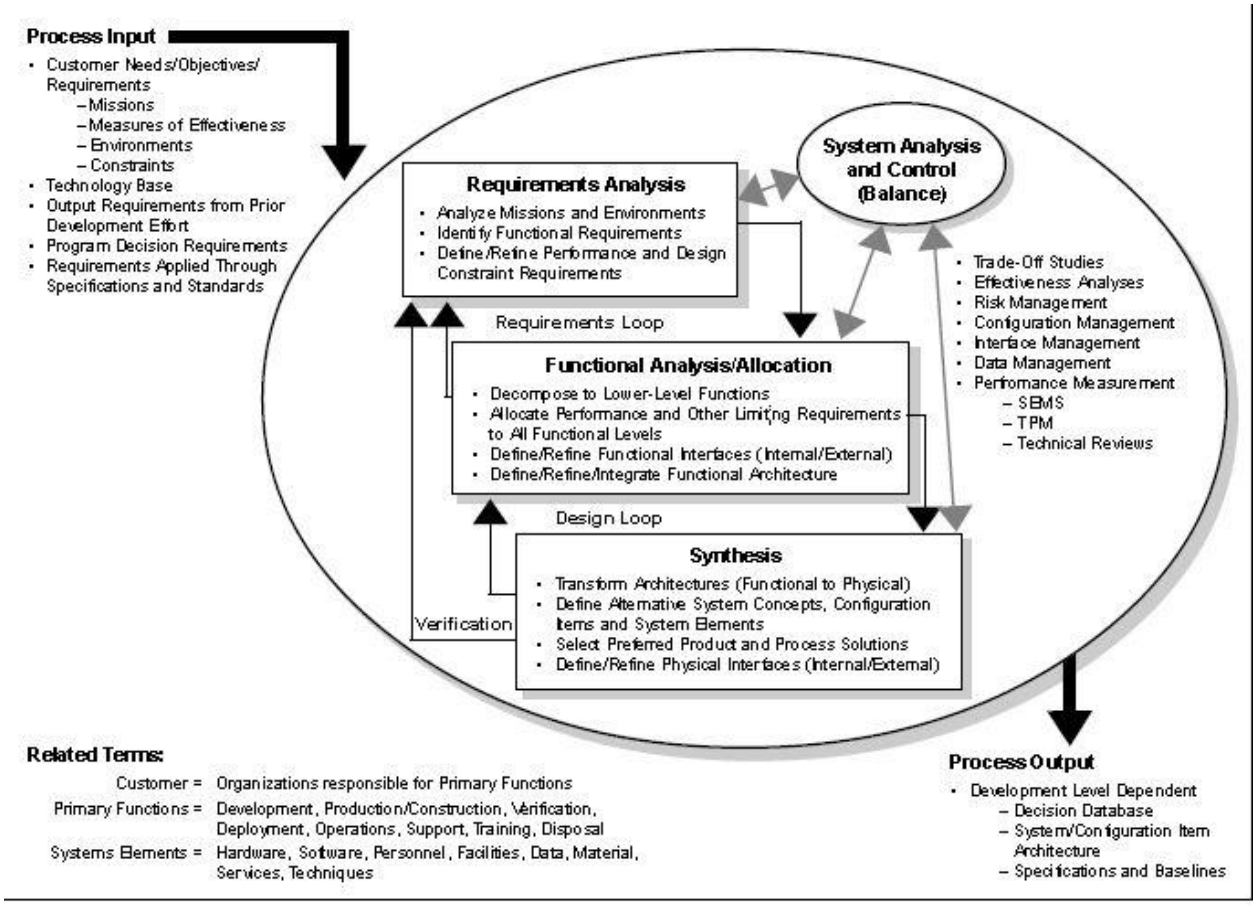
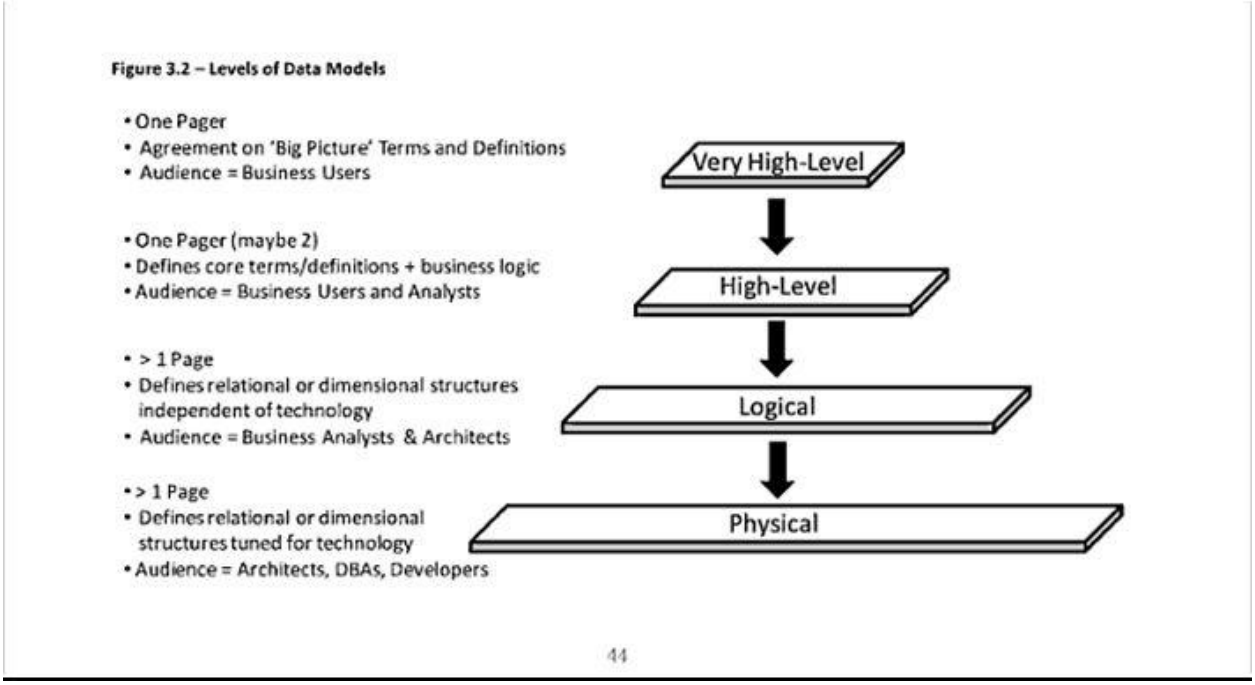


Figure 3-1. The Systems Engineering Process

Data model is a conceptual representation of data structures required for a database and is very powerful in expressing and communicating the business requirements (Learn Data Modeling). It visually represents the nature of data, business rules that are applicable to data, and how it will be organized in the database. There are three main designs for the data model, namely conceptual design, logical design and the physical design (Itl Education Solutions Limited). Data model is used by both functional team and the technical team in a project. Functional team consists of the business analysts and the end users, and the technical team consists of the developers and the programmers. There are data modelers who are responsible for designing the data model which meets the expectations of the functional team, and provide requirements for the technical team (Chuck Ballard, Dirk Herreman, Don Schau, Rhonda Bell, 1998).

Levels of Data Models



4.1.Data Modelling and Analysis

In today's information rich world, we are seeing more and more data-related analysis skills in business analysis jobs. We've been asked several times whether business intelligence and business analysis roles are really different roles, and how to build a career path into business analysis without getting wrapped into business intelligence and data analysis.Video Link:https://youtu.be/w1fq8t1xZ7k

Data Analysis and Data Modelling - What's the difference?

Nowadays, we are seeing an increase in data-related analysis skills in business analysis jobs. Some data skills are crucial for business analysts while others are better suited to other job functions - such as data analyst, financial analyst, reporting analyst, marketing analyst, and product management.

We take a look at the set of skills required for both data analysis and data modeling, investigate how data modeling can require some data analysis, and detail how skilled business analysts complete this level of analysis without technical data analysis skills.

What is Data Analysis?

- Data analysis is a technique to gain insight into an organisation's data. A data analyst might have the following responsibilities:

- To create and analyse important reports (possibly using a third-party reporting, data warehousing, or business intelligence system) to help the business make better decisions.
- To merge data from multiple data sources together, as part of data mining, so it can be analyzed and reported on.
- To run queries on existing data sources to evaluate analytics and analyses trends.

Data analysts will have hands-on access to the organization's data repositories and use their technical skills to query and manipulate the data. They may also be skilled in statistical analysis, having a high-level of mathematical experience.

Alternative job titles for this type of role include; Report Analyst, Data Warehousing Analyst, Business Intelligence Analyst, or even Product/Marketing Analyst. The common thread among this diverse set of job titles is that each role is responsible for analyzing a specific type of data or using a specific type of tool to analyses data.

What is Data Modelling?

Data modeling is a set of tools and techniques used to understand and analyze how an organization should collect, update, and store data. It is a critical skill for the business analyst who is involved with discovering, analyzing, and specifying changes to how software systems create and maintain information.

What does a Data Modeller do?

- They create an entity relationship diagram to visualise relationships between key business concepts.
- They create a conceptual-level data dictionary to communicate data requirements that are important to business stakeholders.
- They create a data map to resolve potential data issues for a data migration or integration project.

- A data modeller would not necessarily query or manipulate data or become involved in designing or implementing databases or data repositories.

Data Modeling sometimes needs Data Analysis

DA's often need to analyse data as part of making data modeling decisions, and this means that data modeling can include some amount of data analysis. A lot can be accomplished with very basic technical skills, such as the ability to run simple database queries. This is why you may see a technical skill like SQL in a business analyst job description.

Many DA's succeed without knowing these more technical skills, instead, they rely on their ability to collaborate with technical professionals and other knowledgeable stakeholders to ensure the data is understood well enough to make the right modelling decisions.

The non-technical DA can also evaluate sample data, interview stakeholders to discover possible data-related issues, review current state database models, and analyse exception reports.

While data analysis skills are valuable for the business analyst, they are not essential. However, data modelling falls squarely within the business analyst's domain.

Introduction to Entity Relationship Modelling

An **entity-relationship model** (ERM) is a theoretical and conceptual way of showing data **relationships** in software development. ERM is a database **modeling**technique that generates an abstract diagram or visual representation of a system's data that can be helpful in designing a relational database.

Conceptual Data Modelling using Entity Relationship Diagram (ERD)

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an **n-ary relation**.

Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database

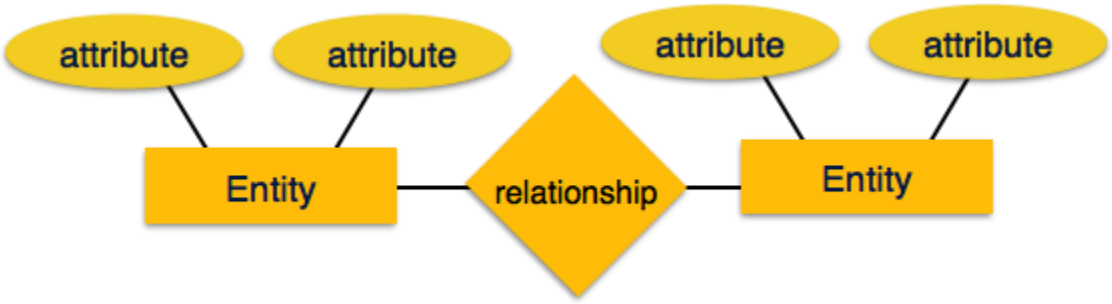
model, the ER Model creates entity set, relationship set, general attributes and constraints.

ER Model is best used for the conceptual design of a database.

ER Model is based on −

- **Entities** and their *attributes.*
- **Relationships** among entities.

These concepts are explained below.



- **Entity** − An entity in an ER Model is a real-world entity having properties called **attributes**. Every **attribute** is defined by its set of values called **domain**. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age, class, etc.

- **Relationship** − The logical association among entities is called ***relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.

  Mapping cardinalities −

  - one to one
  - one to many
  - many to one
  - many to many

- On an ER diagram, if the end of a relationship is straight, it represents 1, while a "crow's foot" end represents many.
- **A one to one relationship** - a man can only marry one woman, and a woman can only marry one man, so it is a one to one (1:1) relationship

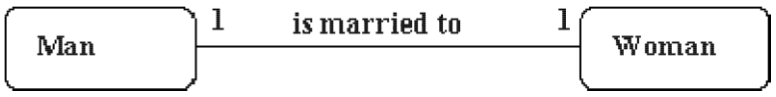


Figure: One to One relationship example

- **A one to may relationship -** one manager manages many employees, but each employee only has one manager, so it is a one to many (1:n) relationship

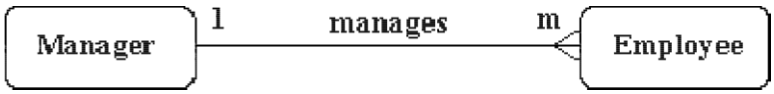


Figure: One to Many relationship example

- **A many to one relationship** - many students study one course. They do not study more than one course, so it is a many to one (m:1) relationship

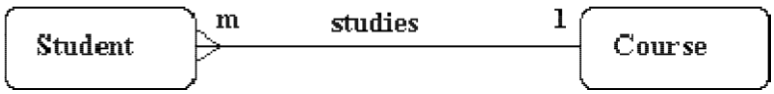


Figure: Many to One relationship example

- **A many to many relationship -** One lecturer teaches many students and a student is taught by many lecturers, so it is a many to many (m:n) relationship

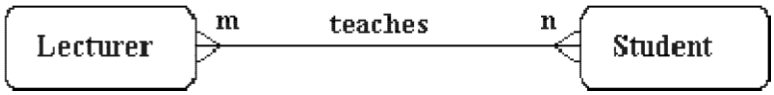


Figure: Many to Many relationship example

Optionality

A relationship can be optional or mandatory.

- If the relationship is mandatory
- An entity at one end of the relationship must be related to an entity at the other end.
- The optionality can be different at each end of the relationship
- For example, a student must be on a course. This is mandatory. To the relationship `student studies course' is mandatory.

- But a course can exist before any students have enrolled. Thus the relationship `course is studied by student' is optional.
- To show optionality, put a circle or `0' at the `optional end' of the relationship.
- As the optional relationship is `course is_studied_by student', and the optional part of this is the student, then the `O' goes at the student end of the relationship connection.

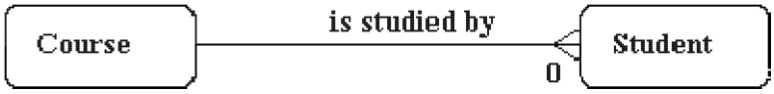


Figure: Optionality example

It is important to know the optionality because you must ensure that whenever you create a new entity it has the required mandatory links.

Advantages and Disadvantages of E-R Data Model

Following are advantages of an E-R Model:

• **Straightforward relation representation:** Having designed an E-R diagram for a database application, the relational representation of the database model becomes relatively straightforward.

• **Easy conversion for E-R to other data model:** Conversion from E-R diagram to a network or hierarchical data model can· easily be accomplished.

• **Graphical representation for better understanding:** An E-R model gives graphical and diagrammatical representation of various entities, its attributes and relationships between entities. This is turn helps in the clear understanding of the data structure and in minimizing redundancy and other problems.

Disadvantages of E-R Data Model

Following are disadvantages of an E-R Model:

• **No industry standard for notation:** There is no industry standard notation for developing an E-R diagram.

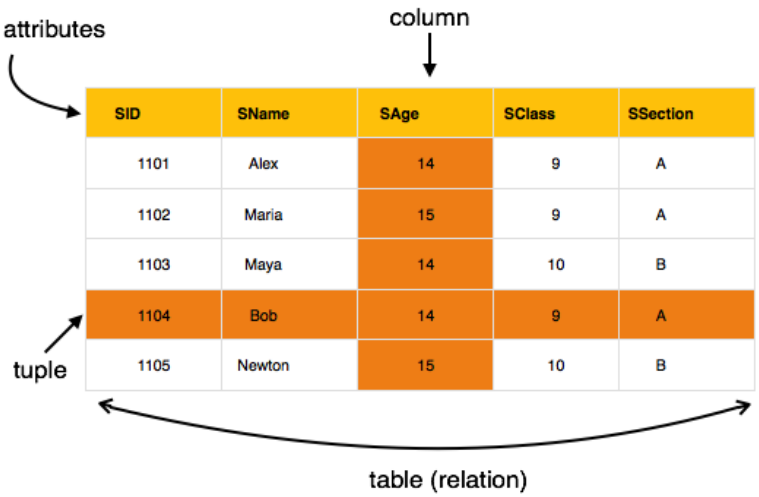
• **Popular for high-level design:** The E-R data model is especially popular for high level.

Advantages	Disadvantages
• Conceptual simplicity	• Limited constraint representation

- Visual representation
- Effective communication
- Integration with the relational database model
- Limited relationship representation
- No representation of data manipulation
- Loss of information

The main highlights of this model are −

- Data is stored in tables called **relations**.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain.



SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

Crow's-foot Notation of ER Diagram

An entity–relationship model (ER model) describes inter-related things of interest in a specific domain of knowledge. An ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between instances of those entity types.

Entity–relationship modeling was developed for database design by Peter Chen and published in a 1976 paper. However, variants of the idea existed previously. Some ER modelers show super and subtype entities connected by generalization-specialization relationships, and an ER model can be used also in the specification of domain-specific ontologies.

Crow's foot notation

Crow's foot notation, the beginning of which dates back to an article by Gordon Everest, is used in Barker's Notation, Structured Systems Analysis and Design Method (SSADM) and information engineering. Crow's foot diagrams represent entities as boxes, and relationships as lines between the boxes. Different shapes at the ends of these lines represent the cardinality of the relationship.

Crow's foot notation was used in the consultancy practice CACI. Many of the consultants at CACI (including Richard Barker) subsequently moved to Oracle UK, where they developed the early versions of Oracle's CASE tools, introducing the notation to a wider audience.

With this notation, relationships cannot have attributes. Where necessary, relationships are promoted to entities in their own right: for example, if it is necessary to capture where and when an artist performed a song, a new entity "performance" is introduced (with attributes reflecting the time and place), and the relationship of an artist to a song becomes an indirect relationship via the performance (artist-performs-performance, performance-features-song).

Background

There are many techniques are in use among data architects for designing data models, such as Entity Relationship Diagram (ERD) and Data Matrix etc. This article however

will be demonstrating only the most widely used technique, which is ERD. In ERD, there is a wide range of notations used by data architects for denoting the relationship and cardinality between the data entities. Some of such notations are OMT, IDEF, Bachman, Chen, Martin, UML and Crow Foo , however this article is intended for demonstrating Crow Foot Notation only.

Relationship and Cardinality/Multiplicity

The understanding of relationship and cardinality/multiplicity between entities are vital in modelling a database system. When it comes to relationship between entities, one of the following three relationships can exist between two entities.

- One to One
- One to Many
- Many to Many

Let me explain them with some examples.

- A car needs a tax disc (One to One),
- A car has four wheels (One to Many)
- A car can carry more than one person (One to Many)
- A driver is allowed to drive more than one car and a car can be driven by more than one driver (Many to Many).

One must understand the possible cardinality a table/entity can take in a relationship too . A cardinality is the number of rows a table can have in the relationship. As you may have noticed that I have used the terms table and entity interchangably, and the reason for that is an enitity eventually becomes a table in the database. we will see how in detail in the coming sections.

The possible cardinalities are

- One and Only One
- One or Many
- Zero or One or Many

Zero or One

In the relationships that we have mentioned the relationship section, when we say a car can carry more than one person then we know that a minimum of one person the car will

carry, which is the driver. Also it can carry many upto 5 or 7 or more depends on the vehicle. So we can now add further constraints - One to (One or Many). The one or many is called the cardinality/multipilicy in a relationship.

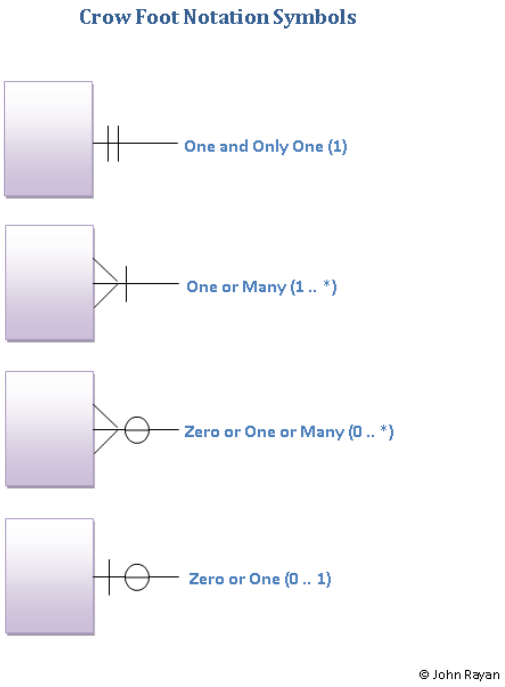
Let us see some more examples.

A car can have one and only tax disc – One to (One and Only One)

A driver can drive more than one vehicle but at the same time he doesn't need to own a car and can use public transport. In this instance the driver drives zero vehicles – One to (Zero or One or Many).

A car can be declared off the road and doesn't require a tax disc. In this instance, a car doesn't have a tax disc - One to (Zero or One).

A car owner or the owner's spouse or any comprehensive licence holder can drive the owner's car – (Many to One). Again, the car can be declared off road and no one can drive it – (Zero or One or Many) to One



You get the idea!

Crow Foot Notation Symbols

Now let us take a look at the crow foot notation symbols and understand their meaning before dive in to datamodelling.

To illustrate the above relationship and cardinality of the entities in an ERD, the Crow Foot Notation Symbols are used with cardinality. The Crow Foot's symbols and its meaning are given below. The symbols given in the parenthesis are used in UML.

4.2.Relationships: Unary, Binary and N-ary, Cardinalities in Relationships, IdentifyingRelationship, Non-Identifying Relationship, Associative Entity and Non-specific Relationships, Examples of ERD

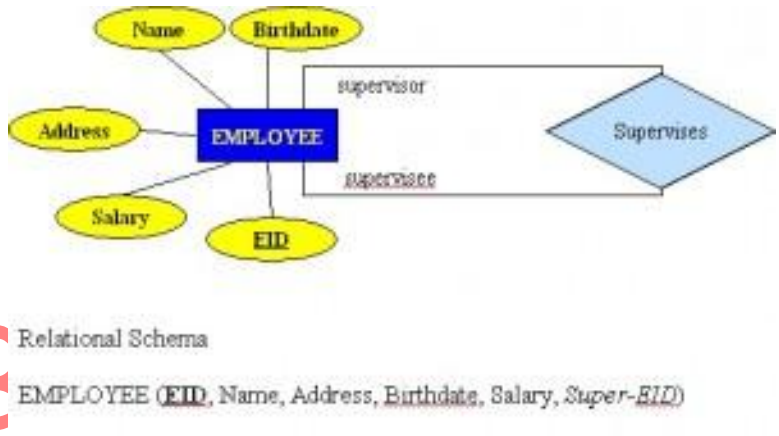
Reference Reading: http://www.opentextbooks.org.hk/ditatopic/30761

A unary

Relationship is when both participants in the relationship are the same entity. For Example: Subjects may be prerequisites for other subjects. A ternary relationship is when three entities participate in the relationship.

One in which a relationship exists between occurrences of the same entity set

The two keys (primary key and foreign key) are the same but they represents two entities of different roles relate to this relationship.



In some entities, a separate column can be created that refers to the primary key of the same entity set.

A Unary relationship between entities in a single entity type is presented on the picture below.

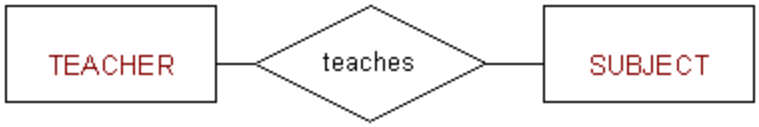
As we see, a person can be in the relationship with another person, such as:

- A woman who can be someone's mother
- A person that is a someone's child
- A teacher who teaches student

Binary and N-ary,

A binary relationship is when two entities participate and is the most common relationship degree. For Example: A unary relationship is when both participants in the relationship are the same entity.Binary and N-ary, Cardinalities in Relationships, Identifying Relationship, Non-Identifying Relationship, Associative Entity and Non-

specific Relationships, Examples of ERD Binary and N-ary, Cardinalities in Relationships, Identifying Relationship, Non-Identifying Relationship, Associative Entity and Non-specific Relationships, Examples of ERD.



Cardinalities in Relationships

Definition - What does Cardinality mean?

In the context of databases, cardinality refers to the uniqueness of data values contained in a column. High cardinality means that the column contains a large percentage of totally unique values. Low cardinality means that the column contains a lot of "repeats" in its data range.

It is not common, but cardinality also sometimes refers to the relationships between tables. Cardinality between tables can be one-to-one, many-to-one or many-to-many.

High cardinality columns are those with very unique or uncommon data values. For example, in a database table that stores bank account numbers, the "Account Number" column should have very high cardinality – by definition, every item of data in this column should be totally unique.

Normal cardinality columns are those with a somewhat unique percentage of data values. For instance, if a table holds customer information, the "Last Name" column would have normal cardinality. Not every last name will be unique (for example, there will likely be several occurrences of "Smith") but on the whole, the data is fairly non-repetitive.

Low cardinality columns are those with very few unique values. In a customer table, a low cardinality column would be the "Gender" column. This column will likely only have "M" and "F" as the range of values to choose from, and all the thousands or millions of records in the table can only pick one of these two values for this column.

Cardinality relationships between tables can take the form of one-to-one, one-to-many (whose reversal is many-to-one) or many-to-many. These terms simply refer to the relationships of data between the tables. For example, the relationship between the

"Customers" table and the "Bank Accounts" table is one-to-many, that is, one customer can have several accounts, but one account cannot belong to more than one customer. That is, of course, assuming this bank has never heard of joint accounts!

Cardinality is a constraint on a relationship specifying the number of entity instances that a specific entity may be related to via the relationship. Consider the relationship "works in".

When we ask How many employees can work in a single department? or How many departments can an employee work in? we are asking questions regarding the cardinality of the relationship.

The three classifications are: one-to-one, one-to-many, and many-to-many.

Below, the ERD shows a relationship between invoice lines and products.

The "n" represents an "arbitrary number of instances", and the "1" represents "at most one instance". We interpret the cardinality specifications with the following business rule statements:

The "n" indicates that the same Product entity can be specified on "any number of" Invoice Lines.The "1" indicates that an Invoice Line entity specifies "at most one" Product entity.

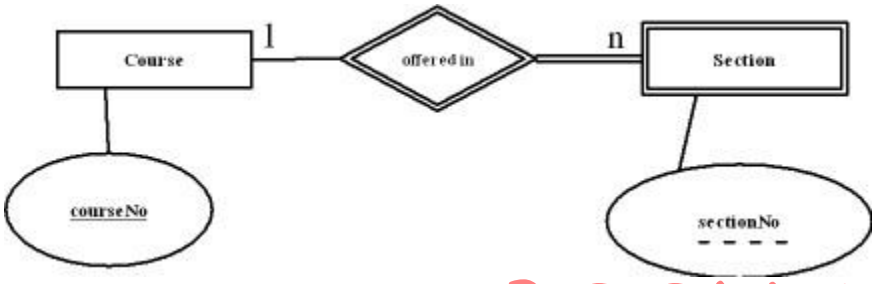
IdentifyingRelationship

An Identifying Relationship is a relationship between a strong and a weak entity type, where the key of the strong entity type is required to uniquely identify instances of the weak entity type. The weak entity type will have a partial key (discriminator) attribute that, in conjunction with the key of the strong entity type, uniquely identifies weak entity instances.

Consider a university environment where many different courses are offered. Many universities assign numbers to their courses and so, for example, we could have courses such as 2914, 3902, 3913, etc. It is typical that a Course may be offered zero, one, or

more times, in a single term. Hence, there may be several Sections of the same Course each term. Each Section needs to be identified distinctly from any other. The typical solution is use a section number such as 001, 002, 003, etc. The section number is appended to the course identifier to yield identifiers for the course offerings, such as 2914-001, 2914-002, 3902-050, etc. In this example, Section is modeled as a Weak Entity Type and its relationship to Course is an identifying relationship.

Note in the following diagram that the identifying relationship is drawn with a double line. Since Section is a weak entity it is shown with a double lined box, and since a Section cannot exist on its own but in relationship to a Course we shown a mandatory relationship too. Note that Section has a partial key sectionNo, Course has a key courseNo.

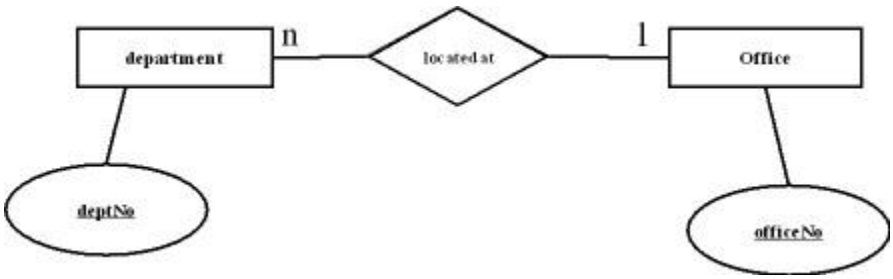


Non-Identifying Relationship

A Non-Identifying Relationship is a relationship between strong entity types. Each entity type has a key specified; each entity has an attribute that uniquely identifies it and distinguishes it from any other instance in the corresponding entity set or extension.

Suppose each University department has one office, but that the office could be the location for more than one department.

Associative Entity and Non-specific Relationships, Examples of ERD



Associative Entity

An associative entity is a term used in relational and entity–relationship theory. A relational database requires the implementation of a base relation (or base table) to resolve many-to-many relationships. A base relation representing this kind of entity is called, informally, an "associative table" (though it is a table like any other).

An associative entity (using Chen notation)

As mentioned above, associative entities are implemented in a database structure using associative tables, which are tables that can contain references to columns from the same or different database tables within the same database.

Concept of a mapping table

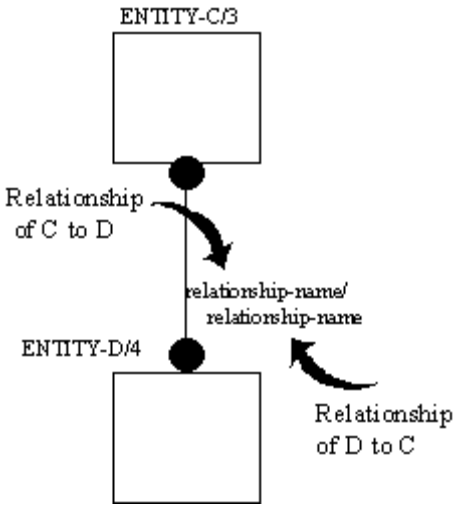
An associative (or junction) table maps two or more tables together by referencing the primary keys of each data table. In effect, it contains a number of foreign keys, each in a many-to-one relationship from the junction table to the individual data tables. The PK of the associative table is typically composed of the FK columns themselves.

Associative tables are colloquially known under many names, including association table, bridge table, cross-reference table, crosswalk, intermediary table, intersection table, join table, junction table, link table, linking table, many-to-many resolver, map table, mapping table, pairing table, pivot table (as used in Laravel - not to be confused with pivot table (spreadsheets)), or transition table.

Non-specific Relationships

In a key-based and fully attributed IDEF1X model, all associations between entities must be expressed as specific binary relationships. However, in the initial development of a model, it is often helpful to identify "non-specific relationship" between two entities. These non-specific relationships are refined in later development phases of the model.

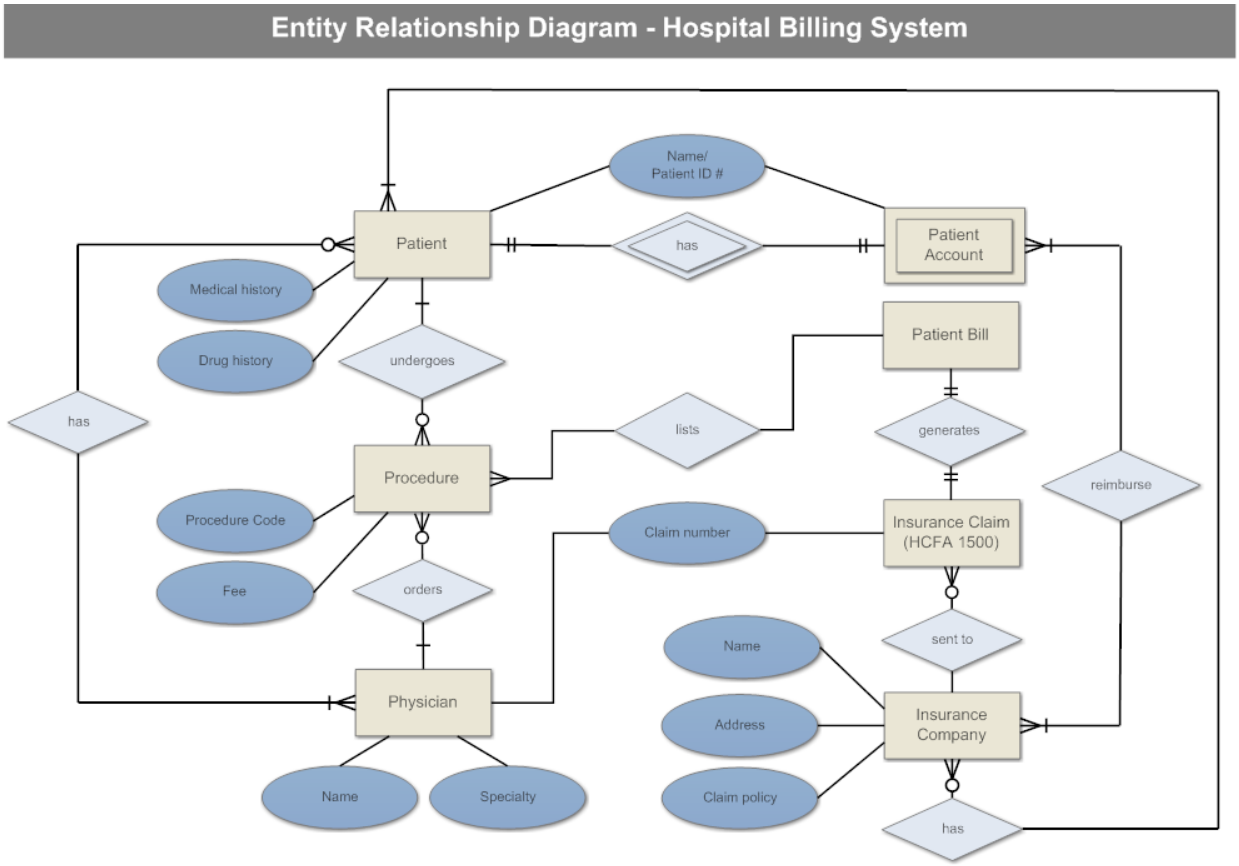
A non-specific relationship, also referred to as a "many to many relationship", is an association between two entities in which each instance of the first entity is associated with zero, one, or many instances of the second entity and each instance of the second entity is associated with zero, one, or many instances of the first entity. For example, if an employee can be assigned to many projects and a project can have many employees assigned, then the connection between the entities EMPLOYEE and PROJECT can be expressed as a non-specific relationship. This non-specific relationship can be replaced with specific relationships later on in the model development by introducing a third entity, such as PROJECT-ASSIGNMENT, which is a common child entity in specific connection relationships with the EMPLOYEE and PROJECT entities. The new relations ips would specify that an employee has zero, one, or more project assignments and that a project has zero, one or more project assignments. Each project assignment is for exactly one employee and exactly one project. Entities introduced to resolve non-specific relationships are sometimes called "intersection" or "associative" entities.



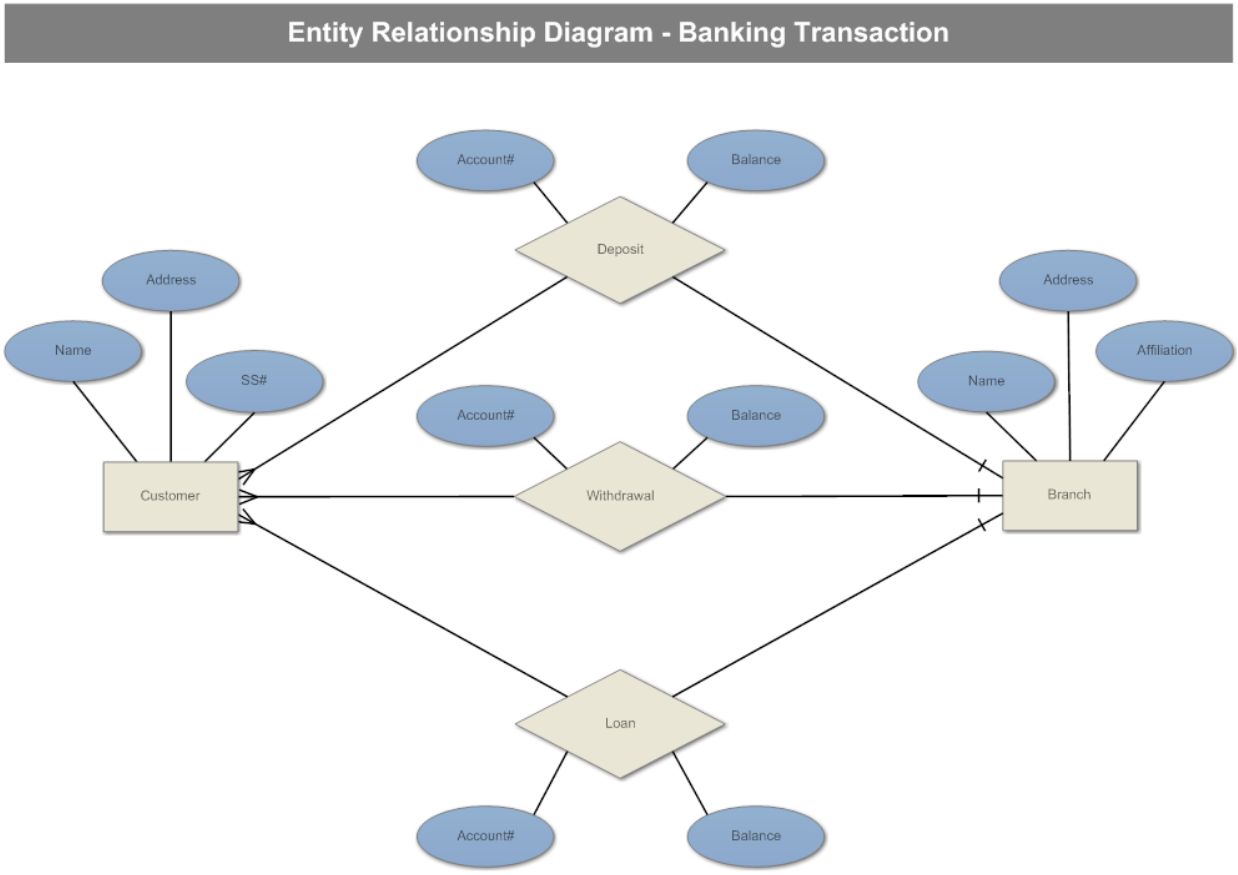
A non-specific relationship is depicted as a line drawn between the two associated entities with a dot at each end of the line. See Figure 4-2. A non-specific relationship is named in both directions. The relationship names are expressed as a verb phrase (a verb with optional adverbs and prepositions) placed beside the relationship line and separated by a slash, "/".

Examples of ERD

Hospital billing



Banking Transaction


Entity Relationship Diagram - Banking Transaction
Account#
Balance
Deposit
Address
Name
SS#
Customer
Account#
Balance
Withdrawal
Address
Affiliation
Name
Branch
Loan
Account#
Balance

4.3.The Process of Logical Data Modelling: Context Data Model, Key-based Data Model, Fully Attributed data model

Context Data Model

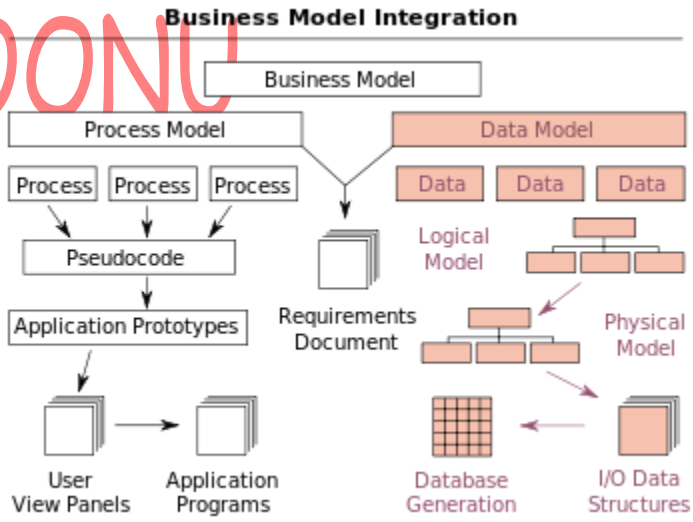
A context model (or context modeling) defines how context data are structured and maintained (It plays a key role in supporting efficient context management). It aims to produce a formal or semi-formal description of the context information that is present in a context-aware system. In other words, the context is the surrounding element for the system, and a model provides the mathematical interface and a behavioral description of the surrounding environment.

It is used to represent the reusable context information of the components (The top level classes consist of Operating system, component container, hardware requirement and Software requirement).

A key role of context model is to simplify and introduce greater structure into the task of developing context-aware applications.

Examples of Context Models

The Unified Modeling Language as used in systems engineering defines a context model as the physical scope of the system being designed, which could include the user as well as the environment and other actors. A System context diagram represents the context graphically..



Several examples of context models occur under other domains.

In the situation of parsing a grammar, a context model defines the surrounding text of a lexical element. This enables a context sensitive grammar that can have deterministic or stochastic rules. In the latter case, a hidden Markov model can provide the probabilities for the surrounding context.

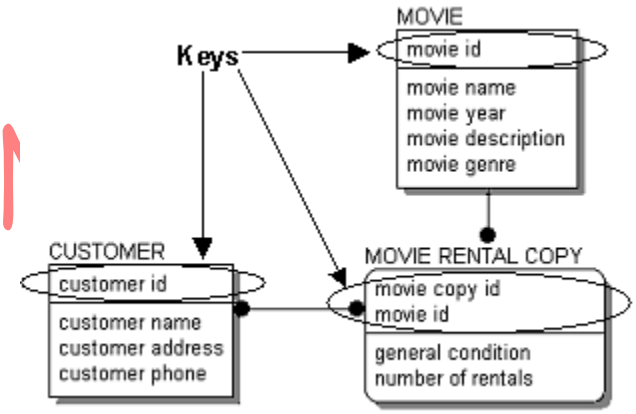
A context model can also apply to the surrounding elements in a gene sequence. Like the context rules of a grammar disambiguating a lexical element, this helps to disambiguate the role of the gene.

Within an ontology, a context model provides disambiguation of a subject via semantic analysis of information related to the subject.

In terms of a physical environment, a context model defines the external interfaces that a system will interact with. This type of context model has been used to create models for virtual environments such as the Adaptive Vehicle Make program. A context model used during design defines land, aquatic, or atmospheric characteristics (stated in terms of mathematical algorithms or a simulation) that the eventual product will face in the real environment.

Key-based Data Model

A Key-Based (KB) Model is a data model that fully describes all of the major data structures that support a wide business area. The goal of a KB model is to include all entities and attributes that are of interest to the business.



As its name suggests, a KB model also includes keys. In a logical model, a key identifies unique instances within an entity. When implemented in a physical model, a key provides easy access to the underlying data.

Basically, the key-based model covers the same scope as the Entity Relationship Diagram (ERD) but exposes more of the detail, including the context where detailed implementation level models can be constructed.

Key Types

Whenever you create an entity in your data model, one of the most important questions you need to ask is: “How can a unique instance be identified?” In order to develop a correct logical data model, you must be able to uniquely identify each instance in an entity.

In each entity in a data model, a horizontal line separates the attributes into two groups, key areas and non-key areas. The area above the line is called the key area, and the area below the line is called the non-key area or data area. The key area of CUSTOMER contains "customer-id" and the data area contains "customer-name," "customer-address," and "customer-phone."

Entity and Non-Key Areas

The key area contains the primary key for the entity. The primary key is a set of attributes used to identify unique instances of an entity. The primary key may be comprised of one or more primary key attributes, as long as the chosen attributes form a unique identifier for each instance in an entity.

An entity usually has many non-key attributes, which appear below the horizontal line. A non-key attribute does not uniquely identify an instance of an entity. For example, a database may have multiple instances of the same customer name, which means that "customer-name" is not unique and would probably be a non-key attribute.

Primary Key Selection

Choosing the primary key of an entity is an important step that requires some serious consideration. Before you actually select a primary key, you may need to consider several attributes, which are referred to as candidate key attributes. Typically, the business user who has knowledge of the business and business data can help identify candidate keys.

For example, to correctly use the EMPLOYEE entity in a data model (and later in a database), you must be able to uniquely identify instances. In the customer table, you could choose from several potential key attributes including: the employee name, a unique employee number assigned to each instance of EMPLOYEE, or a group of attributes, such as name and birth date.

The rules that you use to select a primary key from the list of all candidate keys are stringent and can be consistently applied across all types of databases and information. The rules state that the attribute or attribute group must:

- Uniquely identify an instance.
- Never include a NULL value.

- Not change over time. An instance takes its identity from the key. If the key changes, it is a different instance.

Be as short as possible, to facilitate indexing and retrieval. If you need to use a key that is a combination of keys from other entities, make sure that each part of the key adheres to the other rules.

Example:

Consider which attribute you would select as a primary key from the following list of candidate keys for an EMPLOYEE entity:

- employee-number
- employee-name
- employee-social-security number
- employee-birth-date
- employee-bonus-amount

BOONU

Fully Attributed data model

A logical **data model** is a **fully attributed data model** that is **fully** normalized. **Fully attributed** means that the entity types have all the **attributes** and relationship types for all the **data** that is required by the application(s) it serves. It may include: Restrictions on the **data** that can be held.

Different Types of Data Models

One of the things you often find people arguing about is what a data model is, and what it is for. Here's one of the secrets of analysis: when you find people arguing passionately about something, try to discover why they are both right. So it is with data models. Data models have many purposes. These cause differences in both style and content, which can cause confusion, surprise, and disagreement. This section looks at some different types of data models (I do not claim necessarily to have exhausted the possibilities) and how their purposes might lead them to differ for nominally the same scope. A particular data model may be of more than one of the types identified.

Physical Data Model

A *physical data model* represents the actual structure of a database—tables and columns, or the messages sent between computer processes. Here the entity types usually represent tables, and the relationship type lines represent the foreign keys between tables. The data model's structure will often be tuned to the particular needs of the processes that operate on the data to ensure adequate performance. It will typically include:

Restrictions on the data that can be held

Denormalization to improve performance of specific queries

Referential integrity rules to implement relationship types

Rules and derived data that are relevant to the processes of the application(s) the physical data model serves

Logical Data Model

There is a range of views on what a logical data model is. So I will start by talking about how I see them and then mention the divergences that I have noticed.

A *logical data model* is a fully attributed data model that is fully normalized. *Fully attributed* means that the entity types have all the attributes and relationship types for all the data that is required by the application(s) it serves. It may include:

Restrictions on the data that can be held

Rules and derived data that are relevant to the processes of the application(s) the logical data model serves

The main difference I see from this in practice is that many data models that are described as logical actually have some level of denormalization in them, particularly where change over time is involved.

A logical data model might relate to a physical data model, but this is not the only possibility. For example, with a software application, it would be quite appropriate for a logical data model to be developed of the user view of the application through the screens, and/or the computer interfaces to and from the application. This might be considerably less flexible than the underlying database, with restrictions imposed either by the application code, or by the configuration of the application.

It should be clear from this description that a physical data model can also be a logical data model, provided it does not include any denormalizations.

Conceptual Data Model

As with logical data models, there are some differing opinions about what a conceptual data model is. So again, I will state the way that I understand the term and then identify some key variations I have noticed.

A *conceptual data model* is a model of the things in the business and the relationships among them, rather than a model of the data about those things. So in a conceptual data model, when you see an entity type called car, then you should think about pieces of metal with engines, not records in databases. As a result, conceptual data models usually have few, if any, attributes. What would often be attributes may well be treated as entity types or relationship types in their own right, and where information is considered, it is considered as an object in its own right, rather than as being necessarily about something else. A conceptual data model may still be sufficiently attributed to be fully instantiable, though usually in a somewhat generic way.

Variations in view seem to focus on the level of attribution and therefore whether or not a conceptual data model is instantiable.

A conceptual data model might include some rules, but it would not place limits on the data that can be held about something (whether or not it was instantiable) or include derived data.

The result of this is that it is possible for a conceptual data model and a logical data model to be very similar, or even the same for the same subject area, depending on the approach that is taken with each.

Canonical Data Model

In the context of data models, a *canonical data model* means a data model that is fully normalized and in which no derived data is held. So a logical data model might or might not also be a canonical data model.

Application Data Model

An *application data model* is (obviously) one that relates to a particular application. It may be any or all of the following data models: conceptual, logical, physical, or canonical.

Business Requirements Data Model

The purpose of a *business requirements data model* is to capture and reflect a statement of business requirements. For such a model, it is important that the notation is simple and easily understood. This data model will form a basis for further analysis, so it does not

need to capture all the detail. It can also function as a useful framework for capturing business rules as part of the definition of the entity types.

So this type of data model is essentially a simplified conceptual data model—perhaps without cardinalities and without taking account of change over time, since most users do not understand their implications. My preferred notation for this, first introduced by CDIF (CASE Data Interchange Format) is particularly easy to read. It consists of boxes and arrows with the names of the relationship types on them, where the direction of the arrow just tells you in which direction to read the relationship type name. This is very easy and natural to read.

The idea is that you can just read around it following the arrows, to get phrases like:

Order of **offering** of **product** at **price**.

Order from **customer**.

Order delivered to **address**.

Address within **location**.

Delivery charge for **product** to **location**.

Integration Data Model

An *integration data model* integrates a number of separate applications. In order to do this, it needs to be instantiable. Its scope is usually either all the data for the applications it integrates or any data that is shared between at least two of these applications.

You can also use an integration data model to share data between enterprises, for example, in the supply chain.

Enterprise Data Model

An *enterprise data model* is a type of integration model that covers all (well, probably most in practice) of the data of an enterprise. Your Enterprise Architecture may include enterprise-wide data models that are also conceptual, logical, or physical data models.

For most types of data model, it is fairly obvious when you need to develop them. Enterprise data models, however, seem to be the exception. There are many cases where enterprise data model projects have been abandoned, or where the results have languished unused, even when what was asked for has been delivered. The reason for these failures is usually straightforward: It was not clear at the outset what questions the enterprise data model needed to provide the answers for nor was it clear what the economic imperative to answer these questions was.

Establishing the questions to be answered as the purpose of the enterprise data model is not only good because it means you have a clear purpose, it also means you know when you can stop data modeling. Otherwise it is perfectly possible and very tempting to develop the enterprise data model to a level of detail that is unnecessary, and this adds both cost and time to the exercise. It is, of course, always possible to return to the enterprise data model later and develop more detail when questions arise that require that detail.

There are two occasions when I think an enterprise data model is clearly justified. The first is when a major business re-engineering project is being undertaken and the processes of the enterprise are being fundamentally revised. In this case, developing an enterprise data model alongside the enterprise process model will deliver significant value to the re-engineering process. The second occasion arises from a bottom-up approach to enterprise architecture. As the need arises to integrate across applications, a logical data model showing the overlaps between the various systems becomes necessary. A key element of this will be master and reference data, since it is getting this consistent that enables consistency across different applications, but also the data exchanged between systems will need to be in scope. Since most transaction data is eventually transferred to data warehouses and reporting systems, it is likely that this will grow to cover most of the enterprises data.

Business Information Model

Business information models are a type of application data model that is used in data warehouses for reporting and slicing and dicing your transaction data. Instead of being normalized in their structure, these models are arranged in terms of “facts” (transactions, typically), and the “dimensions” (such as time, geography, or product lines) used to specify reports. The simplest structure is a “star” pattern, with a fact or group of facts in the middle, and dimensions radiating out from there. More complex structures resemble “snowflakes.” There are some special rules that apply to business information models; for example, only hierarchical relationship types are allowed, otherwise as you summarize up the relationships, your data may get counted more than once. On the other hand, at different levels in the hierarchy, you may use different relationship types for summarization.

Data Usage Model (Data Flow Diagram)

A *data usage model* shows where data is created and used by which processes. Some examples of data usage models are CRUD (create, read, update, delete) matrices and data flow diagrams. It is these that show how the process and data models interact with each other.

One of the challenges here is that the processes in a data usage model may themselves be things about which we wish to hold information, so you need to recognize that a process in such a model may also be represented in an entity-relationship model.

Summary of Different Types of Data Model

You will see from the earlier descriptions that these classifications of data models are not mutually exclusive.

This now enables me to draw your attention to the focus of this book, which is conceptual and logical data models that are also enterprise or integration data models. This does not mean that other types of data model will be ignored entirely, but I think this is where data modelers face the greatest challenge. Indeed, I have heard people say that enterprise data models are simply impossible, to which I would respond that the only thing you know for certain is that the person who says that does not know how to create one. I hope to show that such data models are quite achievable while pointing out some of the reasons that people fail and how you can overcome them.

Integration of Data and Data Models

In the previous sections I explained the different purposes and types of data model. In this section I am going to look in somewhat more detail at integration data models and at integrating data including an architecture and a methodology you can use for data integration.1 The reason for this is that the approach to data modeling presented here is very much aresponse to the demanding requirements of data integration.

I will start by introducing the basic principles for the architecture and integration methodology presented here.

Integration Models

The three-schema architecture for data models2 shows that, for any data model, it is possible to construct views on the original model. This principle can be extended to cover other types of model. In the integration of models, this process is reversed: a model is created for which the initial models are views. A model created in this way is an integration model with respect to the initial models in that it is capable of representing information with the scope of either or both of the original models.

You can create an integration model if you can establish a common understanding of the application models to be integrated. Difficulties in creating such a model point to a gap in human knowledge about the subject of the application models. Further, you may be able to integrate an application model to more than one integration model, where the integration models support different ways of looking at the

Integration models can themselves be integrated. This means that any arbitrary set of models can, in principle, be integrated at the cost of creating a new model

To have to create a new integration model each time you add a data model to the set can be time-consuming and expensive.What you want is an integration model that is stable in the face of the integration of additional models. Here stable means that the existing integration model does not need to be changed as more models are integrated, though extensions of the integration model may be necessary. So it is worth looking at some of the barriers that mean that the existing integration model needs to be changed, rather than simply extended.

Scope and Context

The scope of a data model is defined by the processes supported by its actual content. The context of a data model is the broadest scope that it could be part of without being changed.

When you create an integration model, its scope and context must be no smaller than the combined scope of the application models being integrated

If an additional application model needs to be integrated that falls outside the context of the existing integration model, then a new integration model will have to be developed to integrate the existing integration model with the additional application model.

However, you can choose for the initial integration model to have a wide model context. This means that it can support the information needs of many different applications, even though its initial model scope is limited to that of the models that it integrates, as shown in

You can then integrate further application models by extending the integration model—enlarging the model scope within the broad model context.

It turns out that the main barrier to having a broad context is the implementation of rules that apply in the narrow context of the original use and intention of the data model but that do not apply in a wider context.

Integration Model Content

The content of an integration model can be divided into primitive concepts and derived concepts. *Primitive concepts* are those that cannot be defined in terms of other concepts in the integration model, are in turn the building blocks for the definition of other

concepts, and can be further divided into foundation concepts, general concepts, and specific concepts as represented.

Discipline-specific concepts depend on general concepts that depend on foundation concepts, since all the lower concepts rely on the existence of one or more higher level concepts. For example, without the foundation concept of classification, relatively little can be said about anything.

At the top level, an integration model might have foundation concepts like classification, connection, and composition. General concepts might include agreements and organization, and finally discipline-specific concepts that are limited in their range of application, such as pumps and valves.

An integration model is not just a data model. It includes master and reference data that adds detail, particularly about the detailed kinds of things that are of interest.

A Full Integration Model

A full integration model, is more than just primitive concepts; it includes derived concepts—useful and valid combinations of primitive concepts. You only need to record derived concepts that are of interest, since their existence is implicitly recognized.

A primitive concept is not necessarily primitive forever. If a concept that is initially thought to be a primitive concept turns out not to be, then you can identify and add the concepts it is derived from,and add the derivation, so that it becomes a derived concept away from the front face of the pyramid. This allows flexibility to reflect an improved knowledge of the world, rather than reflecting knowledge of the world that is constrained by a modeler's knowledge at a point in time. This is one of the ways that you will need to maintain and extend an integration model.

Mapping Specifications

Mapping specifications specify the transformations that determine how the instances of one model can be represented as instances of another model.

When you create a mapping specification, it is important to note that:

• New concepts or constraints are not introduced in the mapping specification; that is, mapping specifications are limited to transformations of structure, terminology, and encoding.

• A complete mapping specification is bidirectional. However, the transformations of the first model to the second model may have to be specified separately from those of the

second model to the first model, and the mapping in one direction need not be derivable from the mapping in the other direction.

• Before you can support a bidirectional mapping, you may need to make explicit some of the context of the data model being integrated.

Overview of the Model Integration Process

The model integration process takes a number of application models and an integration model. It ensures that all the concepts of the application models are represented in the integration model, and it develops a mapping specification between the integration model and each of the application models.

There are three possible cases for the integration process:

The integration model and the application models both exist before the integration process starts.

The application models to be integrated exist before the integration process starts, but the integration model does not.

The integration model exists before the integration process starts, but the application model needs to be developed from some statement of requirements.

The process of integrating an application model with an integration model is illustrated in . The goal of this integration process is to allow the same information that is represented in the application model to be represented in the integration model without losing any meaning and to allow transformations between these representations. The result of the integration process is a mapping specification between the application model and a part of the integration model. In order to define this mapping, you may need to extend the integration model so that it precisely represents the concepts found in the application model.

The process of integrating application models with an integration model is divided into a number of steps, as follows:

Analyze the application models and identify the equivalent concept of the integration model, including any constraints that apply.Most application models have a context within which the model has to be understood but which is not explicit in the model itself. Usually it is inappropriate to add this information explicitly to the application model. In this case, you should capture these requirements in the mapping specification as part of the integration process.

If necessary, extend the integration model so that it includes all the concepts found in the application models

Identify the part of the integration model that represents the concepts in each application model

Create the mapping in each direction between each application model and the appropriate subset of the integration model (see **Figure 15**).

Data Mapping and Consolidation

Mapping between models is not sufficient to achieve integration. Integration requires reconciliation of information represented according to the different models.

Translate the data population 1 and 2 according to their source models into the data populations 10 and 20 according to the model C.

Identify which data elements in the two data sets represent the same things, and consolidate them in data population 3.

It should be noted that this requires a common, reliable, persistent identification mechanism. This will enable the different ways the same object is identified in different systems to be captured and effectively provide a translation service between those systems.

Concluding Remarks

In this chapter I have looked at the different types of data model that you can find, and in particular at the business of data integration. You will have noticed that data integration presents some particular challenges, and that the desirable characteristics of integration models are not necessarily the same as those developed for other purposes. In particular it is desirable that they are stable and extensible. Parts 2 and 3 of this book are largely about a way to achieve this.

4.4.Data Analysis: 1NF, 2NF and 3NF, Mapping Data Requirements to Locations

Database normalization is a database schema design technique, by which an existing schema is modified to minimize redundancy and dependency of data.

Normalization split a large table into smaller tables and define relationships between them to increases the clarity in organizing data.

Some facts about database normalization

- The words normalization and normal form refer to the structure of a database.

- Normalization was developed by IBM researcher E.F. Codd In the 1970s.
- Normalization increases the clarity in organizing data in Database.
- Normalization of a Database is achieved by following a set of rules called 'forms' in creating the database.

Database normalization rules

Database normalization process is divided into following the normal form:

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)
- Fourth Normal Form (4NF)
- Fifth Normal Form (5NF)

First Normal Form (1NF)

Each column is unique in 1NF.

Example:

Sample Employee table, it displays employees are working with multiple departments.

Employee	Age	Department
Melvin	32	Marketing, Sales
Edward	45	Quality Assurance
Alex	36	Human Resource

Employee table following 1NF:

Employee	Age	Department

Melvin	32	Marketing
Melvin	32	Sales
Edward	45	Quality Assurance
Alex	36	Human Resource

Second Normal Form (2NF)

The entity should be considered already in 1NF and all attributes within the entity should depend solely on the unique identifier of the entity.

Example:

Sample Products table:

productID	product	Brand
1	Monitor	Apple
2	Monitor	Samsung
3	Scanner	HP
4	Head phone	JBL

Product table following 2NF:

Products Category table:

productID	product
1	Monitor

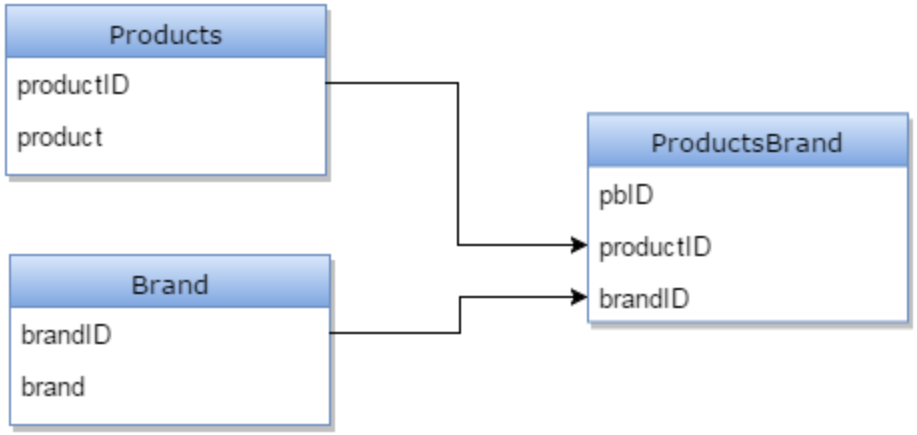
2	Scanner
3	Head phone

Brand table:

brandID	brand
1	Apple
2	Samsung
3	HP
4	JBL

Products Brand table:

pbID	productID	brandID
1	1	1
2	1	2
3	2	3
4	3	4



Second Normal Form (2NF)

Third Normal Form (3NF)

The entity should be considered already in 2NF and no column entry should be dependent on any other entry (value) other than the key for the table.
If such an entity exists, move it outside into a new table.
3NF is achieved are considered as the database is normalized.

Mapping Data Requirements to Locations

A Data Mapping Specification is a special type of data dictionary that shows how data from one information system maps to data from another information system. Creating a data mapping specification helps you and your project team avoid numerous potential issues, the kind that tend to surface late in development or during user acceptance testing and throw off project schedules, not to mention irritating your stakeholders.

(By the way, we cover data mapping specifications in more detail in Data Modeling for Business Analysts – a virtual class covering the most critical data modeling techniques you need to know.)

a. **Data Mapping** Specifications are particularly valuable in the following types of projects:
b. **Data Migration** – When source data is migrated to a new target data repository.

c. **Data Integration** – When source data is sent to a target data repository on a regular basis and the two data sources do not share a common data model. The integration can happen hourly, daily, weekly, monthly, or even in real-time as is typically required for a system integration.

These might sound similar, and they are. The primary difference between the two is that after a data migration project is complete, the original source data is no longer used or maintained while after a data integration project is complete, both data sources are maintained.

The Key Elements of a Data Mapping Specification

Essentially a data mapping specification will analyze, on a field-by-field basis, how to move data from one system to another. For example, if I were to orchestrate a data feed from the Bridging the Gap article repository to a search engine, I would want to map key attributes of the article, such as the title, category, and content to the attributes specified by the search engine. This analysis exercise would ensure that each piece of information ended up in the most appropriate place in the target data repository.

To achieve this goal, a Data Mapping Specification contains the following elements:

- List of attributes for the original source of data (often with additional information from the data dictionary)
- A corresponding (or "mapped") list of attributes for the target data repository (again, with additional information from the data dictionary)
- Translation rules defining any data manipulation that needs to happen as information moves between the two sources, such as setting default values, combining fields, or mapping values

A Data Mapping Sample Template

Here's a simple data mapping template and example you can use to see how this works in action. It's a hypothetical example assuming that we're sending a data feed from the *Bridging the Gap* article repository to a search engine.

	Source Data - BTG Article				Target Data - Search Engine				Translation Rules		
Attribute	lame	Required	Type	Notes	Attribute Name	Required	Type	Notes	Direct Map?	Default Value	Additional Logic
Article T	e	Yes	Text	Can contain HTML.	ContentTitle	Yes	Text		Yes		
Article A	hor	Yes	Look-Up		n/a	n/a	n/a		n/a	n/a	
Article C	egory	Yes	Look-Up		Keyword	No	Text		Yes	n/a	If multiple categories, create a comma separated list.
Article C	tent	No	Text	Can contain HTML.	ContentText	Yes	Text	Truncates to 4000 chars	Yes	n/a	If truncated, back-up to last fu sentence and add "..."
URL		Yes	URL		URL	Yes	URL	Truncates to 150 chars	Yes		Flag any truncated URLs for errors.

As you can see, even a simple data mapping exercise encounters potential data mapping issues that would best be handled proactively in a business-centered way than retroactively in a technically-focused way.

Solid business analysis through data modeling will prevent these issues before they happen, by discovering them in advance, collaborating with business and technical stakeholders to find feasible solutions, and initiating any appropriate data clean-up and normalization efforts.

BOONU

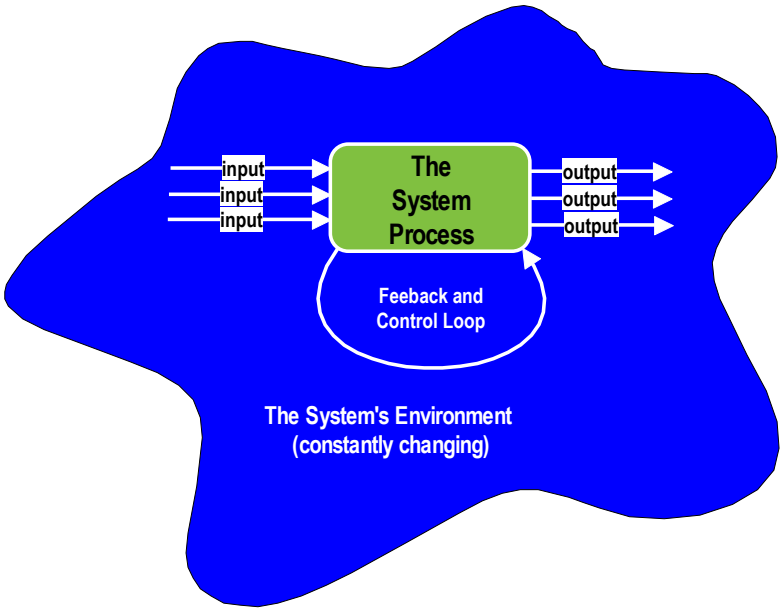
Unit V: Process Modeling (12 Hrs)

7.1. Process Modelling, Data Flow Diagram (DFD), System concepts for process modelling, Components of DFD, Data Flow Diagramming Rules, The Process of Logical Process Modelling

7.1.1 What is process modeling?

- Process modeling is a technique for organizing and documenting the structure and flow of data through a system's PROCESSES and/or the logic, policies, and procedures to be implemented by a system's PROCESSES.
- Process modeling originated in classical software engineering methods.
- A systems analysis process model consists of *data flow diagrams* (DFDs).
- A data flow diagram (DFD) is a tool that depicts the flow of data through a system and the work or processing performed by that system. Synonyms include bubble chart, transformation graph, and process model.
  - A System is a Process
    - The simplest process model of a system is based on inputs, outputs, and the system itself – viewed a process.
    - The process symbol defines the boundary of the system.
    - The system is inside the boundary; the environment is outside that boundary.
    - The system exchanges inputs and outputs with its environment

- Because the environment is always changing, well designed systems have a feedback and control loop to allow the system to adapt itself to changing conditions.
- Consider a business as a system. It operates within an environment that includes customers, suppliers, competitors, other industries, and the government. Its inputs include materials, services, new employees, new equipment, facilities, money, and orders (to name but a few). Its outputs include products and/or services, waste materials, retired equipment, former employees, and money (payments). It monitors its environment to make necessary changes to its product line, services, operating procedures, and the like.



The Classical Process Model of a System

A common theme in systems analysis is the use of models to view or present a system. As shown in the figure above, the simplest process model of a system is based on inputs, outputs, and the system itself – viewed a process.

7.1.2 Data Flow Diagram (DFD)

A data flow diagram (DFD) shows how data moves through an information system but does not show program logic or processing steps. A set of DFDs provides a logical model that shows what the system does, not how it does it. That distinction is important because focusing on implementation issues at this point would restrict your search for the most effective system design.

A **data flow diagram** (**DFD**) is a graphical representation of the "flow" of data through an information **system**, modelling its process aspects. A **DFD** is often used as a preliminary step to create an overview of the **system** without going into great detail, which can later be elaborated.

7.1.3 System concepts for process modelling,

Process modeling is a technique for organizing and documenting the structure and flow of data through a **system's PROCESSES** and/or the logic, policies, and procedures to be

implemented by a **system's PROCESSES**. ... A systems analysis **process model** consists of data flow diagrams (DFDs).

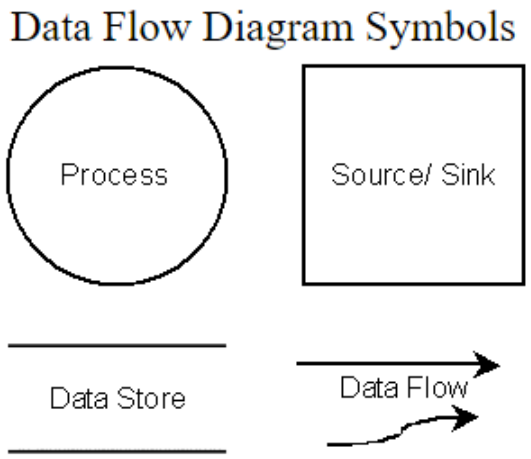
A process model describes business processes—the activities that people do. Process models are developed for the as-is system and/or the to-be system. This chapter describes data flow diagramming, one of the most commonly used process modeling techniques.

7.1.4 Components of DFD,

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model.

- There are Four symbols and one connection:
  - The rounded rectangles represent *processes* or work to be done.
  - The squares represent *external agents* – the boundary of the system.
  - The open-ended boxes represent *data stores*, sometimes called files or databases, and correspond to all instances of a single entity in a data model.



– The arrows represent *data flows*, or inputs and outputs, to and from the processes.

Process

A process receives input data and produces output with a different content or form. Processes can be as simple as collecting input data and saving in the database, or it can be complex as producing a report containing monthly sales of all retail stores in the northwest region.

Every process has a name that identifies the function it performs.

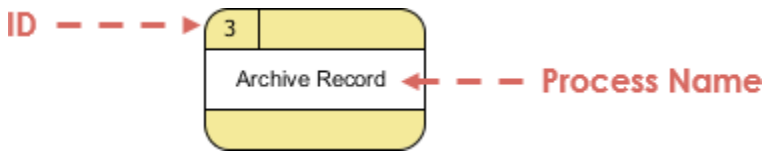
The name consists of a verb, followed by a singular noun.

Example:

- Apply Payment
- Calculate Commission
- Verify Order

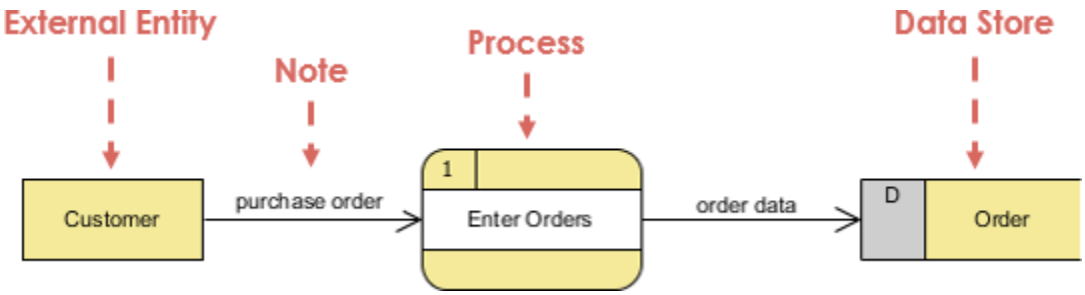
Notation

- A rounded rectangle represents a process
- Processes are given IDs for easy referencing



DFD Process

DFD Process Example



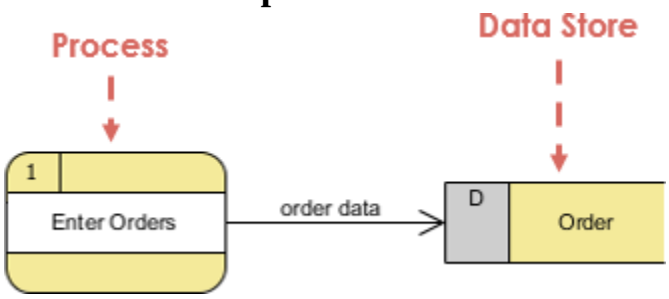
Data Flow

A data-flow is a path for data to move from one part of the information system to another. A data-flow may represent a single data element such the Customer ID or it can represent a set of data element (or a data structure).

Example:

Customer_info (LastName, FirstName, SS#, Tel #, etc.)
Order_info (OrderId, Item#, OrderDate, CustomerID, etc.).

Data flow Example:



DFD Data Store Example

Notation

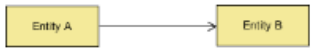
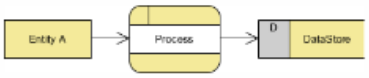
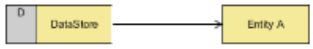
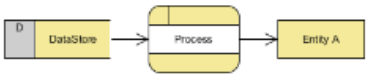
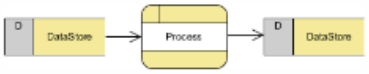
- Straight lines with incoming arrows are input data flow
- Straight lines with outgoing arrows are output data flows

Note that:

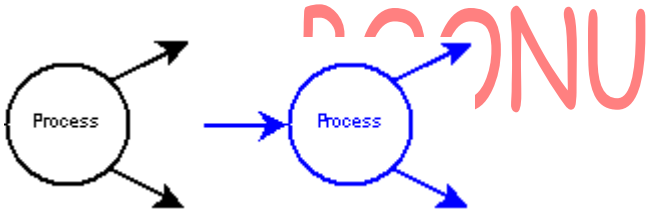
Because every process changes data from one form into another, at least one data-flow must enter and one data-flow must exit each process symbol.

7.1.5 Data Flow Diagramming Rules,
Rule of Data Flow

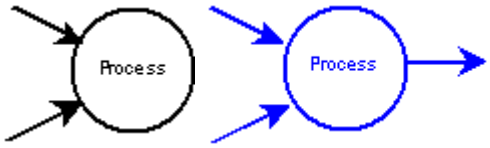
One of the rule for developing DFD is that all flow must begin with and end at a processing step. This is quite logical, because data can't transform on its own with being process. By using the thumb rule, it is quite easily to identify the illegal data flows and correct them in a DFD.

Wrong	Right	Description
		An entity cannot provide data to another entity without some processing occurred.
		Data cannot move directly from an entity to a data story without being processed.
		Data cannot move directly from a data store without being processed.
		Data cannot move directly from one data store to another without being processed.

1. All processes should have unique names. If two data flow lines (or data stores) have the same label, they should both refer to the exact same data flow (or data store).
2. The inputs to a process should differ from the outputs of a process.
3. Any single DFD should not have more than about seven processes.
4. No process can have only outputs. (This would imply that the process is making information from nothing.) If an object has only outputs, then it must be a source.

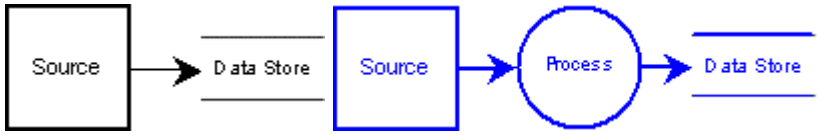


5. No process can have only inputs. (This is referred to as a "black hole".) If an object has only inputs, then it must be a sink.



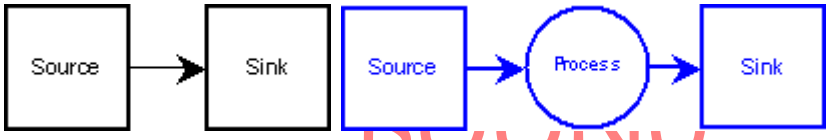
6. A process has a verb phrase label.
7. Data cannot move directly from one data store to another data store. Data must be moved by a process.

8. Data cannot move directly from an outside source to a data store. Data must be moved by a process that receives data from the source and places the data in the data store.



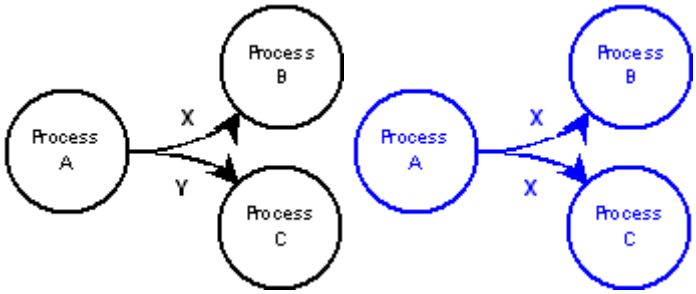
9. Data cannot move directly to an outside sink from a data store. Data must be moved by a process.

10. A data store has a noun phrase label.
11. Data cannot move directly from a source to a sink. It must be moved by a process if the data are of any concern to the system. If data flows directly from a source to a sink (and does not involved processing) then it is outside the scope of the system and is not shown on the system data flow diagram DFD.

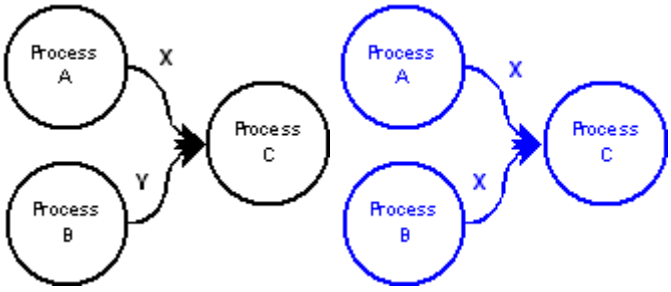


12. A source/sink has a noun phrase label.
13. A data flow has only one direction between symbols. It may flow in both directions between a process and a data store to show a read before an update. To effectively show a read before an update, draw two separate arrows because the two steps (reading and updating) occur at separate times.

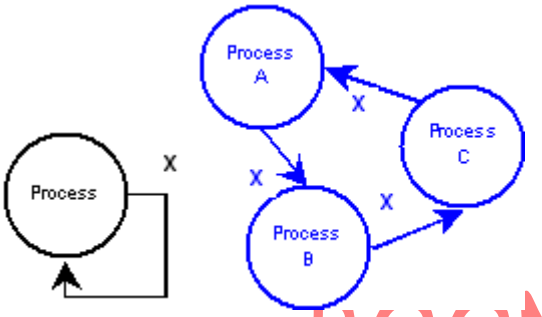
14. A fork in a data flow means that exactly the same data goes from a common location to two or more different processes, data stores, or sources/sinks. (This usually indicates different copies of the same data going to different locations.)



15. A join in a data flow means that exactly the same data comes from any of two or more different processes, data stores, or sources/sinks, to a common location.



16. A data flow cannot go directly back to the same process it leaves. There must be at least one other process that handles the data flow, produces some other data flow, and returns the original data flow to the originating process.



17. A data flow to a data store means update (i.e., delete, add, or change).
18. A data flow from a data store means retrieve or use.
19. A data flow has a noun phrase label. More than one data flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.

Why DFD?

DFD graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment and between components of a system. The visual representation makes it a good communication tool between User and System designer. Structure of DFD allows starting from a broad overview and expand it to a hierarchy of detailed diagrams. DFD has often been used due to the following reasons:

- Logical information flow of the system
- Determination of physical system construction requirements

- Simplicity of notation
- Establishment of manual and automated systems requirements

7.1.6 The Process of Logical Process Modelling

Logical vs Physical Data Flow Diagrams

Data flow diagrams are categorized as either logical or physical. A logical data flow diagram focuses on the business and how the business operates. It is not concerned with how the system will be constructed. We can ignore implementation specifics such as, computer configuration, data storage technology, communication or message passing methods by focusing on the functions performed by the system, such as, data collection, data to information transformation and information reporting.

A physical data flow diagram shows how the system will be implemented, including the hardware, software, files, and people in the system. It is developed such that the processes described in the logical data flow diagrams are implemented correctly to achieve the goal of the business.

Benefits of Logical Data Flow Diagram

- A logical diagram is drawn present business information and centered on business activities, which makes it an ideal communication tool when use in communicating with project users.
- Logical DFD is based on business events and independent of particular technology or physical arrangement, which makes the resulting system more stable.
- Logical DFD allows analyst to understand the business being studied and to identify the reason behind implementation plans.
- Systems implemented based on logical DFD will be easier to maintain because business functions are not subject to frequent change.
- Very often, logical DFD does not contain data stores other than files or a database, making less complex than physical DFD and is easier to develop.
- Physical DFD can be easily formed by modifying a logical DFD.

Benefits of Physical Data Flow Diagram

- Clarifying which processes are manual and which are automated: Manual processes require detailed documentation and automated process require computer programs to be developed.
- Describing processes in more detail than do logical DFDs: Describes all steps for processing of data.
- Sequencing processes that have to be done in a particular order: Sequence of activities that lead to a meaningful result are described. For example, update must be performed before a producing a summary report.
- Identifying temporary data storage: Temporary storage such as a sales transaction file for a customer receipt (report) in a grocery store, are described.
- Specifying actual names of files and printouts: Logical data flow diagrams describes actual filenames and reports, so that the programmers can relate those with the data dictionary during the developmental phase of the system.
- Adding controls to ensure the processes are done properly: These are conditions or validations of data that are to be met during input, update, delete, and other processing of data.

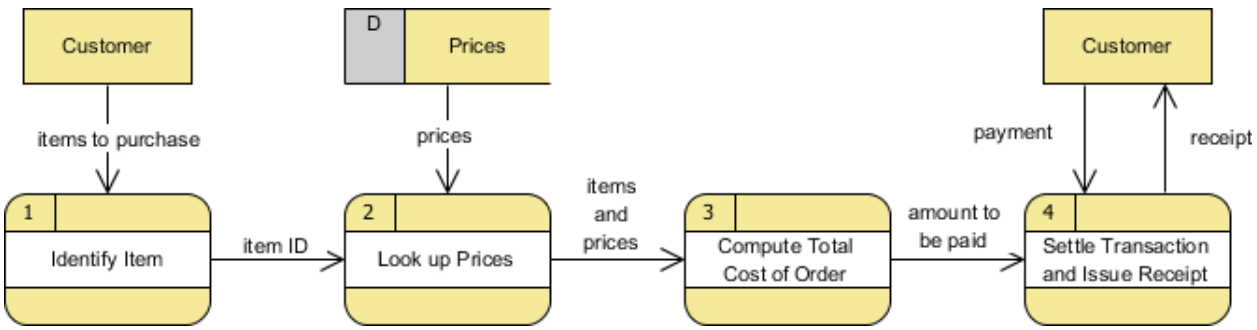
Refining Physical DFD for Logical DFD

The example below shows a logical DFD and a physical DFD for a grocery store cashier:

- The CUSTOMER brings the ITEMS to the register;
- PRICES for all ITEMS are LOOKED UP, and then totaled;
- Next, PAYMENT is given to the cashier finally, the CUSTOMER is given a receipt.

Logical DFD Example - Grocery Store

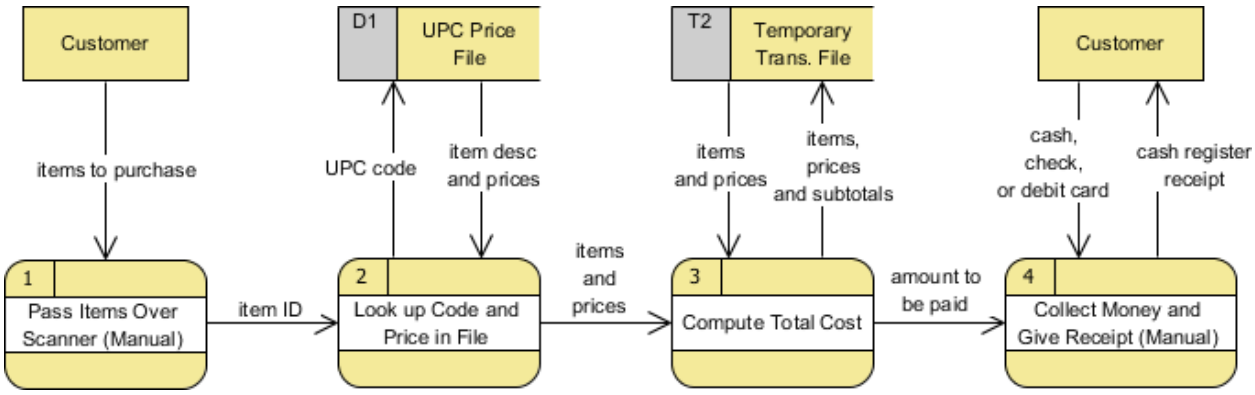
The logical DFD illustrates the processes involved without going into detail about the physical implementation of activities



Physical DFD Example - Grocery Store

The physical DFD shows that a bar code-the UPC PRICE code found on most grocery store items is used

- In addition, the physical DFD mentions manual processes such as scanning, explains that a temporary file is used to keep a subtotal of items
- The PAYMENT could be made by CASH, CHECK, or DEBIT CARD
- Finally, it refers to the receipt by its name, CASH REGISTER RECEIPT



7.2. Decomposition of DFD: Context dataflow diagram, Functional Decomposition Diagram, Level-1 DFD, Level-2 DFD, Level-n DFD, Guidelines for Drawing DFD

7.2.1 Decomposition of DFD:

To specify what a high-level process does, break it down into smaller units in more DFDs. A high-level process is an entire DFD. Each high-level process is decomposed into other processes with data flows and data stores. Each decomposition is a DFD in itself. You can continue to break down processes until you reach a level on which further decomposition seems impossible or meaningless.

The data flows of the opened process are connected in the new diagram to the process related to the opened process. Vertices, and the flows and objects connected to them, are transferred with the flows that are connected to the decomposed process.

7.2.2 Context dataflow diagram,

Context Diagram

A data flow diagram (DFD) of the scope of an organizational system that shows the system boundaries, external entities that interact with the system and the major information flows between the entities and the system

Context Diagram. A context diagram is a top level (also known as "Level 0") data flow diagram. It only contains one process node ("Process 0") that generalizes the function of the entire system in relationship to external entities. DFD Layers. Draw data flow diagrams can be made in several nested layers.

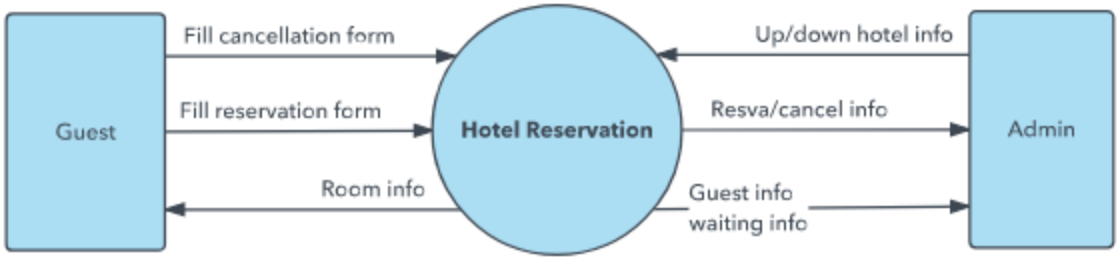
7.2.3 Functional Decomposition Diagram,

- Functional decomposition
  - Act of going from one single system to many component processes
  - Repetitive procedure
- Level-N Diagrams
  - A DFD that is the result of *n* nested decompositions of a series of subprocesses from a process on a level-0 diagram

DFD levels and layers: From context diagrams to pseudocode

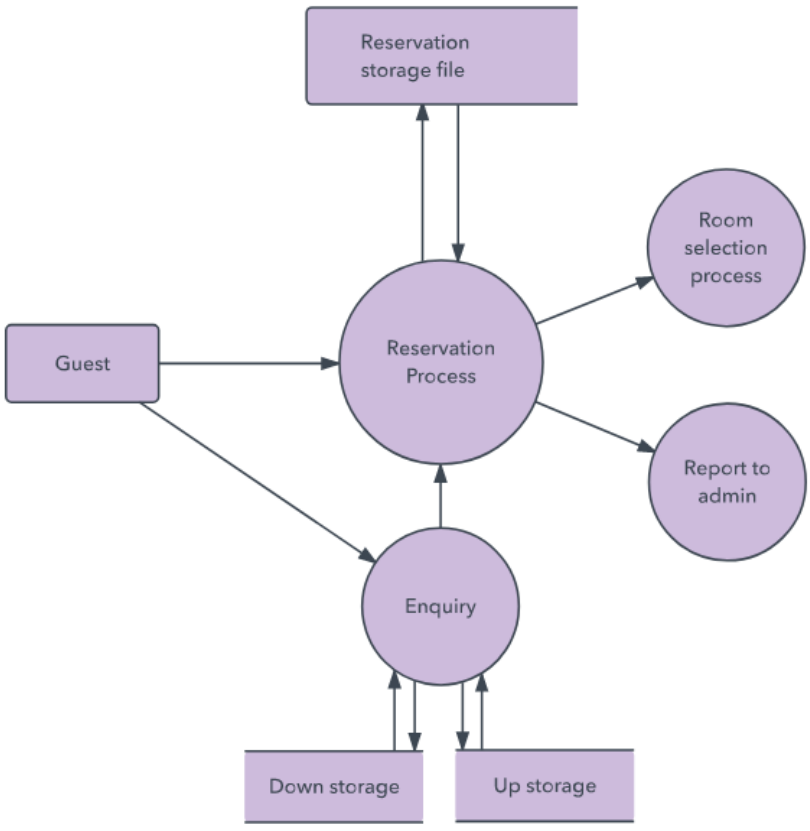
A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond. The necessary level of detail depends on the scope of what you are trying to accomplish.

DFD Level 0 is also called a Context Diagram. It's a basic overview of the whole system or process being analyzed or modeled. It's designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.



7.2.4 Level-1 DFD,

DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its sub processes.

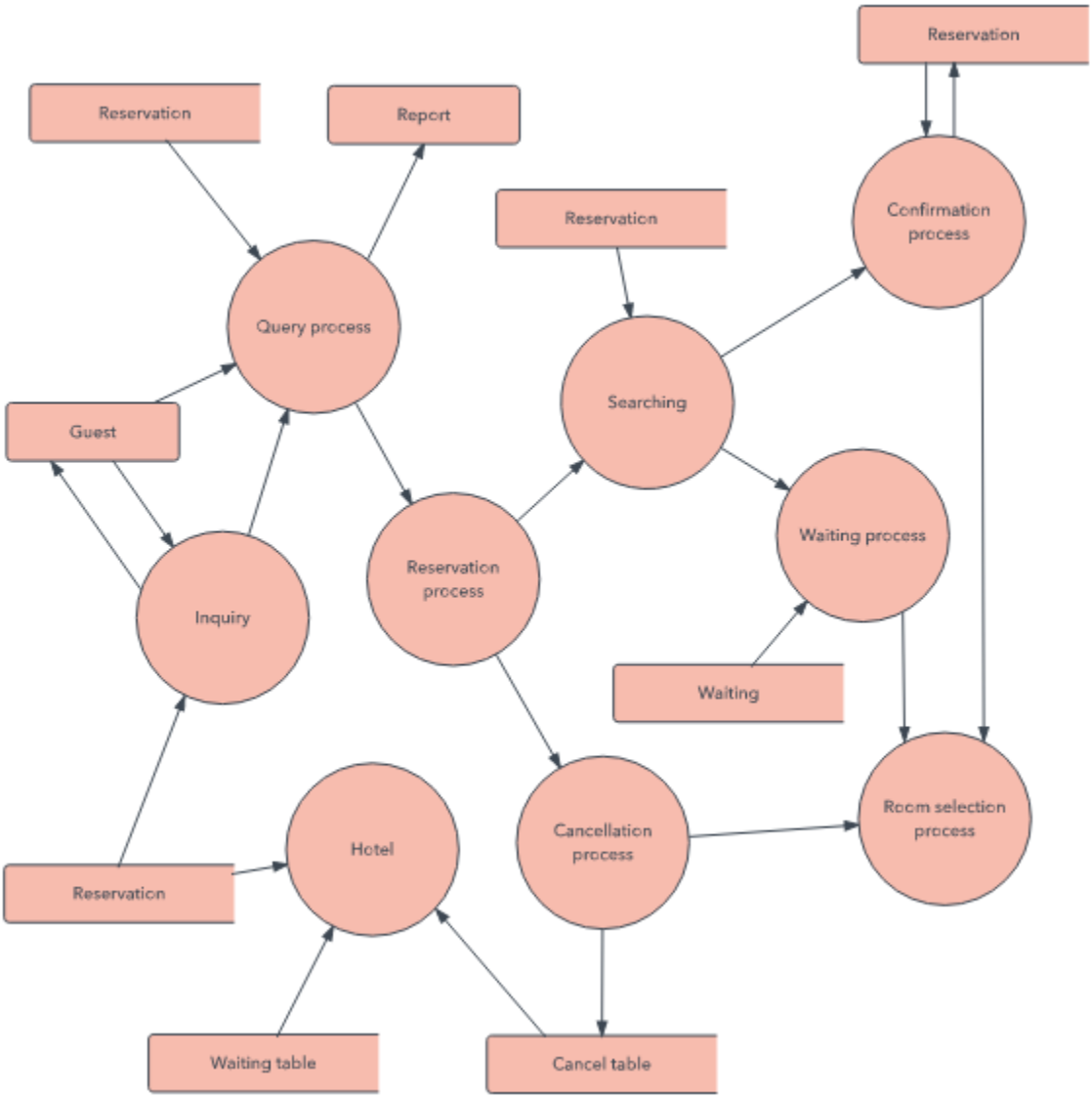


7.2.5 Level-2 DFD,

DFD Level 2 then goes one step deeper into parts of Level 1. It may require more text to reach the necessary level of detail about the system's functioning.

Progression to Levels 3, 4 and beyond is possible, but going beyond Level 3 is uncommon. Doing so can create complexity that makes it difficult to communicate, compare or model effectively.
Using DFD layers, the cascading levels can be nested directly in the diagram, providing a cleaner look with easy access to the deeper dive.

By becoming sufficiently detailed in the DFD, developers and designers can use it to write pseudocode, which is a combination of English and the coding language. Pseudocode facilitates the development of the actual code.



7.2.6 Level-n DFD,

7.2.7 Guidelines for Drawing DFD

1. Each process must have a minimum of one data flow going into it and one data flow leaving it.
2. Each data store must have at least one data flow going into it and one data flow leaving it.
3. A data flow out of a process should have some relevance to one or more of the data flows into a process.
4. Data stored in a system must go through a process.
5. Filing systems within an organisation cannot logically communicate with one another unless there is a process involved.
6. All processes in DFD must be linked to either another process or a data store.

https://www.dlsweb.rmit.edu.au/toolbox/ecommerce/tbn_respak/tbn_e2/html/tbn_e2_devsol/dfds/dfdrules.htm

7.3. Logic Modelling: Structured English & Decision Tables

Decision Trees, Decision Tables, and Structured English

Decision Trees, Decision Tables, and Structured English are tools used to represent process logic.

The two building blocks of Structured English are (1) structured logic or instructions organized into nested or grouped procedures, and (2) simple English statements such as add, multiply, move, etc. (strong, active, specific verbs)

Five conventions to follow when using Structured English:

1. Express all logic in terms of sequential structures, decision structures, or iterations.
2. Use and capitalize accepted keywords such as: IF, THEN, ELSE, DO, DO WHILE, DO UNTIL, PERFORM
3. Indent blocks of statements to show their hierarchy (nesting) clearly.
4. When words or phrases have been defined in the Data Dictionary, underline those words or phrases to indicate that they have a specialized, reserved meaning.
5. Be careful when using "and" and "or" as well as "greater than" and "greater than or equal to" and other logical comparisons.

Four major steps in building Decision Trees:

1. Identify the conditions
2. Identify the outcomes (condition alternatives) for each decision
3. Identify the actions

4. Identify the rules.

   Decision Tables are useful when complex combinations of conditions, actions, and rules are found or you require a method that effectively avoids impossible situations, redundancies, and contradictions. Decision Trees are useful when the sequence of conditions and actions is critical or not every condition is relevant to every action.

Three Basic Computer Programming Constructs:

1. Sequence
2. Selection (If-Else Conditional)
3. Iteration (Loops)

Decision Table - Jacket Weather

Determine whether you need to wear a light jacket, heavy jacket, or no jacket at all when you leave the house. You need a heavy jacket if the temperature is below 50 degrees. You can make do with a light jacket if the temperature is between 50 and 70. At 70 and above you only need a light jacket if there is precipitation.

There will be 6 rules. The first condition (Is there precipitation?) has two possible outcomes, yes or no. The second condition (temperature) has three possible outcomes. Two times three is 6.

Conditions	1	2	3	4	5	6
1. Precipitation?	Y	Y	Y	N	N	N
2. Temperature	Less than 50	Between 50-70	More than 70	Less than 50	Between 50-70	More than 70

Actions						
1. Heavy Jacket	X			X		
2. Light Jacket		X	X		X	
3. No Jacket						X

Lab Work

Draw data flow diagrams of real world problems by using CASE Tools

The Food Ordering System Example

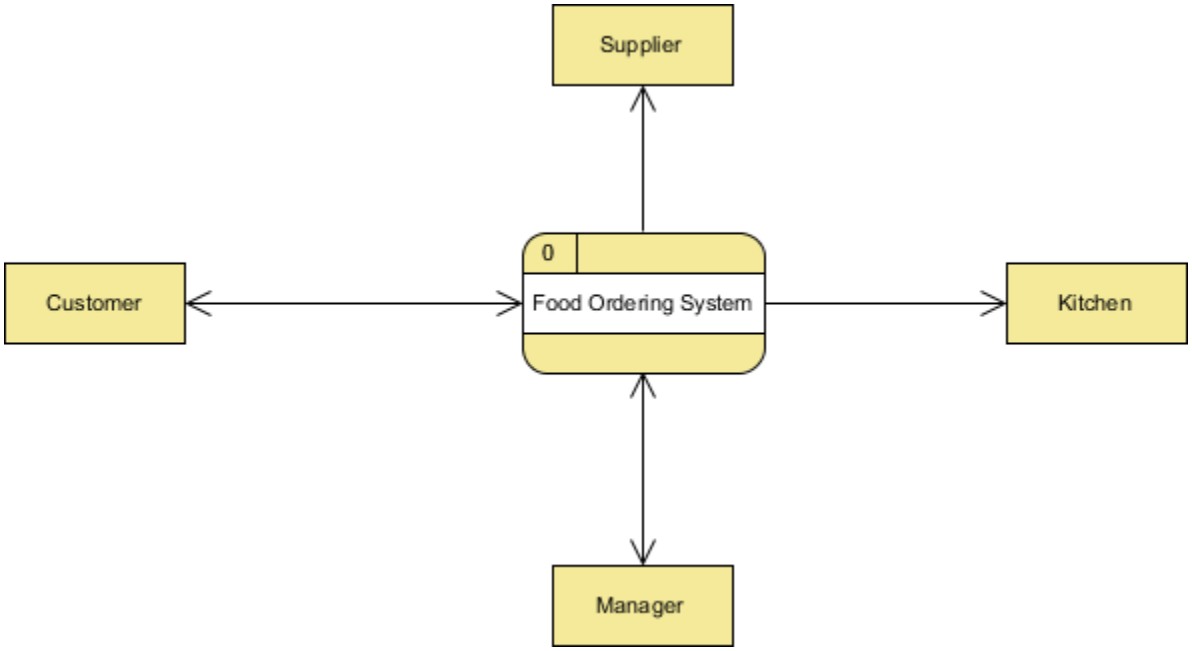
Context DFD

A context diagram is a data flow diagram that only shows the top level, otherwise known as Level 0. At this level, there is only one visible process node that represents the functions of a complete system in regards to how it interacts with external entities. Some of the benefits of a Context Diagram are:

1. Shows the overview of the boundaries of a system
2. No technical knowledge is required to understand with the simple notation
3. Simple to draw, amend and elaborate as its limited notation

The figure below shows a context Data Flow Diagram that is drawn for a Food Ordering System. It contains a process (shape) that represents the system to model, in this case, the "Food Ordering System". It also shows the participants who will interact with the system, called the external entities. In this example, Supplier, Kitchen, Manager and Customer

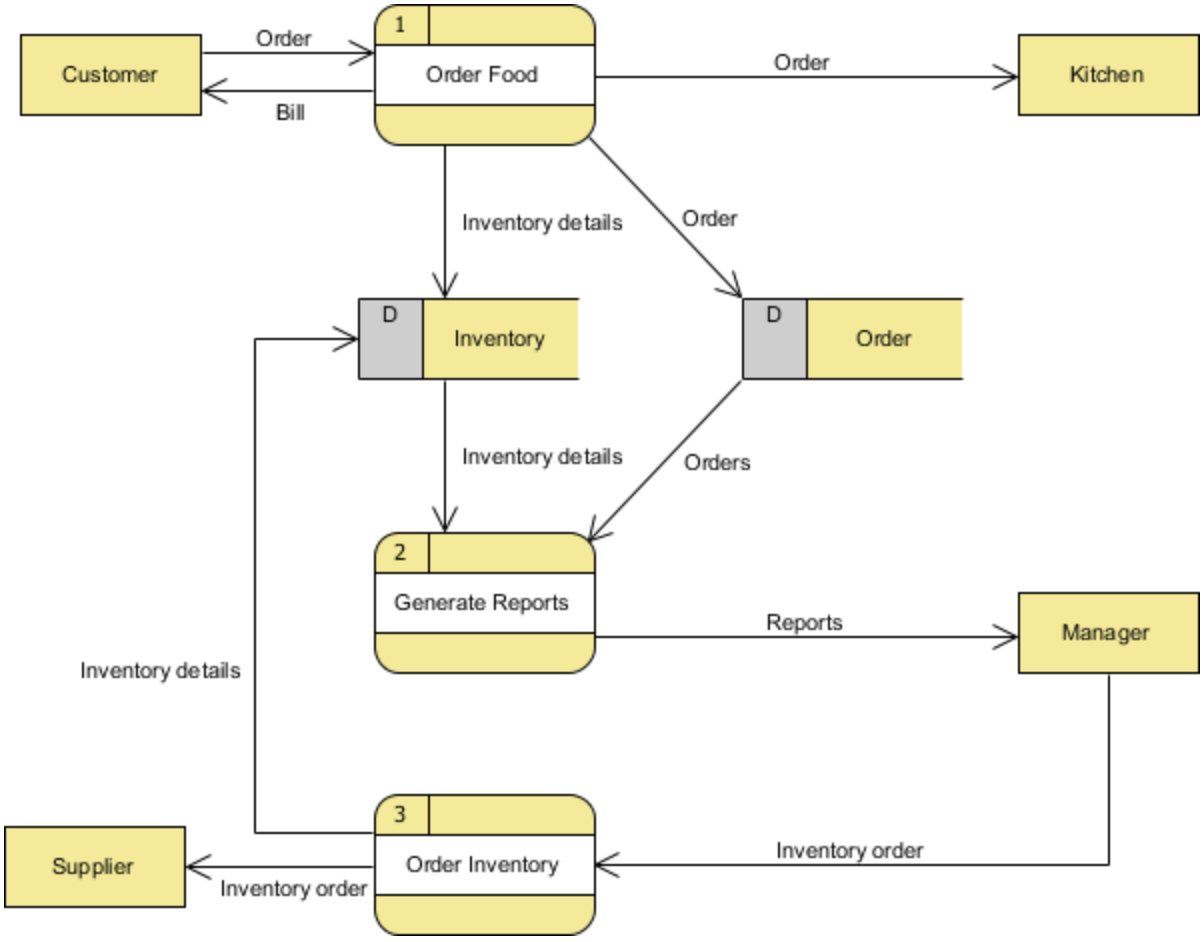
are the entities who will interact with the system. In between the process and the external entities, there are data flow (connectors) that indicate the existence of information exchange between the entities and the system.



Context DFD is the entrance of a data flow model. It contains one and only one process and does not show any data store.

Level 1 DFD

The figure below shows the level 1 DFD, which is the decomposition (i.e. break down) of the Food Ordering System process shown in the context DFD. Read through the diagram and then we will introduce some of the key concepts based on this diagram.



The Food Order System Data Flow Diagram example contains three processes, four external entities and two data stores.

Based on the diagram, we know that a Customer can place an Order. The Order Food process receives the Order, forwards it to the Kitchen, store it in the Order data store, and store the updated Inventory details in the Inventory data store. The process also deliver a Bill to the Customer.

Manager can receive Reports through the Generate Reports process, which takes Inventory details and Orders as input from the Inventory and Order data store respectively.

Manager can also initiate the Order Inventory process by providing Inventory order. The process forwards the Inventory order to the Supplier and stores the updated Inventory details in the Inventory data store.

Data Flow Diagram Tips and Cautions

Tips

1. Process labels should be verb phrases; data stores are represented by nouns

2. A data store must be associated to at least a process
3. An external entity must be associated to at least a process
4. Don't let it get too complex; normally 5 - 7 average people can manage processes
5. DFD is non-deterministic - The numbering does not necessarily indicate sequence, it's useful in identifying the processes when discussing with users
6. Data stores should not be connected to an external entity, otherwise, it would mean that you're giving an external entity direct access to your data files
7. Data flows should not exist between 2 external entities without going through a process
8. A process that has inputs but no outputs is considered to be a black-hole process

Cautions

Don't mix up data flow and process flow

Some designers may feel uncomfortable when seeing a connector connecting from a data store to a process, without seeing the step of data request being shown on the diagram somehow. Some of them will try to represent a request by adding a connector between a process and a data store, labeling it "a request" or "request for something", which is wrong.

Keep in mind that Data Flow Diagram was designed for representing the exchange of information. Connectors in a Data Flow Diagram are for representing data, not for representing process flow, step or anything else. When we label a data flow that ends at a data store "a request", this literally means we are passing a request as data into a data store. Although this may be the case in implementation level as some of the DBMS do support the use of functions, which intake some values as parameters and return a result, in Data Flow Diagram, we tend to treat data store as a sole data holder that does not possess any processing capability. If you want to model the system flow or process flow, use UML Activity Diagram or BPMN Business Process Diagram instead. If you want to model the internal structure of data store, use Entity Relationship Diagram.

Unit VI: System Implementation and Operation (12 Hrs)

6.1 System Construction and Implementation: The Construction Phase, The Implementation Phase, Testing: Unit, System and Regression Testing

6.1 System Construction and Implementation:

Construction software is the collection of programs, processes and information used to perform various tasks within the building or assembling of a structure or infrastructure as a means of increasing productivity, efficiency, and competitiveness. Tasks previously administered by a project manager, construction manager, design engineer, construction engineer and project architect can be performed by construction software applications.

Implementation is a process of ensuring that the information system is operational. It involves −

- Constructing a new system from scratch
- Constructing a new system from the existing one.

Implementation allows the users to take over its operation for use and evaluation. It involves training the users to handle the system and plan for a smooth conversion.

6.1.1 The Construction Phase,

The System Development Life Cycle (SDLC) is a series of six steps that a project team works through in order to conceptualize, analyze, design, construct and implement a new information technology system. Adhering to a SDLC increases efficiency and accuracy and reduces the risk of product failure.

1. **Planning**

During the planning phase, the objective of the project is determined and the requirements to produce the product are considered. An estimate of resources, such as personnel and costs, is prepared, along with a concept for the new product. All of the information is analyzed to see if there is an alternative solution to creating a new product. If there is no other viable alternative, the information is assembled into a project plan and presented to management for approval.

2. **Analysis**

During the analysis stage the project team determines the end-user requirements. Often this is done with the assistance of client focus groups, which provide an explanation of their needs and what their expectations are for the finished product and how it will perform. The project team documents all of the user requirements and gets a sign-off from the client and management to move forward with system design.

3. **Design**

The design phase is the "architectural" phase of system design. The flow of data processing is developed into charts, and the project team determines the most logical design and structure for data flow and storage. For the user interface, the project team designs mock-up screen layouts that the developers uses to write the code for the actual interface.

4. **Construction**

During the construction phase developers execute the plans laid out in the design phase. The developers design the database, generate the code for the data flow process and design the actual user interface screens. During the construction phase, test data is prepared and processed as many times as necessary to refine the code.

5. **Test**

During the test phase all aspects of the system are tested for functionality and

performance. The system is tested for integration with other products as well as any previous versions with which it needs to communicate. Essentially, the key elements of the testing phase are to verify that the system contains all the end user requirements laid out in the analysis phase, that all the functions are accurately processing data, that the new system works with all other systems or prior systems, and that the new system meets the quality standards of the company and the customers.

6. Rollout

The rollout phase is when customers receive the new system as an update or a full-scale conversion. Once rollout begins, feedback from clients begins, and the code is tweaked for any performance issues or the mishandling of data.

6.1.2 The Implementation Phase,

Systems implementation is the process of:

- defining how the information system should be built (i.e., physical system design),
- ensuring that the information system is operational and used,
- ensuring that the information system meets quality standard (i.e., quality assurance).

After having the user acceptance of the new system developed, the implementation phase begins. Implementation is the stage of a project during which theory is turned into practice. The major steps involved in this phase are:

- Acquisition and Installation of Hardware and Software
- Conversion
- User Training
- Documentation

The hardware and the relevant software required for running the system must be made fully operational before implementation. The conversion is also one of the most critical and expensive activities in the system development life cycle. The data from the old system needs to be converted to operate in the new format of the new system. The database needs to be setup with security and recovery procedures fully defined.

During this phase, all the programs of the system are loaded onto the user's computer. After loading the system, training of the user starts. Main topics of such type of training are:

- How to execute the package?
- How to enter the data?
- How to process the data (processing details)?
- How to take out the reports?

After the users are trained about the computerized system, working has to shift from manual to computerized working. The process is called **Changeover**. The following strategies are followed for changeover of the system.

1. **Direct Changeover:** This is the complete replacement of the old system by the new system. It is a risky approach and requires comprehensive system testing and training.
2. **Parallel run** : In parallel run both the systems, i.e., computerized and manual, are executed simultaneously for certain defined period. The same data is processed by both the systems. This strategy is less risky but more expensive because of the following facts:

- Manual results can be compared with the results of the computerized system.
- The operational work is doubled.
- Failure of the computerised system at the early stage does not affect the working of the organization, because the manual system continues to work, as it used to do.

(iii) Pilot run: In this type of run, the new system is run with the data from one or more of the previous periods for the whole or part of the system. The results are compared with the old

system results. It is less expensive and risky than parallel run approach. This strategy builds the confidence and the errors are traced easily without affecting the operations.

The documentation of the system is also one of the most important activity in the

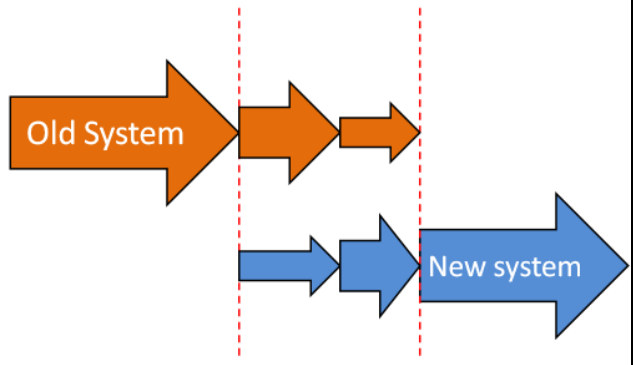
system development life cycle. This ensures the continuity of the system. Generally following two types of documentations are prepared for any system.

- User or Operator Documentation
- System Documentation

User Documentation: The user documentation is a complete description of the system from the user's point of view detailing how to use or operate the system. It also includes the major error messages likely to be encountered by the user.

System Documentation: The system documentation contains the details of system design, programs, their coding, system flow, data dictionary, process description, etc. This helps to understand the system and permit changes to be made in the existing system to satisfy new user needs.

Phased implementation is a method of changing from an existing system to a new one.



Phased implementation is a changeover process that takes place in stages.

As an example, think of a supermarket. In this supermarket the checkout system is being upgraded to a newer version. Imagine that only the checkout counters of the vegetable section are changed over to the new system, while the other counters carry on with the old system. If the new system does not work properly, it would not matter because only a small portion of the supermarket has been computerized. If it does work, staff can take turns working on the vegetable counters to get some practice using the new system.

After the vegetables section is working perfectly, the meat section might be next, then the confectionery section, and so on. Eventually all the various counters in the supermarket system would have been phased in, and everything would be running. This takes a long time as there are two systems working until the changeover is completed. However, the supermarket is never in danger of having to close and the staff are all able to get plenty of training in operating the new system, so it is a much friendlier method.

Other methods of system changeover include direct changeover and parallel running.

6.1.3 Unit Testing,

Unit testing is a confusing part of the software development process. Unit

testing involves individually testing unit of code separately to make sure that it works on its own, independent of the other units.

Unit testing is essentially a set of path, test performed to examine the several different paths through the modules. Unit testing is remarkably done by programmers with the help of Unit framework (like J Unit, CPP Unit etc. depending up on the language source code is written). Unit testing is usually an automated process and performed within the programmers IDE.

Unit testing is an action used to validate that separate units of source code remains working properly. Example: - A function, method, Loop or statement in program is working fine. It is executed by the Developer. In unit testing Individual functions or procedures are tested to make sure that they are operating correctly and all components are tested individually.

Unit testing is a strategy that utilizes the white-box method and concentrates on testing individual programming units. These units are sometimes specifying to as modules or atomic modules and they represent the smallest programming entity.

Unit Testing Example
Example of Unit testing is explain below

For example you are testing a function; whether loop or statement in a program is working properly or not than this is called as unit testing. A beneficial example of a framework that allows automated unit testing is JUNIT (a unit testing framework for java). XUnit [20] is a more general framework which supports other languages like C#, ASP, C++, Delphi and Python to name a few.

Tests that are performed during the unit testing are explained below:
1) Module Interface test: In module interface test, it is checked whether the information is properly flowing in to the program unit (or module) and properly happen out of it or not.

2) Local data structures: These are tested to inquiry if the local data within the module is stored properly or not.

3) Boundary conditions: It is observed that much software often fails at boundary related conditions. That's why boundary related conditions are always tested to make safe that the program is properly working at its boundary condition's.

4) Independent paths: All independent paths are tested to see that they are properly executing their task and terminating at the end of the program.

5) Error handling paths: These are tested to review if errors are handled properly by them or not.

Conclusion: At last we conclude that Unit testing focuses on the distinct modules of the product.

Unit testing is a software testing method by which individual units of source code, such as functions, methods, and class are tested to determine whether they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. Unit tests are short code fragments created by programmers during the development process. It forms the basis for component testing.

Unit testing can be done in the following two ways −

Manual Testing	Automated Testing
Executing the test cases manually without any tool support is known as manual testing. • Since test cases are executed by human resources so it is very **time consuming and tedious**.	Taking tool support and executing the test cases by using automation tool is known as automation testing. • Fast Automation runs test cases significantly faster than human resources. • The **investment over human resources is less** as test cases are

• As test cases need to be executed manually so more testers are required in manual testing. • It is less reliable as tests may not be performed with precision each time because of human errors. • No programming can be done to write sophisticated tests which fetch hidden information.	executed by using automation tool. • Automation tests perform precisely same operation each time they are run and **are more reliable**. • Testers **can program sophisticated tests** to bring out hidden information.

6.1.4 System and Regression Testing

Most of the Software applications which are developed are usually broken into many modules and given to different teams. These modules are then usually developed individually and later on integrated to form the complete software application. When two or more modules are combined and tested, it is called integration testing. After all the modules are combined and the complete system is made, testing of the whole system is known as System Testing.

Before going into the differences let us first understand each type of testing in brief :

What is System Testing?

System Testing is testing of the software application as a whole to check if the system is complaint with the user requirements. It is an end to end user perspective testing intended to find defects in the software system.

System Testing is a type of black box testing technique thus the knowledge of internal code in not required. It is a high level testing always performed after integration testing. Regression and Re Testing is performed many times in system testing. The user can perform different type of tests under System Testing .It would depend on the user or the organisation to choose which type of system testing should be performed on the application. System testing can be broadly classified in two

types:

1. Functional Testing
2. Non Functional Testing

System testing is performed to check the following points:

- To check whether the software system is made according to the customer needs written in Software Requirements Specifications, it meets both functional and non-functional design requirements of the system.
- When all the modules are combined as a whole, many errors and facts may arise which may not give the expected results? So system testing is performed to find the defects or bugs in all the interfaces as well the whole system.
- To execute the real –life scenarios on the software. It is done on the staging server which is very much similar to the production server where the software would actually be deployed. The system test cases are made according to the end- to – end use perspective.

What is Integration Testing?

Integration testing tests the interface between modules of the software application. The different modules are first testing individually and then combined to make a system. Testing the interface between the small units or modules is integration testing. It is usually conducted by software integration tester and in continuation to the development. There are different techniques available for integration testing:

1. Big Bang Integration Testing: In type of integration testing all the modules are combined first and then tested together.
2. Top Bottom Integration Testing: This type of testing take place from top to bottom uses Stubs which are substitutes of components. The top module is tested first.
3. Bottom to Top Integration Testing: This type of testing take from bottom to top and uses Drivers which are substitutes of components. The bottom module is tested first.

Objective of Integration Testing:

Integration testing is performed to check the following points:

- To check whether the modules developed by individual developers when combined are according to standards and gives the expected results.
- When modules are combined, sometimes the data travelling between modules

has many errors which may not give the expected results. So integration testing is performed to find the defects or bugs in all the interfaces.

- To check the integration between any third party tools used

Difference between System Testing and Integration Testing:

System Testing	Integration Testing
1. Testing the completed product to check if it meets the specification requirements.	1. Testing the collection and interface modules to check whether they give the expected result
2. Both functional and non-functional testing are covered like sanity, usability, performance, stress an load .	2.Only Functional testing is performed to check whether the two modules when combined give correct outcome.
3. It is a high level testing performed after integration testing	3. It is a low level testing performed after unit testing
4. It is a black box testing technique so no knowledge of internal structure or code is required	4. It is both black box and white box testing approach so it requires the knowledge of the two modules and the interface
5. It is performed by test engineers only	5. Integration testing is performed by developers as well test engineers
6. Here the testing is performed on the system as a whole including all the external interfaces, so any defect found in it is regarded as defect of whole system	6. Here the testing is performed on interface between individual module thus any defect found is only for individual modules and not the entire system
7. In System Testing the test cases are developed to simulate real life scenarios	7. Here the test cases are developed to simulate the interaction between the two module
8. The System testing covers many	8. Integration testing techniques

different testing types like sanity, usability, maintenance, regression, retesting and performance	includes big bang approach, top bottom , bottom to top and sandwich approach.

Example to differentiate between System testing and Integration Testing:

Project : Online Shopping

Modules: The Integration Testing would include checking the interaction between different modules like :

1. Searching a product using search bar, checking if the search bar accepts space, characters.
2. Checking the search results after hitting enter button, if it is according to the searched words. Changing the search item and checking the results again
3. Clicking on the product we select
4. Check the product detail page
5. Adding the product to the cart by clicking on "Add to Cart", check by adding few more items
6. Checking the shopping cart page, if all the data coming from the product detail page is proper.

System: The System Testing would include checking system as a whole:

1. Add items to the Product page from back-end, check if it stored correctly in database table.
2. Add few items to shopping cart and check if these items are correctly stored in temporary tables in database.
3. Delete few items from shopping cart, it should be deleted from DB as well.
4. Perform a check out transaction; see in DB if the payment option and other details are updated in DB as real time.
5. Checking the payment page and the transaction id generated. Checking the same in database.

6.2System Operation and Support:
Systems Development, Operation and Support Functions

Operational support systems are software programs and sometimes computer hardware applications that assist workplace actions and operate a telco's network, provision and sustain client services.

Visional Operational Support Systems - OSS Fulfillment An operational support system (OSS) is a confluence of computer applications or an IT system employed by communications services companies for overseeing, controlling, interpreting and managing a computer or telecom network program. OSS is historically utilized by network planners, services architects, operations, support, and engineering organizations in the services supplier. More and more product managers and senior personnel beneath the CTO or COO could also utilize or depend on Operational Support Services in many circumstances.

Systems Development

The systems development life cycle (SDLC), also referred to as the application development life-cycle, is a term used in systems engineering, information systems and software engineering to describe a process for planning, creating, testing, and deploying an information system.

Operation and Support Functions

The deployment of the system includes changes and enhancements before the decommissioning or sunset of the system. Maintaining the system is an important aspect of SDLC. As key personnel change positions in the organization, new changes will be implemented. There are two approaches to system development; there is the traditional approach (structured) and object oriented. Information Engineering includes the traditional system approach, which is also called the structured analysis and design technique. The object oriented approach views the information system as a collection of objects that are integrated with each other to make a full and complete information system.

The Systems Team supports the system and its end users as an integral part of the information system's day-to-day operations. The Operations or System Administration Manual defines tasks, activities, and responsible parties for these daily activities and must be updated as changes occur. By monitoring the system continuously, the Systems Team ensures that the production environment is fully functional and performs as specified. Critical support tasks and activities include:

1. Guarantee of required system availability
2. Implementation of non-emergency requests during scheduled outages
3. Ensuring the documentation in the operating procedures of all processes,

manual and
automated

4. Acquisition of critical system supplies (e.g., paper, cabling, toner, tapes, removable disks)
   before the supply is exhausted
5. Performance of routine backup and recovery procedures
6. Performance of physical security functions by ensuring all system staff and end users have
   the proper clearances and access privileges
7. Ensuring currency and testing of contingency planning for disaster recovery
8. Ensuring periodic training on current and new processes for end users
9. Ensuring monitoring and meeting of service level objectives while taking remedial actions
   for any deficiencies
10. Monitoring of performance measurements, statistics, and system logs, including monitoring
    and tuning network traffic.

The system log, the Operations Manual, journals, and other logs are invaluable in emergencies and should be kept in a central repository with other operational documentation.

The System Manager reviews the Program Trouble Reports, which document problems with the system through an automated system. The reports typically include:

- Date and time
- Issue reporter
- Problem description
- Apparent cause
- Assigned developer
- Resolution
- Test results

Other information may be included and depends on the reporting system. The reporting system should permit an audit of the entire process from problem identification to problem resolution and case closure.

6.3 Program/ System Maintenance, System recovery, System Enhancement

6.3.1 System Maintenance

The results obtained from the evaluation process help the organization to determine

whether its information systems are effective and efficient or otherwise. The process of monitoring, evaluating, and modifying of existing information systems to make required or desirable improvements may be termed as System Maintenance.

System maintenance is an ongoing activity, which covers a wide variety of activities, including removing program and design errors, updating documentation and test data and updating user support. For the purpose of convenience, maintenance may be categorized into three classes, namely:

i) Corrective Maintenance: This type of maintenance implies removing errors in a program, which might have crept in the system due to faulty design or wrong assumptions. Thus, in corrective maintenance, processing or performance failures are repaired.

ii) Adaptive Maintenance: In adaptive maintenance, program functions are changed to enable the information system to satisfy the information needs of the user. This type of maintenance may become necessary because of organizational changes which may include:

a) Change in the organizational procedures,
b) Change in organizational objectives, goals, policies, etc.
c) Change in forms,
d) Change in information needs of managers.
e) Change in system controls and security needs, etc.

iii)Perfective Maintenance: Perfective maintenance means adding new programs or modifying the existing programs to enhance the performance of the information system. This type of maintenance undertaken to respond to user's additional needs which may be due to the changes within or outside of the organization. Outside changes are primarily environmental changes, which may in the absence of system maintenance, render the information system ineffective and inefficient. These environmental changes include:

a) Changes in governmental policies, laws, etc.,
b) Economic and competitive conditions, and
c) New technology.

6.3.2 System recovery

6.3.3 System Enhancement

Maintenance means restoring something to its original conditions. Enhancement means adding, modifying the code to support the changes in the user specification. System maintenance conforms the system to its original requirements and

enhancement adds to system capability by incorporating new requirements.

Thus, maintenance changes the existing system, enhancement adds features to the existing system, and development replaces the existing system. It is an important part of system development that includes the activities which corrects errors in system design and implementation, updates the documents, and tests the data.

Maintenance Types

System maintenance can be classified into three types −

- **Corrective Maintenance** − Enables user to carry out the repairing and correcting leftover problems.
- **Adaptive Maintenance** − Enables user to replace the functions of the programs.
- **Perfective Maintenance** − Enables user to modify or enhance the programs according to the users' requirements and changing needs.

Lab Work

• Demonstrate unit and integration testing.
https://www.toptal.com/qa/how-to-write-testable-code-and-why-it-matters

7.1 Object Oriented Development Life Cycle, Unified Modelling Language

7.2 UML Diagrams: Use-Case Diagram, Class Diagram, Object Diagram, Interaction Diagrams: Sequence and Collaboration Diagram, State Diagram, Activity Diagram, Component Diagram, Deployment Diagram

7.3 Object Oriented Analysis: Requirement Analysis using Use Case Model, Conceptual Modeling

7.4 Object Oriented Design: Defining Interaction Diagrams, Defining Design Class Diagrams

Lab Work

- Draw UML diagram by using CASE Tools.

https://www.smartdraw.com/use-case-diagram/

https://creately.com/diagram-type/use-case

7.1 Object Oriented Development Life Cycle, Unified Modelling Language

Object Oriented Development Life Cycle

Object-oriented analysis and design (OOAD) is a popular technical approach for analyzing and designing an application, system, or business by applying object-oriented programming, as well as using visual modeling throughout the development life cycles to foster better stakeholder communication and product quality.

According to the popular guide Unified Process, OOAD in modern software engineering is best conducted in an iterative and incremental way. Iteration by iteration, the outputs of OOAD activities, analysis models for OOA and design models for OOD respectively, will be refined and evolve continuously driven by key factors like risks and business value.

The purpose of any analysis activity in the software life-cycle is to create a model of the system's functional requirements that is independent of implementation constraints.

The main difference between object-oriented analysis and other forms of analysis is that by the object-oriented approach we organize requirements around objects, which integrate both behaviors (processes) and states (data) modeled after real world objects that the system interacts with. In other or traditional analysis methodologies, the two aspects: processes and data are considered separately.

For example, data may be modeled by ER diagrams, and behaviors by flow charts or structure charts.

The primary tasks in object-oriented analysis (OOA) are:

1. Find the objects
2. Organize the objects
3. Describe how the objects interact
4. Define the behavior of the objects
5. Define the internals of the objects

Common models used in OOA are use cases and object models. Use cases describe scenarios for standard domain functions that the system must accomplish. Object models describe the names, class relations (e.g. Circle is a subclass of Shape), operations, and properties of the main objects. User-interface mockups or prototypes can also be created to help understanding.

Object-oriented design

During object-oriented design (OOD), a developer applies implementation constraints to the conceptual model produced in object-oriented analysis. Such constraints could include the hardware and software platforms, the performance requirements, persistent storage and transaction, usability of the system, and limitations imposed by budgets and time. Concepts in the analysis model which is technology independent, are mapped onto implementing classes and interfaces resulting in a model of the solution domain, i.e., a detailed description of how the system is to be built on concrete technologies.

Important topics during OOD also include the design of software architectures by applying architectural patterns and design patterns with object-oriented design principles.

Object-oriented modeling (OOM) is a common approach to modeling applications, systems, and business domains by using the object-oriented paradigm throughout the entire development life cycles. OOM is a main technique heavily used by both OOD and OOA activities in modern software engineering.

Object-oriented modeling typically divides into two aspects of work: the modeling of dynamic behaviors like business processes and use cases, and the modeling of

static structures like classes and components. OOA and OOD are the two distinct abstract levels (i.e. the analysis level and the design level) during OOM. The Unified Modeling Language (UML) and SysML are the two popular international standard languages used for object-oriented modeling.

The benefits of OOM are:

Efficient and effective communication

Users typically have difficulties in understanding comprehensive documents and programming language codes well. Visual model diagrams can be more understandable and can allow users and stakeholders to give developers feedback on the appropriate requirements and structure of the system. A key goal of the object-oriented approach is to decrease the "semantic gap" between the system and the real world, and to have the system be constructed using terminology that is almost the same as the stakeholders use in everyday business. Object-oriented modeling is an essential tool to facilitate this.

Useful and stable abstraction

Modeling helps coding. A goal of most modern software methodologies is to first address "what" questions and then address "how" questions, i.e. first determine the functionality the system is to provide without consideration of implementation constraints, and then consider how to make specific solutions to these abstract requirements, and refine them into detailed designs and codes by constraints such as technology and budget. Object-oriented modeling enables this by producing abstract and accessible descriptions of both system requirements and designs, i.e. models that define their essential structures and behaviors like processes and objects, which are important and valuable development assets with higher abstraction levels above concrete and complex source code.

Object – Oriented Systems Development Life Cycle:-

Introduction:

The essence of the software Development Process that consists of analysis, design, implementation, testing, and refinement is to transform users' needs into a software solution that satisfy those needs.

The Software Development Process:

System development can be viewed as a process. Within the process, it is possible to replace one sub process has the same interface as the old one to allow it to fit into the process as a whole.

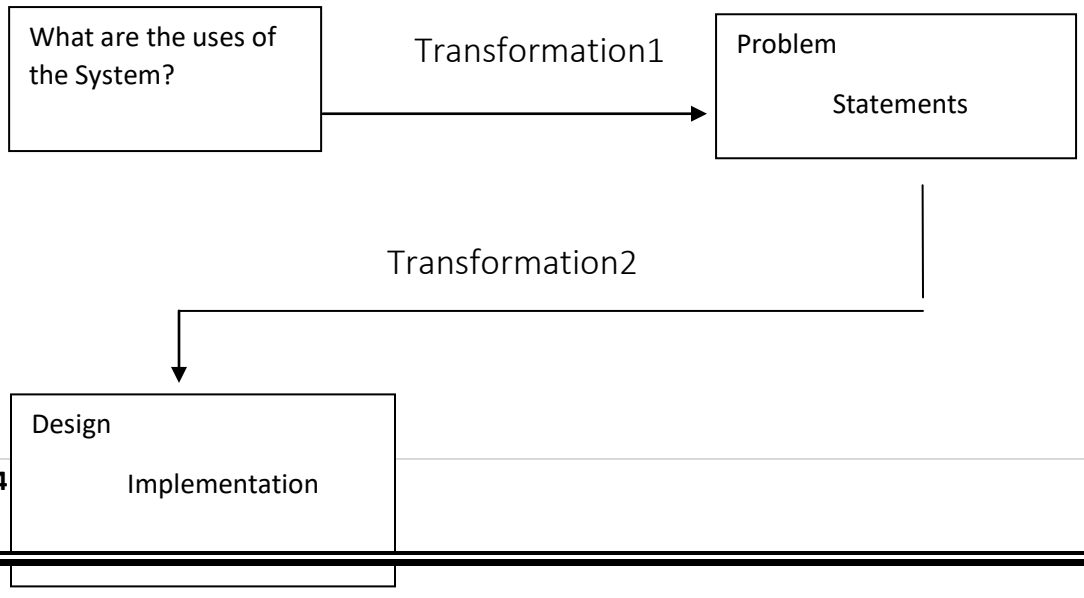
For example, Object oriented approach

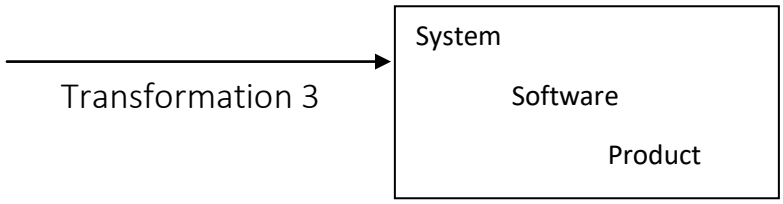
The process can be divided into small, interacting phases- sub processes. The sub processes must be defined in such a way that they are clearly spelled out; to allow each activity to be performed as independently of other sub processes as possible. Each sub process must have the following:

- A description in terms of how it works.
- Specification of the input required for the process.
- Specification of the output to be produced.

The software development process also can be divided into smaller, interacting subprocesses. Generally, the software development process can be viewed as a series of transformations, where the output of one transformation becomes the input of the subsequent transformation.

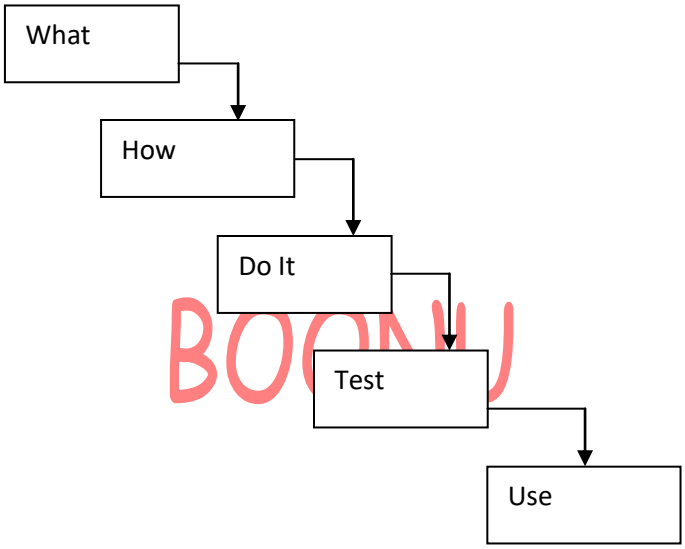
- **Transformation1 (Analysis):** It translates the users need into the System requirements and responsibilities. The way they use the system can provide insight into the user's requirements.
- **Tranformation2 (Design):** It begins with a problem statement and ends with a detailed design that can be transformed into an OS. This transformation includes the bulk of the software development activity, including the definition of how to build the software, its development, and it's testing.





- Transformation3 (Implementation) refines the detailed design into the system deployment that will satisfy the user needs. This takes into account the equipment, procedures, peoples and the like. It represents embedding the software product within its operational environment.

An example of the software development process is the waterfall approach, which starts with deciding what is to be done (what is the problem).



The waterfall model is the best way to manage a project with a well understood product, especially very large products.

Building high Quality software:-

The software process transforms the users' needs via the application domain to a software solution that satisfies those needs. Once the system exists, we must test it to see if is free of bugs.

To achieve high quality in software we need to be able to answer the following questions:

- How do we determine when the system is ready for delivery?
- Is it now an operational system that satisfies users' needs?

- Is it correct and operating as we thought it should?
- Does it pass an evaluation process?

There are four quality measures: Correspondence, Correctness, Verification and validation.

Correspondence measures how well the delivered system matches the needs of the operational environment, as described in the original requirements statements.

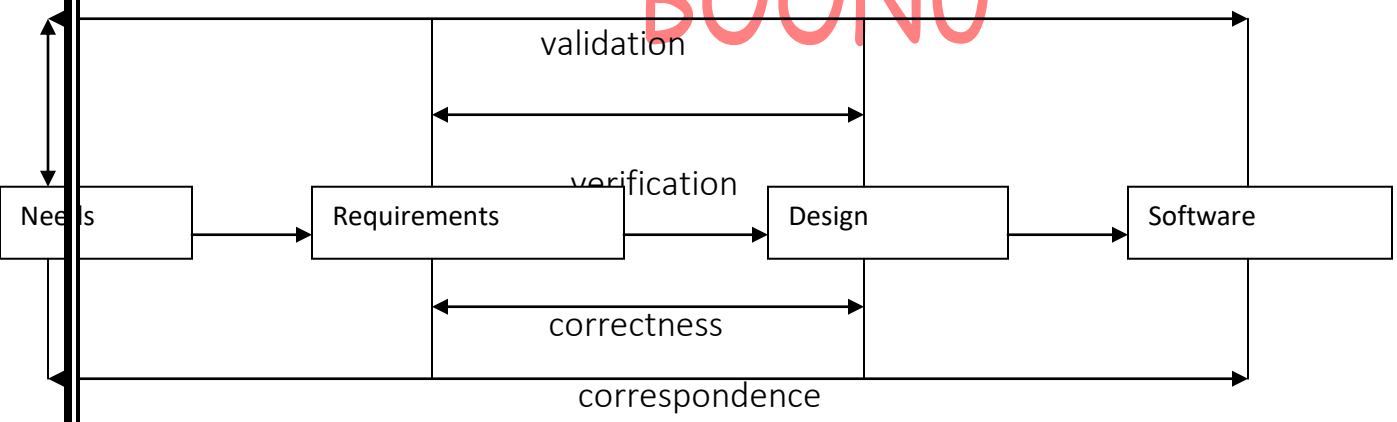
Validation is the task of predicting correspondence.

Correctness measures the consistency of the product requirements with respect to the design specification.

Verification is the exercise of determining correctness.

Verification and validation, answer the following question:

- *Verification: Am I building the product right?*
- *Validation: Am I building the right product?*



OBJECT ORIENTED SYSTEM DEVELOPMENT: A USECASE DRIVEN APPROACH:

The object-oriented **software development life cycle** (SDLC) consists of three macro processes: Object-oriented analysis, Object-oriented design, and object-oriented implementation.

Object-Oriented system development includes these activities:

- Object-oriented analysis-use case driven
- Object- Oriented design
- Prototyping
- Component- based development
- Incremental Testing

Object-Oriented Analysis--Use-case Driven:

The object-oriented Analysis phase of software development is concerned with determining the system requirements and identifying classes and their relationship to other classes in the problem domain. To understand the system requirements, we need to identify the users or the actors.

The intersection among objects' roles to achieve a given goal is called **collaboration.**

Object-Oriented Design:-

The goal of Object-Oriented design (OOD) is to design the classes identified during the analysis phase and the user interface. During this phase, we identify and define additional objects and classes that support implementation of the requirements.

First, build the object model based on objects and their relationships, then iterate and refine the model:

- Design and refine classes.
- Design and refine attributes.
- Design and refine methods.
- Design and refine Structures.
- Design and refine Associations.

Prototyping:-

Prototypes have been categorized in various ways. They are

- A **horizontal prototype** is a simulation of the interface but contains no functionality. This has the advantages of being very quick to implement, providing a good overall feel of the system.
- A **vertical prototype** is a subset of the system features with complete functionality. The principal advantage of this method is that the few implemented functions can be tested in great depth.
- An **analysis prototype** is an aid for the incremental development of the ultimate software solution.

The purpose of this review is threefold:

1. To demonstrate that the prototype has been developed according to the specification and that the final specification is appropriate.
2. To collect information about errors or other problems in the system, such as user interface problems that need to be addressed in the intermediate prototype stage.
3. To give management and everyone connected with the project the first (or it could be second or third)

Implementation: concept- Based Development

Component-based manufacturing makes products available to the marketplace that otherwise would be prohibitively expensive.

For example, computer-aided software engineering (CASE) tools.

Component-based development (CBD) is an industrialized approach to the software development process.

Rapid application development (RAD) is a set of tools and techniques that can be used to build an application faster than typically possible with traditional method. The terms often is used in conjuction with software prototyping.

Reusability:-

A major benefit of object-oriented system development is reusability, and this is the most difficult promise to deliver on. For an object to be really reusable, much more effect must be spent designing it.

Software components for reuse by asking the following questions:

- Has my problem already solved?
- Has my problem been partially solved?
- What has been done before to solve a problem similar to this one?

The reuse strategy can be based on the following:

- Information hiding (encapsulation).
- Conformance to naming standards.
- Creation and administration of an object repository.

- Encouragement by strategic management of reuse as opposed to constant development.
- Establishing targets for a percentage of the objects in the project to be reused.

Unified Modelling Language

The Unified Modeling Language is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.

UML, short for Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software. In this article we will give you detailed ideas about what is UML, the history of UML and a description of each UML diagram type, along with UML examples.

Why UML?

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural

problems. The Unified Modeling Language (UML) was designed to respond to these needs. The primary goals in the design of the UML summarize by Page-Jones in Fundamental Object-Oriented Design in UML as follows:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

7.2 UML Diagrams: Use-Case Diagram, Class Diagram, Object Diagram, Interaction Diagrams: Sequence and Collaboration Diagram, State Diagram, Activity Diagram, Component Diagram, Deployment Diagram

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

UML - An Overview

Before we begin to look at the theory of the UML, we are going to take a very brief run through some of the major concepts of the UML.

The first thing to notice about the UML is that there are a lot of different diagrams (models) to get used to. The reason for this is that it is possible to look at a system from many different viewpoints. A software development will have many stakeholders playing a part.

For Example:

- Analysts
- Designers
- Coders
- Testers
- QA
- The Customer

- Technical Authors

All of these people are interested in different aspects of the system, and each of them require a different level of detail. For example, a coder needs to understand the design of the system and be able to convert the design to a low level code. By contrast, a technical writer is interested in the behavior of the system as a whole, and needs to understand how the product functions. The UML attempts to provide a language so expressive that all stakeholders can benefit from at least one UML diagram.

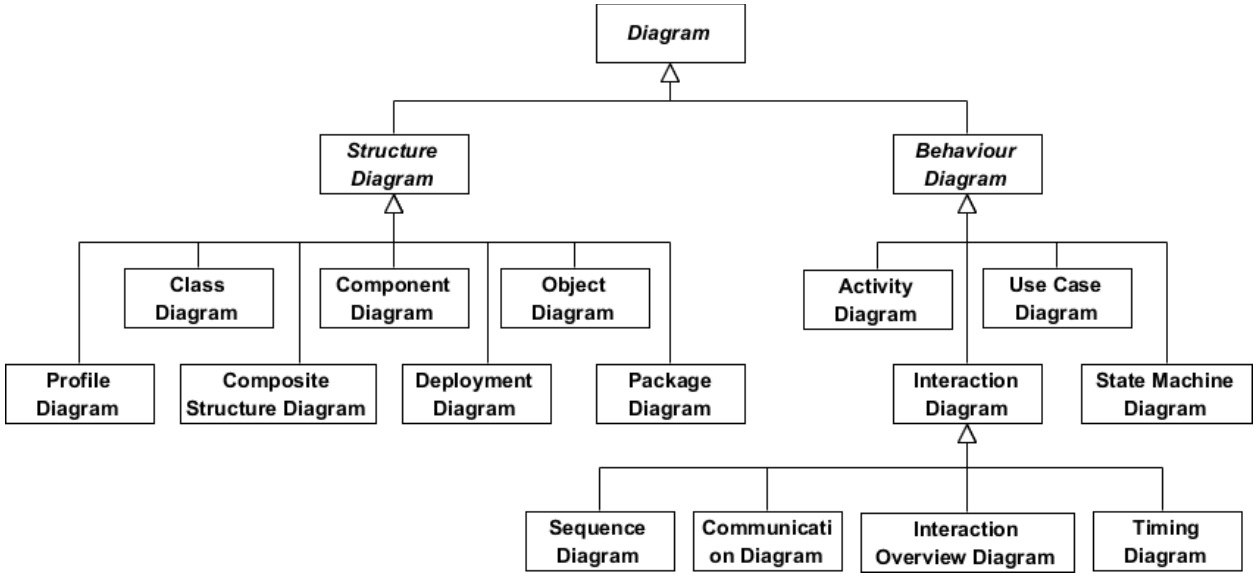
Here's a quick look at each one of these 13 diagrams in as shown in the UML 2 Diagram Structure below:

Structure diagrams show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other. The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts, there are seven types of structure diagram as follows:

1. Class Diagram
2. Component Diagram
3. Deployment Diagram
4. Object Diagram
5. Package Diagram
6. Composite Structure Diagram
7. Profile Diagram

Behavior diagrams show the dynamic behavior of the objects in a system, which can be described as a series of changes to the system over time, there are seven types of behavior diagrams as follows:

1. Use Case Diagram
2. Activity Diagram
3. State Machine Diagram
4. Sequence Diagram
5. Communication Diagram
6. Interaction Overview Diagram
7. Timing Diagram



7.2.1 Use-Case Diagram

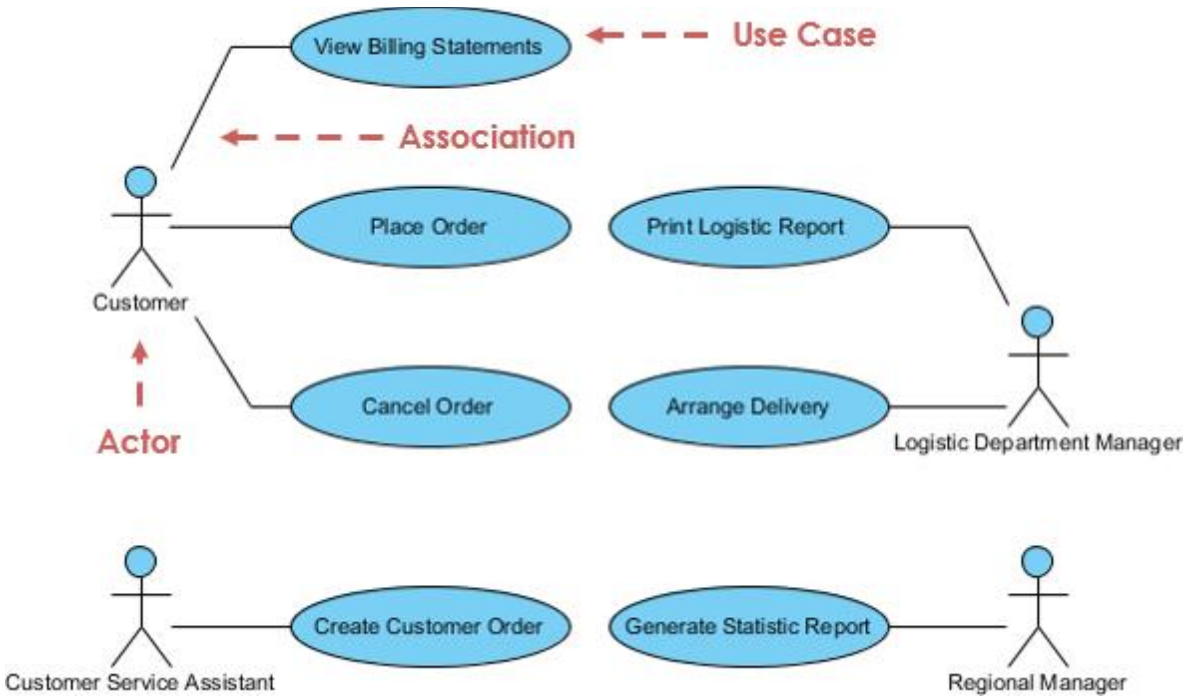
What is a Use Case Diagram?

A use-case model describes a system's functional requirements in terms of use cases. It is a model of the system's intended functionality (use cases) and its environment (actors). Use cases enable you to relate what you need from a system to how the system delivers on those needs.

Think of a use-case model as a menu, much like the menu you'd find in a restaurant. By looking at the menu, you know what's available to you, the individual dishes as well as their prices. You also know what kind of cuisine the restaurant serves: Italian, Mexican, Chinese, and so on. By looking at the menu, you get an overall impression of the dining experience that awaits you in that restaurant. The menu, in effect, "models" the restaurant's behavior.

Because it is a very powerful planning instrument, the use-case model is generally used in all phases of the development cycle by all team members.

Example of case diagram:



7.2.2 Class Diagram

What is a Class Diagram?

The class diagram is a central modeling technique that runs through nearly all object-oriented methods. This diagram describes the types of objects in the system and various kinds of static relationships which exist between them.

Relationships

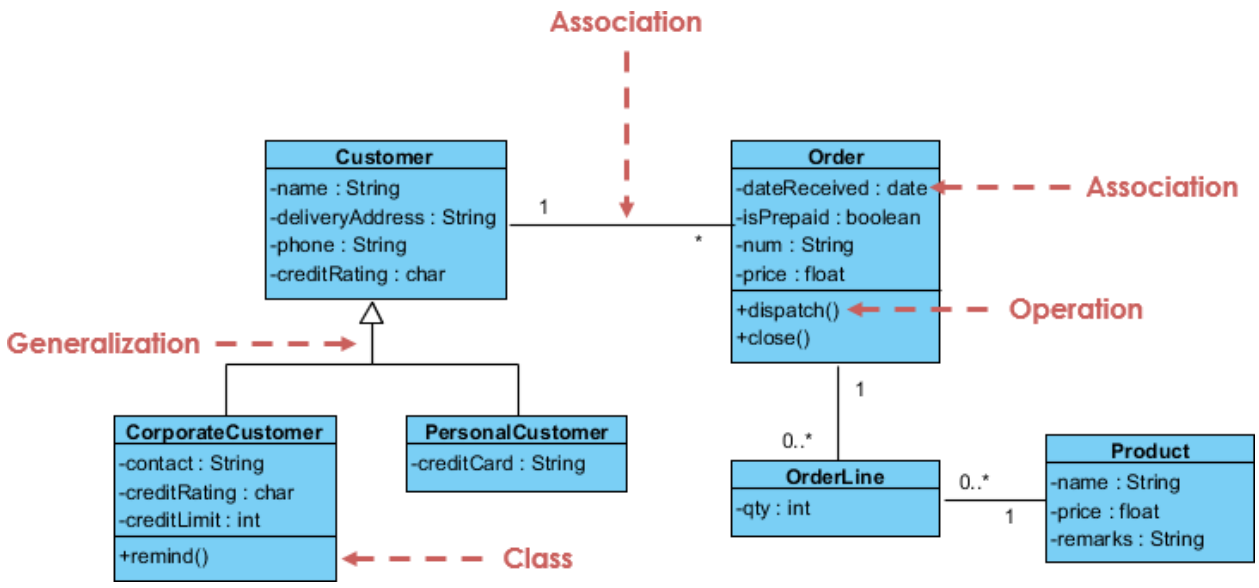
There are three principal kinds of relationships which are important:

Association - represent relationships between instances of types (a person works for a company, a company has a number of offices.

Inheritance - the most obvious addition to ER diagrams for use in OO. It has an immediate correspondence to inheritance in OO design.

Aggregation - Aggregation, a form of object composition in object-oriented design.

Class Diagram Example



7.2.3 Object Diagram

What is an Object Diagram?

An object diagram is a graph of instances, including objects and data values. A static object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time. The difference is that a class diagram represents an abstract model consisting of classes and their relationships. However, an object diagram represents an instance at a particular moment, which is concrete in nature. The use of object diagrams is fairly limited, namely to show examples of data structure.

Class Diagram vs Object Diagram - An Example

Some people may find it difficult to understand the difference between a UML Class Diagram and a UML Object Diagram as they both comprise of named "rectangle blocks", with attributes in them, and with linkages in between, which make the two UML diagrams look similar. Some people may even think they are the same because in the UML tool they use both the notations for Class Diagram and Object Diagram are put inside the same diagram editor - Class Diagram.

But in fact, Class Diagram and Object Diagram represent two different aspects of a code base. In this article, we will provide you with some ideas about these two UML diagrams, what they are, what are their differences and when to use each of them.

Relationship between Class Diagram and Object Diagram

You create "classes" when you are programming. For example, in an online banking system you may create classes like 'User', 'Account', 'Transaction', etc. In a classroom

management system you may create classes like 'Teacher', 'Student', 'Assignment', etc. In each class, there are attributes and operations that represent the characteristic and behavior of the class. Class Diagram is a UML diagram where you can visualize those classes, along with their attributes, operations and the inter-relationship.

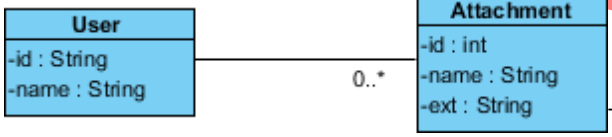
UML Object Diagram shows how object instances in your system are interacting with each other at a particular state. It also represents the data values of those objects at that state. In other words, a UML Object Diagram can be seen as a representation of how classes (drawn in UML Class Diagram) are utilized at a particular state.

If you are not a fan of those definition stuff, take a look at the following UML diagram examples. I believe that you will understand their differences in seconds.

Class Diagram Example

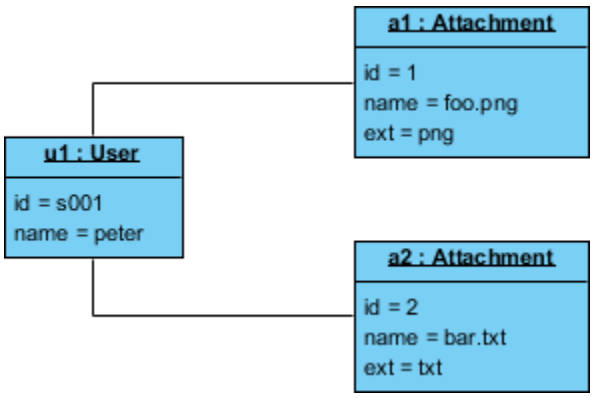
The following Class Diagram example represents two classes - User and Attachment. A user can upload multiple attachment so the two classes are connected with an association, with 0..* as multiplicity on the Attachment side.

Class Diagram



The following Object Diagram example shows you how the object instances of User and Attachment class "look like" at the moment Peter (i.e. the user) is trying to upload two attachments. So there are two Instance Specification for the two attachment objects to be uploaded.

Object Diagram

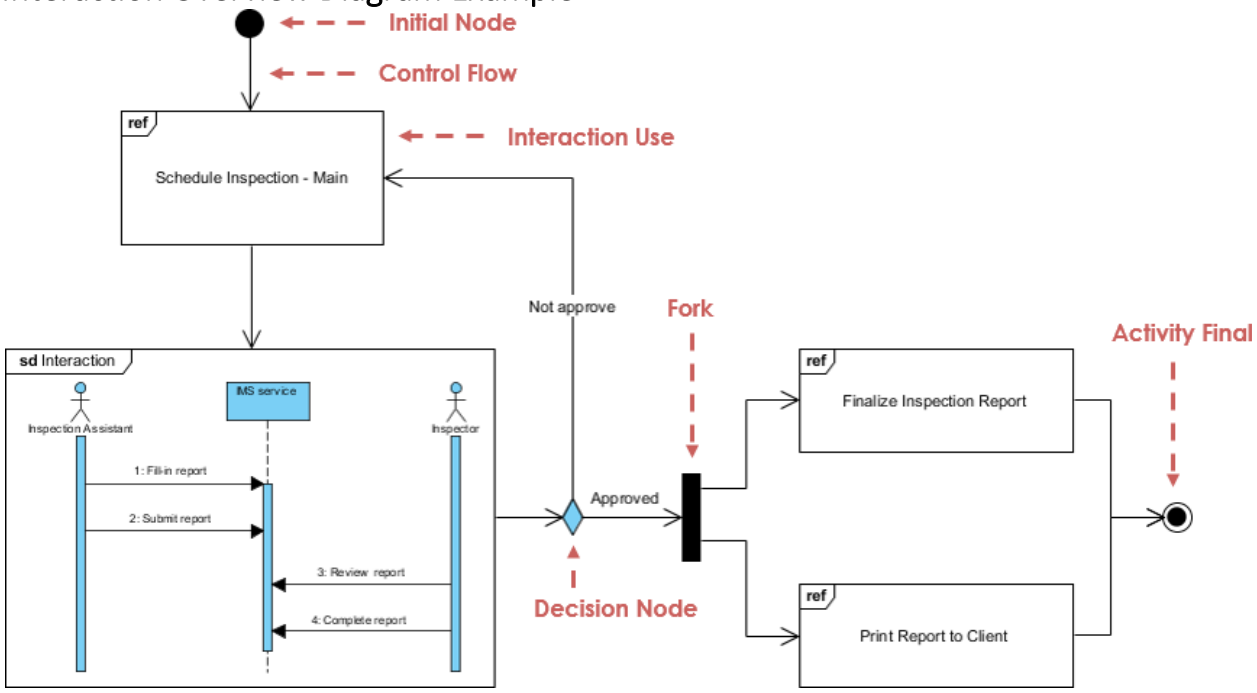


7.2.4 Interaction Diagrams

What is Interaction Overview Diagram?

The Interaction Overview Diagram focuses on the overview of the flow of control of the interactions. It is a variant of the Activity Diagram where the nodes are the interactions or interaction occurrences. The Interaction Overview Diagram describes the interactions where messages and lifelines are hidden. You can link up the "real" diagrams and achieve high degree navigability between diagrams inside the Interaction Overview Diagram.

Interaction Overview Diagram Example



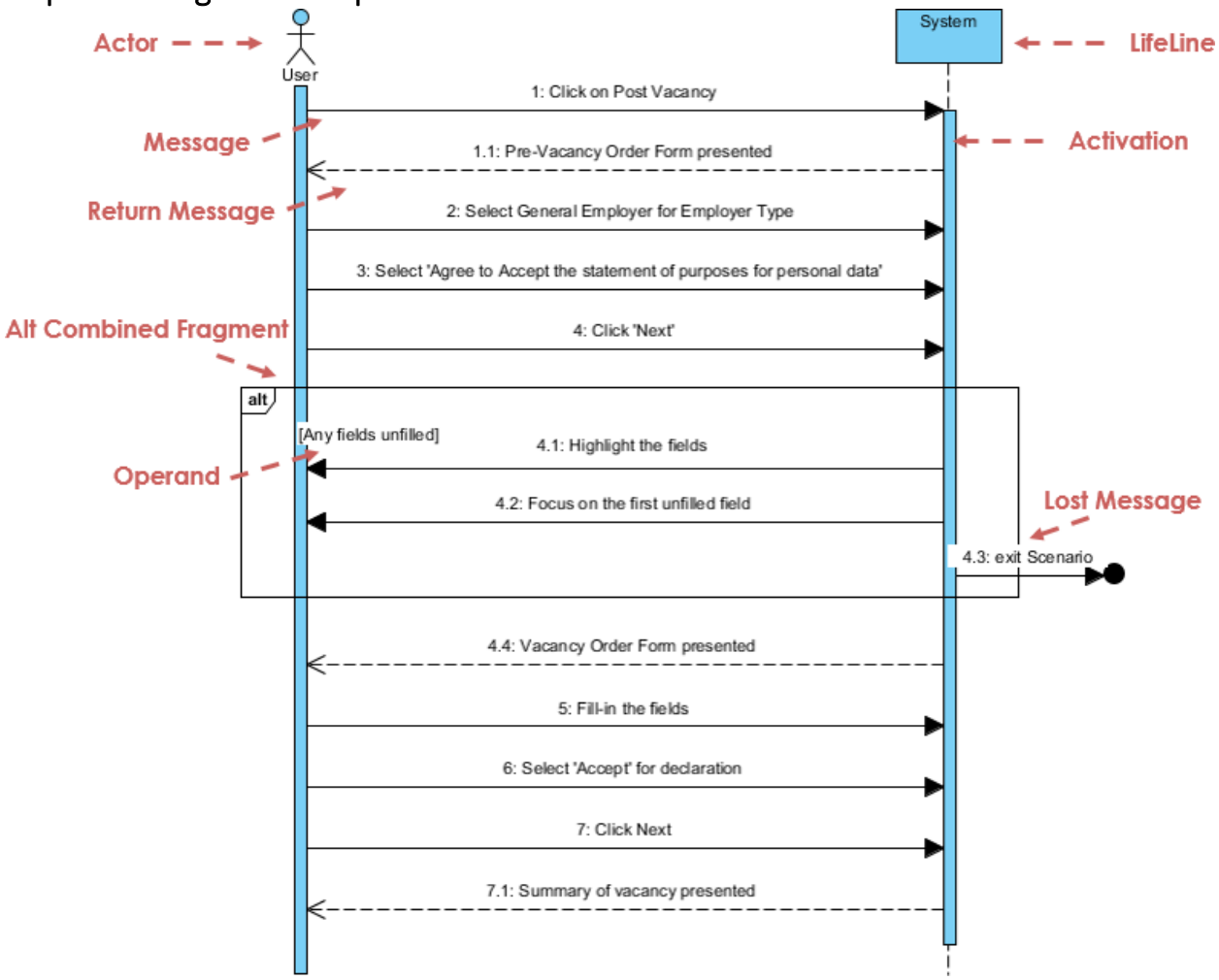
7.2.5 Sequence and Collaboration Diagram

What is a Sequence Diagram?

The Sequence Diagram models the collaboration of objects based on a time sequence. It shows how the objects interact with others in a particular scenario of a use case. With the advanced visual modeling capability, you can create complex sequence diagram in few clicks. Besides, some modeling tool such as Visual

Paradigm can generate sequence diagram from the flow of events which you have defined in the use case description.

Sequence Diagram Example

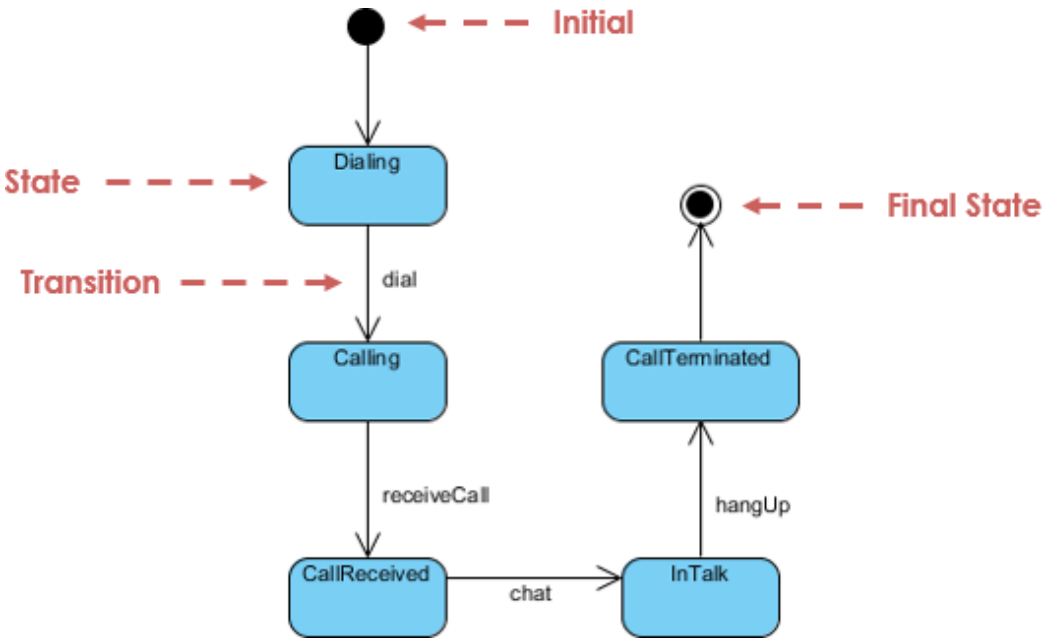


7.2.6 State Diagram

What is a State Machine Diagram?

A state diagram is a type of diagram used in UML to describe the behavior of systems which is based on the concept of state diagrams by David Harel. State diagrams depict the permitted states and transitions as well as the events that effect these transitions. It helps to visualize the entire lifecycle of objects and thus help to provide a better understanding of state-based systems.

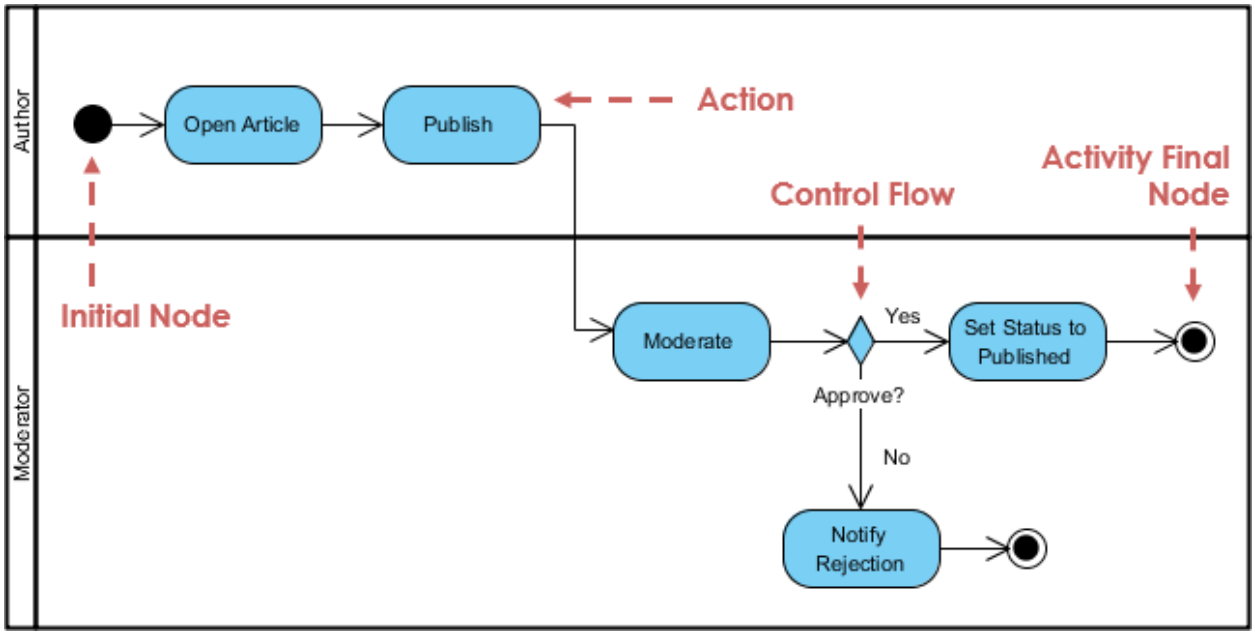
State Machine Diagram Example



7.2.7 Activity Diagram

What is an Activity Diagram?

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. It describes the flow of control of the target system, such as the exploring complex business rules and operations, describing the use case also the business process. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows).



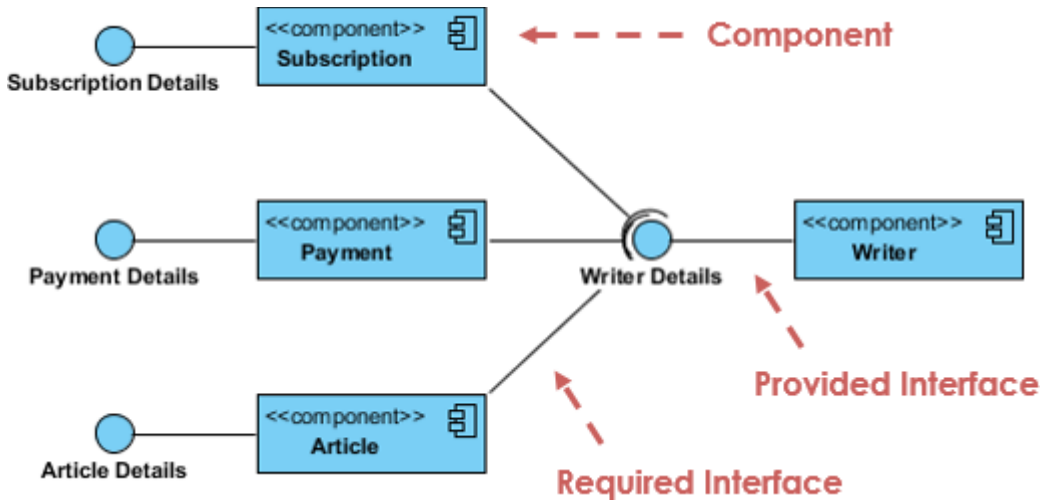
Activity Diagram Example

7.2.8 Component Diagram

What is Component Diagram?

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components or software systems. It illustrates the architectures of the software components and the dependencies between them. Those software components including run-time components, executable components also the source code components.

Component Diagram Example

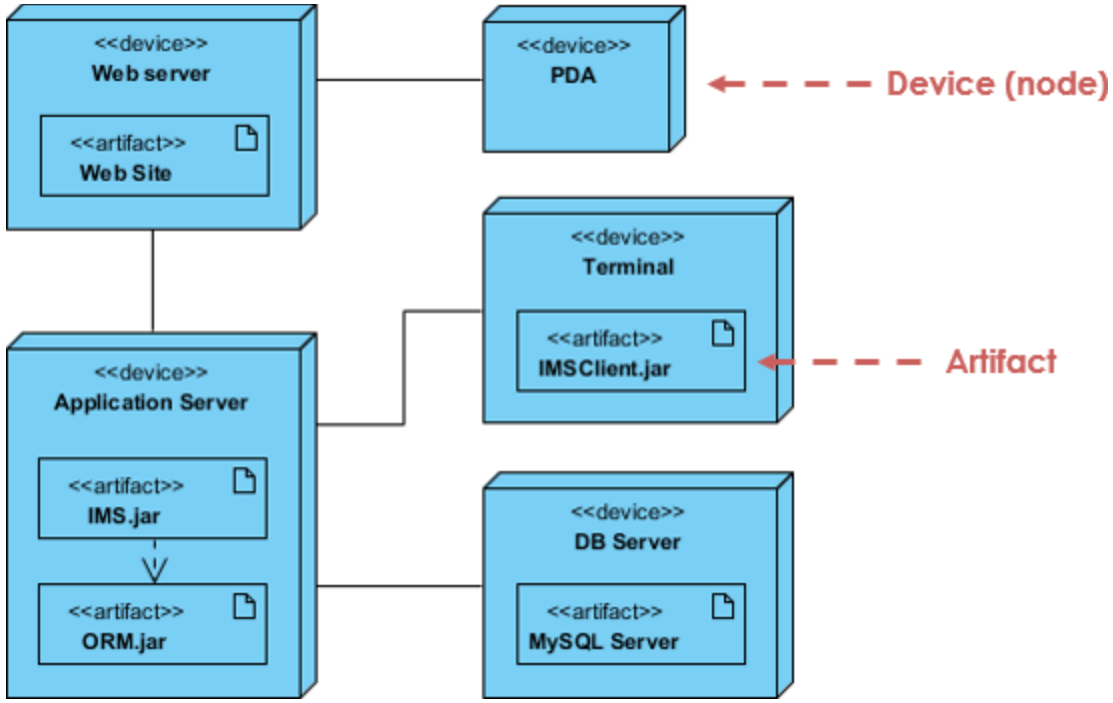


7.2.9 Deployment Diagram

What is a Deployment Diagram?

The Deployment Diagram helps to model the physical aspect of an Object-Oriented software system. It is a structure diagram which shows architecture of the system as deployment (distribution) of software artifacts to deployment targets. Artifacts represent concrete elements in the physical world that are the result of a development process. It models the run-time configuration in a static view and visualizes the distribution of artifacts in an application. In most cases, it involves modeling the hardware configurations together with the software components that lived on.

Deployment Diagram Example


<<device>>
Web server
<<artifact>>
Web Site
<<device>>
PDA
Device (node)
<<device>>
Terminal
<<artifact>>
IMSClient.jar
Artifact
<<device>>
Application Server
<<artifact>>
IMS.jar
<<artifact>>
ORM.jar
<<device>>
DB Server
<<artifact>>
MySQL Server

BOONU

7.3 Object Oriented Analysis: Requirement Analysis using Use Case Model, Conceptual Modeling

7.3.1 Object Oriented Analysis:

Object-oriented analysis and design (OOAD) is a popular technical approach for analyzing and designing an application, system, or business by applying object-oriented programming, as well as using visual modeling throughout the development life cycles to foster better stakeholder communication and product quality.

Structured Analysis vs. Object Oriented Analysis

The Structured Analysis/Structured Design (SASD) approach is the traditional approach of software development based upon the waterfall model. The phases of development of a system using SASD are −

- Feasibility Study
- Requirement Analysis and Specification
- System Design
- Implementation
- Post-implementation Review

Now, we will look at the relative advantages and disadvantages of structured analysis approach and object-oriented analysis approach.

Advantages/Disadvantages of Object Oriented Analysis

Advantages	Disadvantages
Focuses on data rather than the procedures as in Structured Analysis.	Functionality is restricted within objects. This may pose a problem for systems which are intrinsically procedural or computational in nature
The principles of encapsulation and data hiding help the developer to develop systems that cannot be tampered by other parts of the system.	It cannot identify which objects would generate an optimal system design.
The principles of encapsulation and data hiding help the developer to develop systems that cannot be tampered by other parts of the system.	The object-oriented models do not easily show the communications between the objects in the system.

It allows effective management of software complexity by the virtue of modularity.	All the interfaces between the objects cannot be represented in a single diagram.

7.3.2 Requirement Analysis using Use Case Model,

http://www.processimpact.com/articles/usecase.html

7.3.3 Conceptual Modeling

A conceptual model is a representation of a system, made of the composition of concepts which are used to help people know, understand, or simulate a subject the model represents. It is also a set of concepts. Some models are physical objects; for example, a toy model which may be assembled, and may be made to work like the object it represents.

The term conceptual model may be used to refer to models which are formed after a conceptualization or generalization process.

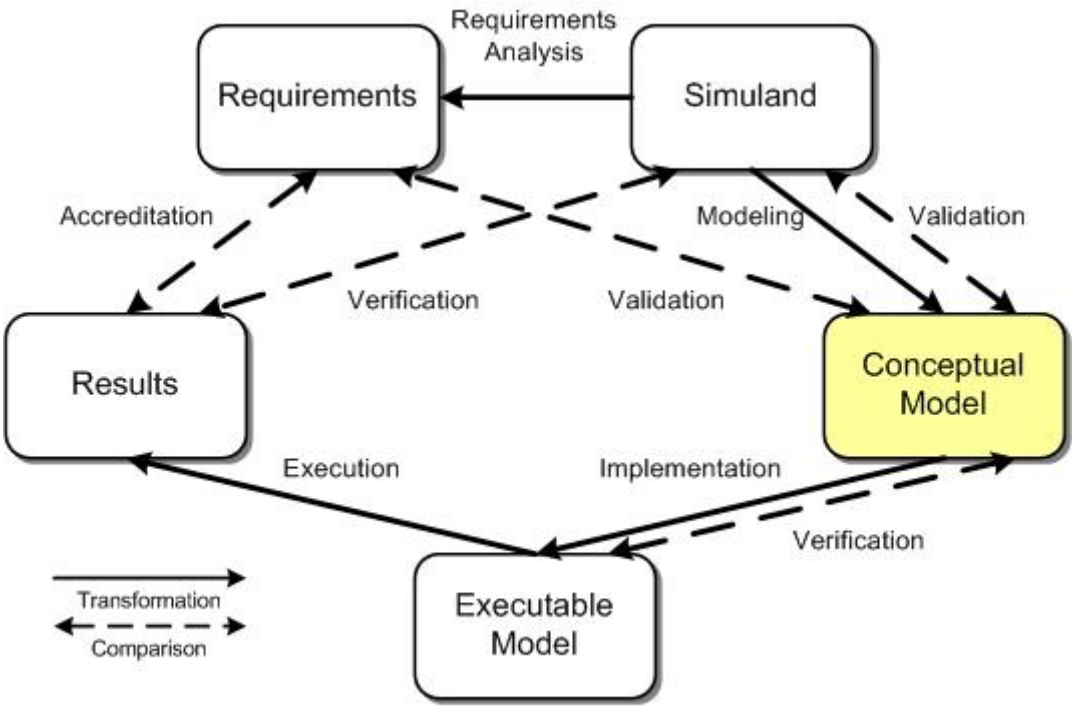
Fundamental objectives

A conceptual model's primary objective is to convey the fundamental principles and basic functionality of the system which it represents. Also, a conceptual model must be developed in such a way as to provide an easily understood system interpretation for the models users. A conceptual model, when implemented properly, should satisfy four fundamental objectives.[4]

- Enhance an individual's understanding of the representative system
- Facilitate efficient conveyance of system details between stakeholders
- Provide a point of reference for system designers to extract system specifications
- Document the system for future reference and provide a means for collaboration

The conceptual model plays an important role in the overall system development life cycle. Figure below, depicts the role of the conceptual model in a typical system development scheme. It is clear that if the conceptual model is not fully developed, the execution of fundamental system properties may not be implemented properly, giving way to future problems or system shortfalls. These failures do occur in the industry and have been linked to; lack of user input, incomplete or unclear requirements, and

changing requirements. Those weak links in the system design and development process can be traced to improper execution of the fundamental objectives of conceptual modeling. The importance of conceptual modeling is evident when such systemic failures are mitigated by thorough system development and adherence to proven development objectives/techniques.



J. Sokolowski, C. Banks, Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains, Wiley, 2010, pp 333

Conceptual models are often abstractions of things in the real world whether physical or social. Semantics studies are relevant to various stages of concept formation and use as Semantics is basically about concepts, the meaning that thinking beings give to various elements of their experience.

A mental model captures ideas in a problem domain, while a conceptual model represents 'concepts' (entities) and relationships between them.

A conceptual model in the field of computer science is a special case of a general conceptual model. To distinguish from other types of models, it is also known as a domain model. Conceptual modeling should not be confused with other modeling disciplines such as data modelling, logical modelling and physical modelling. The conceptual model is explicitly chosen to be independent of design or implementation concerns, for example, concurrency or data storage. The aim of a conceptual model is to

express the meaning of terms and concepts used by domain experts to discuss the problem, and to find the correct relationships between different concepts. The conceptual model attempts to clarify the meaning of various, usually ambiguous terms, and ensure that problems with different interpretations of the terms and concepts cannot occur. Such differing interpretations could easily cause confusion amongst stakeholders, especially those responsible for designing and implementing a solution, where the conceptual model provides a key artifact of business understanding and clarity. Once the domain concepts have been modeled, the model becomes a stable basis for subsequent development of applications in the domain. The concepts of the conceptual model can be mapped into physical design or implementation constructs using either manual or automated code generation approaches. The realization of conceptual models of many domains can be combined to a coherent platform.

A conceptual model can be described using various notations, such as UML, ORM or OMT for object modelling, or IE or IDEF1X for Entity Relationship Modelling. In UML notation, the conceptual model is often described with a class diagram in which classes represent concepts, associations represent relationships between concepts and role types of an association represent role types taken by instances of the modelled concepts in various situations. In ER notation, the conceptual model is described with an ER Diagram in which entities represent concepts, cardinality and optionality represent relationships between concepts. Regardless of the notation used, it is important not to compromise the richness and clarity of the business meaning depicted in the conceptual model by expressing it directly in a form influenced by design or implementation concerns.

This is often used for defining different processes in a particular company or institute.

Advantages of Conceptual Modeling

Since conceptual models are merely representations of abstract concepts and their respective relationships, the potential advantages of implementing a conceptual model are many, but largely depend on your own ability to devise a strong model in the first place. Generally speaking, the primary advantages of a conceptual model include:

Establishes Entities: By establishing and defining all the various entities and concepts that are likely to come up throughout the course of a software development life cycle, a

conceptual model can help ensure that there are fewer surprises down the road, where entities or relationships might otherwise have been neglected or forgotten.

Defines Project Scope: A solid conceptual model can be used as a way to define project scope, which assists with time management and scheduling.

Base Model for Other Models: For most projects, additional, less abstract models will need to be generated beyond the rough concepts defined in the conceptual model. Conceptual models serve as a great jumping-off point from which more concrete models can be created, such as logical data models and the like.

High-Level Understanding: Conceptual models serve as a great tool by providing a high-level understanding of a system throughout the software development life cycle. This can be particularly beneficial for managers and executives, who may not be dealing directly with coding or implementation, but require a solid understanding of the system and the relationships therein.

Disadvantages of Conceptual Modeling

Since a conceptual model is so abstract, and thus, is only as useful as you make it, there can be a few disadvantages or caveats to watch out for when implementing your own conceptual model:

Creation Requires Deep Understanding: While conceptual models can (and should) be adaptive, proper creation and maintenance of a conceptual model requires a fundamental and robust understanding of the project, along with all associated entities and relationships.

Potential Time Sink: Improper modeling of entities or relationships within a conceptual model may lead to massive time waste and potential sunk costs, where development and planning have largely gone astray of what was actually necessary in the first place.

Possible System Clashes: Since conceptual modeling is used to represent such abstract entities and their relationships, it's possible to create clashes between various components. In this case, a clash simply indicates that one component may conflict with another component, somewhere down the line. This may be seen when design or coding

clash with deployment, as the initial assumptions of scaling during design and coding were proven wrong when actual deployment occurred.

Implementation Challenge Scales With Size: While conceptual models are not inherently ill-suited for large applications, it can be challenging to develop and maintain a proper conceptual model for particularly complex projects, as the number of potential issues, or clashes, will grow exponentially as the system size increases.

7.4 Object Oriented Design: Defining Interaction Diagrams, Defining Design Class Diagrams

Ref:
https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_design.htm

7.4.1 Object Oriented Design

What is Object Oriented Design? (OOD)

Object Oriented Design is the concept that forces programmers to plan out their code in order to have a better flowing program. The origins of object oriented design is debated, but the first languages that supported it included Simula and SmallTalk. The term did not become popular until Grady Booch wrote the first paper titled Object-Oriented Design, in 1982.

Object Oriented Design is defined as a programming language that has 5 conceptual tools to aid the programmer. These programs are often more readable than non-object oriented programs, and debugging becomes easier with locality.

Language Concepts

The 5 Basic Concepts of Object Oriented Design are the implementation level features that are built into the programming language. These features are often referred to by these common names:

1. Encapsulation

A tight coupling or association of data structures with the methods or functions that act on the data. This is called a class, or object (an object is often the implementation of a class).

2. Data Protection

The ability to protect some components of the object from external entities. This is realized by language keywords to enable a variable to be declared as private or protected to the owning class.

3. Inheritance

The ability for a class to extend or override functionality of another class. The so called child class has a whole section that is the parent class and then it has it's own set of functions and data.

4. Interface

A definition of functions or methods, and their signatures that are available for use to manipulate a given instance of an object.

5. Polymorphism

The ability to define different functions or classes as having the same name but taking different data types.

Programming Concepts

There are several concepts that were derived from the new languages once they became popular. The new standards that came around pushed on three major things:

1. Re-usability

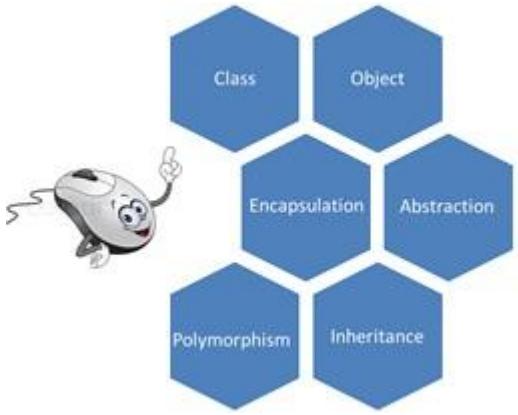
The ability to reuse code for multiple applications. If a programmer has already written a power function, then it should be written that any program can make a call to that function and it should work exactly the same.

2. Privacy

This is important for large programs and preventing loss of data.

3. Documentation

The commenting of a program in mark up that will not be converted to machine code. This mark up can be as long as the programmer wants, and allows for comprehensive information to be passed on to new programmers. This is important for both the re-usability and the maintainability of programs.



7.4.2 Defining Interaction Diagrams

From the term Interaction, it is clear that the diagram is used to describe some type of interactions among the different elements in the model. This interaction is a part of dynamic behavior of the system.

This interactive behavior is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram. The basic purpose of both the diagrams are similar.

Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

Purpose of Interaction Diagrams

The purpose of interaction diagrams is to visualize the interactive behavior of the system. Visualizing the interaction is a difficult task. Hence, the solution is to use different types of models to capture the different aspects of the interaction.

Sequence and collaboration diagrams are used to capture the dynamic nature but from a different angle.

The purpose of interaction diagram is –

1. To capture the dynamic behaviour of a system.

2. To describe the message flow in the system.
3. To describe the structural organization of the objects.
4. To describe the interaction among objects.

How to Draw an Interaction Diagram?

As we have already discussed, the purpose of interaction diagrams is to capture the dynamic aspect of a system. So to capture the dynamic aspect, we need to understand what a dynamic aspect is and how it is visualized. Dynamic aspect can be defined as the snapshot of the running system at a particular moment

We have two types of interaction diagrams in UML. One is the sequence diagram and the other is the collaboration diagram. The sequence diagram captures the time sequence of the message flow from one object to another and the collaboration diagram describes the organization of objects in a system taking part in the message flow.

Following things are to be identified clearly before drawing the interaction diagram

1. Objects taking part in the interaction.
2. Message flows among the objects.
3. The sequence in which the messages are flowing.
4. Object organization.

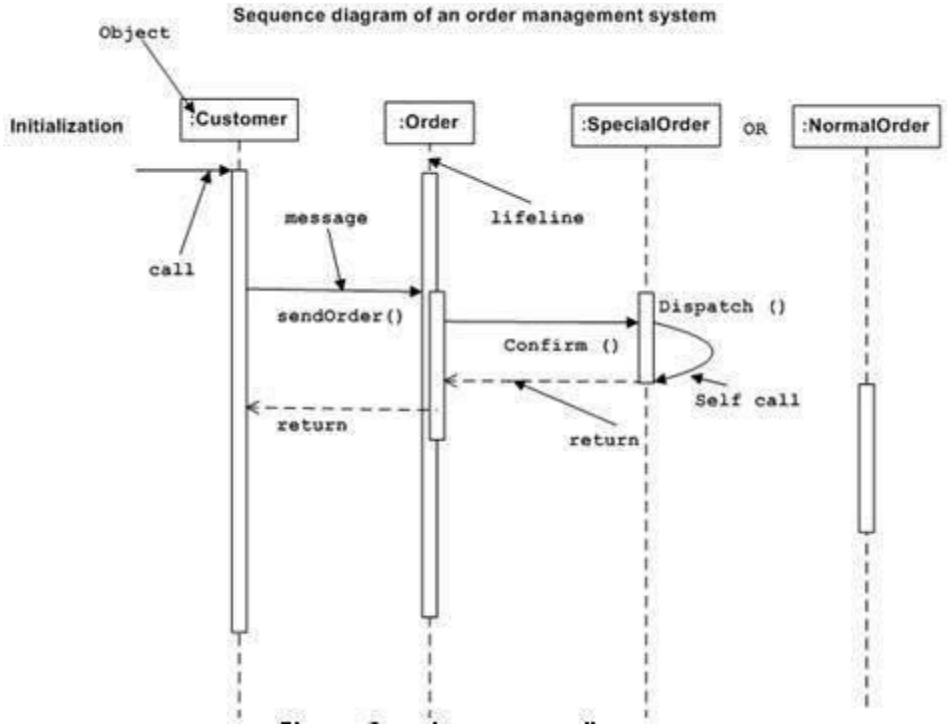
Following are two interaction diagrams modeling the order management system. The first diagram is a sequence diagram and the second is a collaboration diagram

The Sequence Diagram

The sequence diagram has four objects (Customer, Order, SpecialOrder and NormalOrder).

The following diagram shows the message sequence for SpecialOrder object and the same can be used in case of NormalOrder object. It is important to understand the time sequence of message flows. The message flow is nothing but a method call of an object.

The first call is sendOrder () which is a method of Order object. The next call is confirm () which is a method of SpecialOrder object and the last call is Dispatch () which is a method of SpecialOrder object. The following diagram mainly describes the method calls from one object to another, and this is also the actual scenario when the system is running.

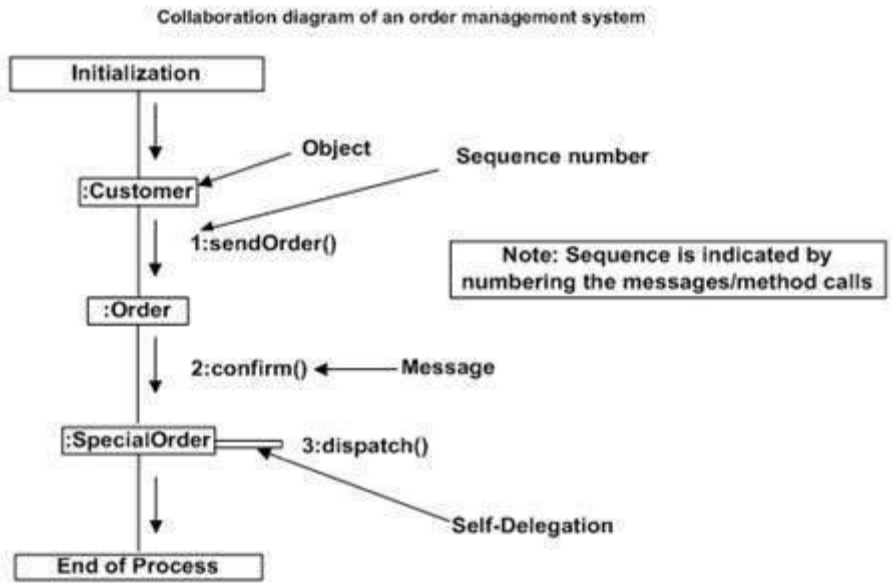


The Collaboration Diagram

The second interaction diagram is the collaboration diagram. It shows the object organization as seen in the following diagram. In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.

Method calls are similar to that of a sequence diagram. However, difference being the sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization.

To choose between these two diagrams, emphasis is placed on the type of requirement. If the time sequence is important, then the sequence diagram is used. If organization is required, then collaboration diagram is used.



Where to Use Interaction Diagrams?

We have already discussed that interaction diagrams are used to describe the dynamic nature of a system. Now, we will look into the practical scenarios where these diagrams are used. To understand the practical application, we need to understand the basic nature of sequence and collaboration diagram.

The main purpose of both the diagrams are similar as they are used to capture the dynamic behavior of a system. However, the specific purpose is more important to clarify and understand.

Sequence diagrams are used to capture the order of messages flowing from one object to another. Collaboration diagrams are used to describe the structural organization of the objects taking part in the interaction. A single diagram is not sufficient to describe the dynamic aspect of an entire system, so a set of diagrams are used to capture it as a whole.

Interaction diagrams are used when we want to understand the message flow and the structural organization. Message flow means the sequence of control flow from one object to another. Structural organization means the visual organization of the elements in a system.

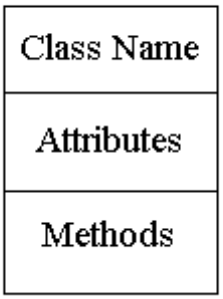
Interaction diagrams can be used –

1. To model the flow of control by time sequence.
2. To model the flow of control by structural organizations.
3. For forward engineering.
4. For reverse engineering.

7.4.3 Defining Design Class Diagrams

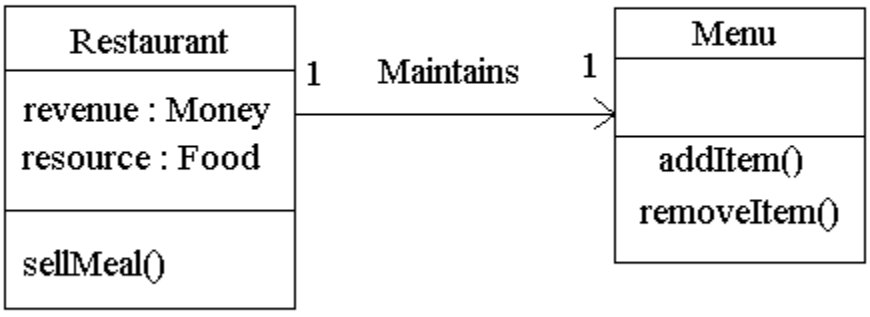
UML design class diagrams (DCD) show software class definitions. They are based on the collaboration diagram. Attribute visibility is shown for permanent connections. Classes are shown with their simple attributes and methods listed.

DCD Syntax



Some attributes are depicted using associations (relationships) rather than actually being listed in the class block. These associated attributes refer to complex objects which should also be shown in the diagram. The collaboration diagram indicates methods to be contained in a class with methods posted as relationships. For instance the Schedule class has a findSeat(route, preference) method.

Partial Design Class Diagram



BOONU