Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

SYSTEMS DEVELOPMENT METHODOLOGY

Day 1

Course Objective

- To understand the need for software engineering.
- To have brief overview of the different phases of software development life cycle.
- To introduce the need for good practice in developing software.
- To have an insight into effective methods of Testing the software.
- To understand the importance of following the quality standards.

Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

Evolution of Software

The Importance of Software

- Software packs and delivers the most important product of our time- *Information*
- Software is the differentiator
  - Hardware is general-purpose, software is application-specific
  - Hardware is becoming a commodity
- Software is often costlier than the hardware it runs on.
- Software is crucial for business success

The Importance of Software

Consider a Nike salesman called who moves around with a laptop taking orders from big departmental stores. Because of the database on the laptop, The salesperson is able to talk of all the styles available, the lead times, the popularity ratings of the various styles, etc.

Once the salesperson takes the order, he thanks the buyer, and promises to be back the next day with a statement of the discounts that will be available to the buyer based on his past buying history.

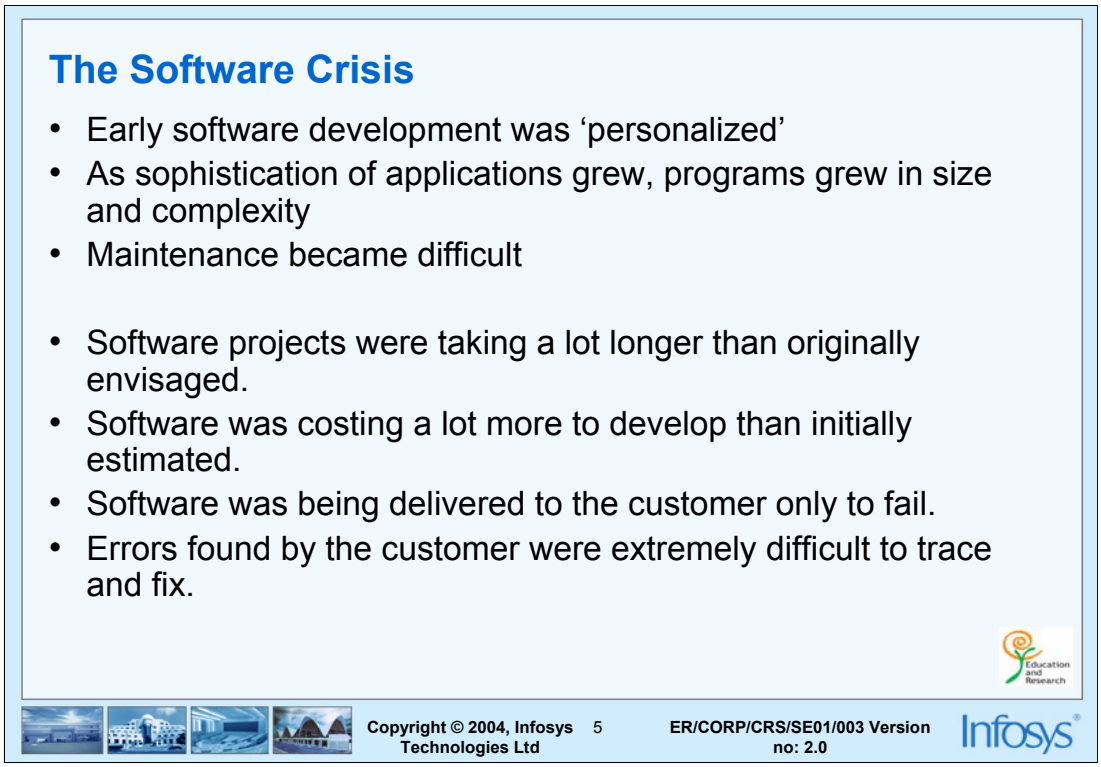
All is well till one day the salesman's girlfriend, who works for the competing shoe and sports apparel firm Reebok, is given a laptop that helps her do not only what Sam (her boyfriend) can do, but also computes discounts the moment the order is entered. This allows her client to modify his order to his satisfaction and confirm the order right away instead of waiting for the salesperson to come back the next day. So Sam finds, in due course, that he loses business from his once-regular customers.

Why does this happen simply because one laptop software is better than the other? Don't the shoes themselves count? Since the quality of shoes is practically the same (and this is the key point in a highly competitive market) it is the *service* that counts. And this example shows how service is catalyzed by superior software.

Software has become a key underpinning element of national activity. Every sector of the economy including transport, telecommunications, manufacturing, services and finance today depends significantly on computers — and hence software — for its operations, and this dependence is increasing.

The basic paradigm governing the application of computers has always been to use hardware that is fairly general-purpose in nature, and develop software to customise that hardware to application-specific needs. Thus, while hardware is rapidly moving towards becoming a commodity, software is the factor that *differentiates.* In the commercial context, this means that the quality of software driving computer-based solutions differentiates a company from its competitors. It is not unusual for organisations to spend more on software than they did on the hardware it runs on.

The development and supply of software, both customised and packaged, has thus been recognised as an industry in its own right, and is in fact one of the fastest growing industry segments in the Indian economy today. Like any other real-world endeavour of economic significance, software development has its own problems and pitfalls. We look at some of these here, after a brief historical interlude to put them in perspective.



The Software Crisis

In the early years of computer applications, the focus of development and innovation was on hardware — software was largely viewed as an afterthought. Computer programming was an art, at best an arcane practice known to a gifted few. Programmers did not follow any disciplined or formalised approaches — these were confined to hardware designers. In this personalised approach, design was implicitly performed in one's head, and documentation was often nonexistent.

This way of doing things was adequate for a while, until the sophistication of computer applications outgrew this rough-hewn approach. Software soon took over more and more functions which were hitherto done manually or were hard-automated. As "software houses" emerged, software began to be developed for widespread distribution. Software development projects produced tens of thousands of source program statements. With no tools or methods of managing this increasing complexity, the number of errors began to accelerate. But the personalised nature of programming made *maintenance,* or the fixing of these errors, exceedingly difficult. The following situation resulted:

•Software projects were taking a lot longer than initially envisaged.

•Software was costing a lot more to develop than at first estimated.

•Software was being delivered to the customer only to fail.

•Errors found by the customer were extremely difficult to track down and fix.

These problems are collectively referred to as the "software crisis". Clearly, there was a need to rethink software development practices. The consensus which emerged was to borrow ideas about systematic development from another discipline, namely engineering.

Need For Engineering Approach

- Programming was considered as an 'art'
- Since it was personalized, the design was implicitly preformed in one's head and not on paper
- Later, business outgrowth called for sophistication in software development
- People tried to use the same techniques they adopted for programming, but failed
- Problems are not technical, but with people management, estimation and configuration management
- People decided to borrow ideas about systematic development from engineering, giving rise to a new discipline called *"Software Engineering"*

Need for Engineering Approach

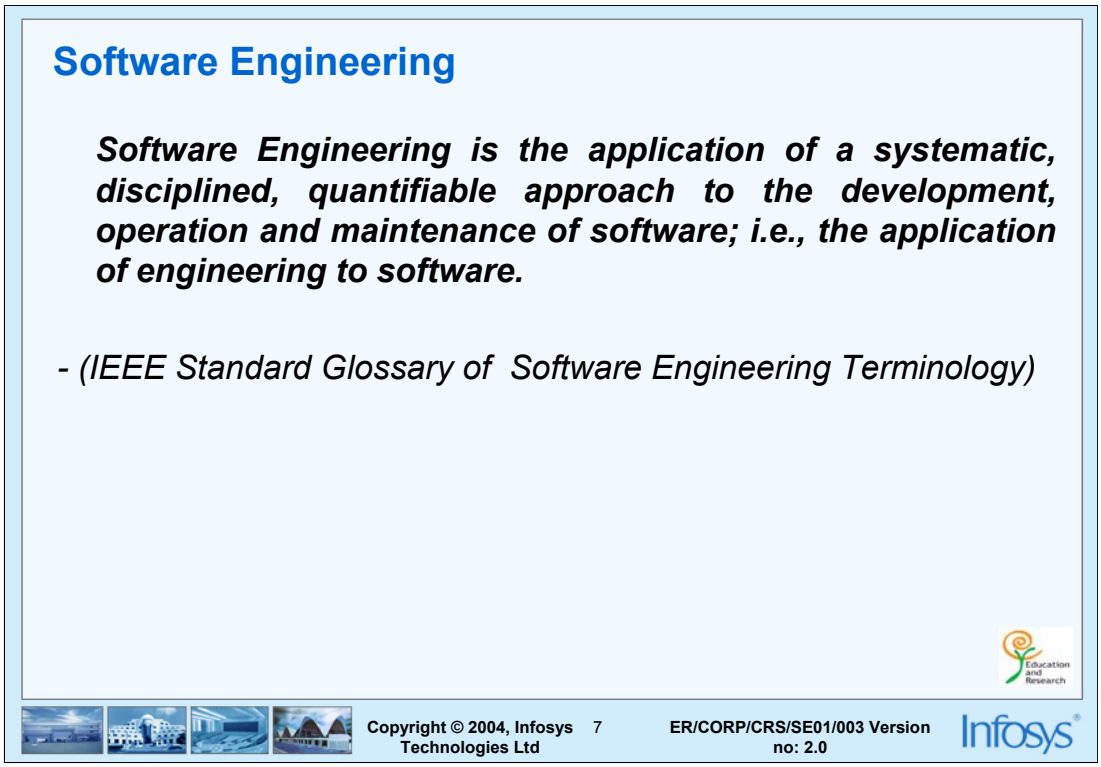
Some characteristics of the software scenario as it obtains today are

•Hardware sophistication has outpaced our ability to build software that completely taps its potential;

•Our ability to build software cannot keep pace with the demand for new software;

•Our ability to maintain existing programs is undermined by poor design and haphazard development practices;

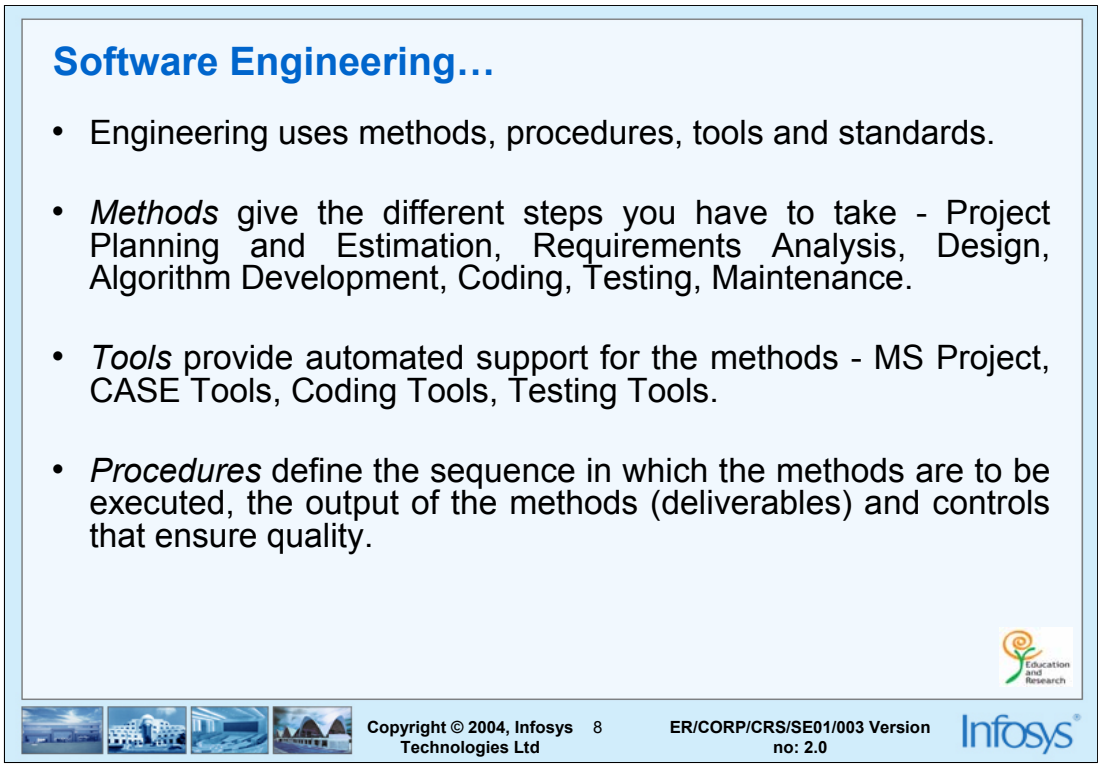
It is clear that a discipline for software development must exist --one that integrates comprehensive methods for all stages in software development, better tools for automating these methods, more powerful techniques for software quality assurance, and an overall philosophy for coordination, control and management. Such a discipline is Engineering.



Software Engineering

Software Engineering, the term coined at a NATO-sponsored conference in 1968, has come to represent the gamut of systematic approaches to the design, development and maintenance of software. The definition given below is reproduced from the IEEE Standard Glossary of Software Engineering Terminology:

Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; i.e., the application of engineering to software.



Software Engineering

The basic principle of Software Engineering is to use structured, formal, disciplined methods for building and using systems, just as in any other branch of engineering. Software Engineering provides a set of methods, a variety of tools, and a collection of procedures.

Methods provide the rules and steps for carrying out software engineering tasks, such as project planning and estimation, system and software requirements analysis, design of data structure, program architecture and algorithm procedure, coding, testing and maintenance.

Tools provide automated or semi-automated support for methods. Tools can be integrated into a system which automates a range of software engineering methods, called Computer-Aided Software Engineering (CASE). CASE tools are available to support and automate most of the methods mentioned above.

Procedures bind the methods and tools into a framework, defining the sequence in which methods will be applied, the deliverables (documents, forms, reports, etc.) that are required, the controls that ensure quality and coordinate change, and the milestones that help a manager assess progress.

Software Engineering provides certain paradigms that encompass the above methods, tools and procedures. These paradigms may be viewed as models of software development.

Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

Software Development Models

Software Project

- Completing a software project is a team effort
- Within each software project team, there are many roles played by people
- Project Manager
  - Initiates, plans, tracks and manages resources of an entire project
- Module Leader
  - A software engineer who manages and leads the team working on a particular module
- Developer
  - A software engineer who writes code, tests it and delivers it error free

Software Project…

- Analyst
  - A software engineer who analyzes the requirements gathered.
- Architect
  - A software engineer involved in the design of the solution after the analyst has clearly specified the business requirements.
- System Engineer/Domain Expert
  - A software engineer who knows the system of which the software is a part.
- Reviewer
  - A software engineer who is involved in reviewing artifacts like project documents or code.

Software Project…

- Tester
  - A software engineer who conducts tests on completed software or a unit of the software.

Software Development Life Cycle

- SDLC is a abstract description of software development and modification process.
- The software development process undergoes several phases.
- All these phases together form the Software Development Life Cycle (SDLC)
- We will discuss three models for Software Development

Software Development Lifecycle Models

A lifecycle model, also called as process model, is an abstract description of the software development and modification process. It maps out the main stages in which the above process is carried out. Software Engineering provides certain process models for carrying out the software development activity. There are various and numerous methods to guide the life cycle of a software project. We can nonetheless define a few canonical models from which the wide variety of methods is local adaptations. We will attempt to compare three of these various canonical models and define how we can adapt them to particular project situations.

The Waterfall Model or Classic Life Cycle Model

- The Waterfall Model contains several sequential steps
  - Systems Engineering
  - Requirements Analysis
  - Design
  - Coding
  - Testing
  - Deployment

The Waterfall Model

The waterfall model, or so-called Classic Life Cycle model ('life cycle' is a term borrowed from biology), was the first structured approach to systems development. It was introduced by Royce in 1970 to help cope with the increasing complexity of aerospace products. It gained a widespread popularity during the early 70s as it allowed a certain amount of order in the process. The waterfall model followed a documentation driven paradigm. It is nothing but a sequence of activities (or lifecycle stages) ordered in time. Following are the lifecycle stages defined in this model in the exact time-order (next activity in sequence will start only after the first is complete):

•System Engineering

•Software Requirements Analysis

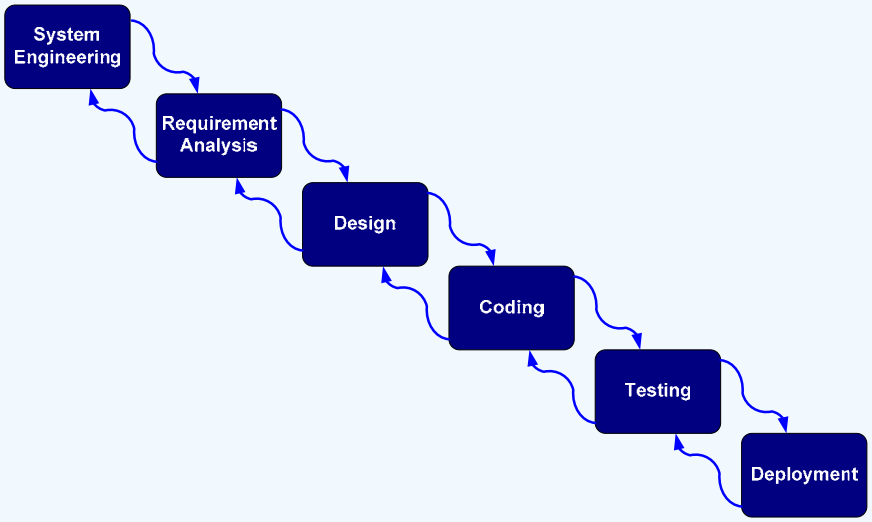
•Design

•Coding (Build)

•Testing

•Deployment

The Waterfall Model



System Engineering

- During the System Engineering phase, the *scope of the project* is defined after understanding the system as a whole
- For some requirements of the client, the solutions can be provided in terms of the technology to be used, area of automation etc.
- Artifacts : User Requirements for the system

System Engineering: The software to be developed must always function as part of a larger system. Hence it is necessary to start out by establishing requirements for the larger system. A subset of these is then assigned to the software. This stage ensures that the software will interface properly with people, hardware and other software in the system.

Both the functional and non-functional requirements must be understood clearly.

Functional requirements are the ones that specify the inputs to the system, the output from the system and the behavioral relationship between them. Eg: The software need to calculate and display the interest at the rate of 12% on all fixed deposits until maturity.

Non-Functional requirements are the requirements that describe the overall attributes of the system. Eg: Good user interface, High performance, Modular software and scalable model.

The activities performed by the project team in this phase are:

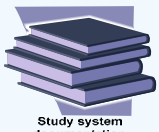
- Interview the customer to understand the requirements
- Interview customer's endusers
- Study the existing software, if any
- Collect and study the documentation of the existing software, if any
- Collect existing documents from the customer
- Interview the customer's software developers for more details on the project, and the standards the project has to follow.
- Identify the hardware platform, software models, programming language and tools that are needed in the project

Once the study is complete, the user requirements for the system to be developed are prepared and reviewed by the project team.

Comments and suggestions if any are incorporated and then sent to the customer for review.

System Engineering…


Interview Customer Stakeholders


Study system documentation


Interview the System Analysts and Software Developers

FEEDBACK


Create list of User Requirements


Review within Project


Final set of User Requirements


Interview end users

FEEDBACK


Study Existing Software

Requirement Analysis

- Once all the requirements are collected, the focus is on clearly understanding them.
- Here the software engineer plays the role of an analyst and understands all the functional, non-functional and interface requirements.
- Analyst has to ensure that the requirements as he understands is what the customer really wants.
- Artifacts :
  - SRS - Software Requirement Specification
  - Acceptance Test Plan
  - System Test Plan

Requirement Analysis : Here the software engineer, playing the role of analyst, understands the information domain, as well as the functional, performance and interfacing requirements. A different model of requirement and analysis is to perceive the system as a combination of process and data. It views the data and the processes as one unit and looks at modeling real life entities that make up the system. The process models and the data models are thoroughly viewed by the project team. Based on the feedback given it is improved. These models and requirements gathered are a part of the software requirement specification (SRS).

An activity conducted in parallel is to devise the System Test plan and Acceptance Test Plan.

System Test Plan : is devised by the domain consultant. It is used to ensure the system meets both the functional and non-functional requirements.

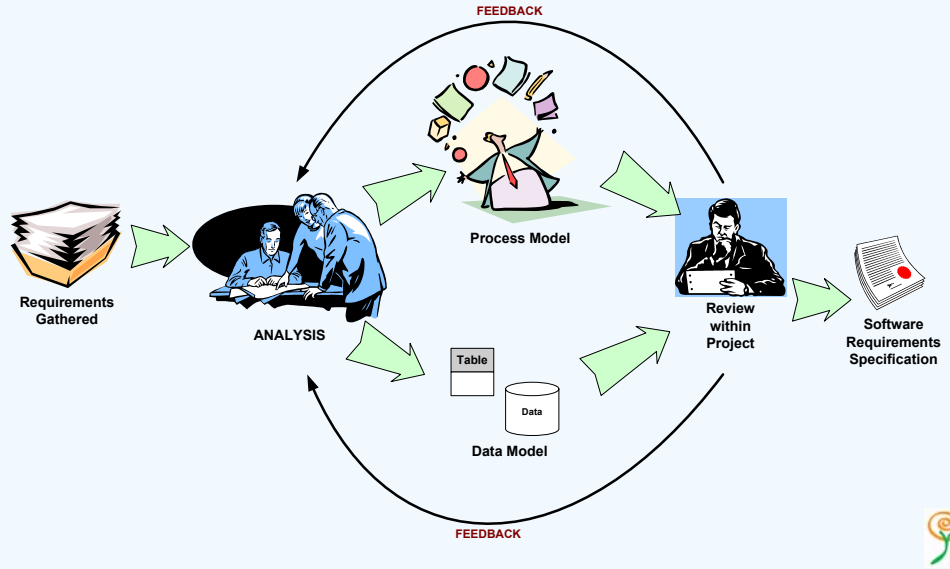
Acceptance Test Plan : unlike system testing it is done by the customer.

The SRS and the Test plan are sent to the customer for review. The customer reviews it and gives their comments and suggestions. After incorporating the review comments from the customer, the SRS and the Test Plans are finalized. Then the customer signs off the same.

The artifacts that need to be ready after this phase are:

- SRS
- Acceptance Test Plan
- System Test Plan

Requirement Analysis...



Design

- The output of the requirements analysis phase forms the input to the design phase.
- In design the requirements are translated into actual representation for software construction.
- The actual structure of the programs will be created in this phase
- Artifacts :
  - » High Level Design
  - » Detailed Design
  - » Integration Test Plan
  - » Unit Test Plan

Design: After the structured requirements are documented, the actual structure of the programs that will make up the software is created. The structure may be thought of as being determined by four distinct attributes: software architecture, data design, procedural detail and interface design.

Software Architecture identifies all the modules that make up the software. It portrays how the various modules of the software integrate to form a single complete entity.

Procedural Detail implies transforming the modules in the architecture into description of the business processes identified when requirements were analyzed.

Data Design focuses on the representation of data or information that is accessed by one or more modules.

Interface Design describes the communication among the different modules and also with the systems that interoperate with the software.

The design process thus translates the structured requirements into an actual representation of the software.

The four artifacts created to form the output of the design phase include :

High Level Design: describes the logical view of the solution. It views the solution at higher levels of abstraction.

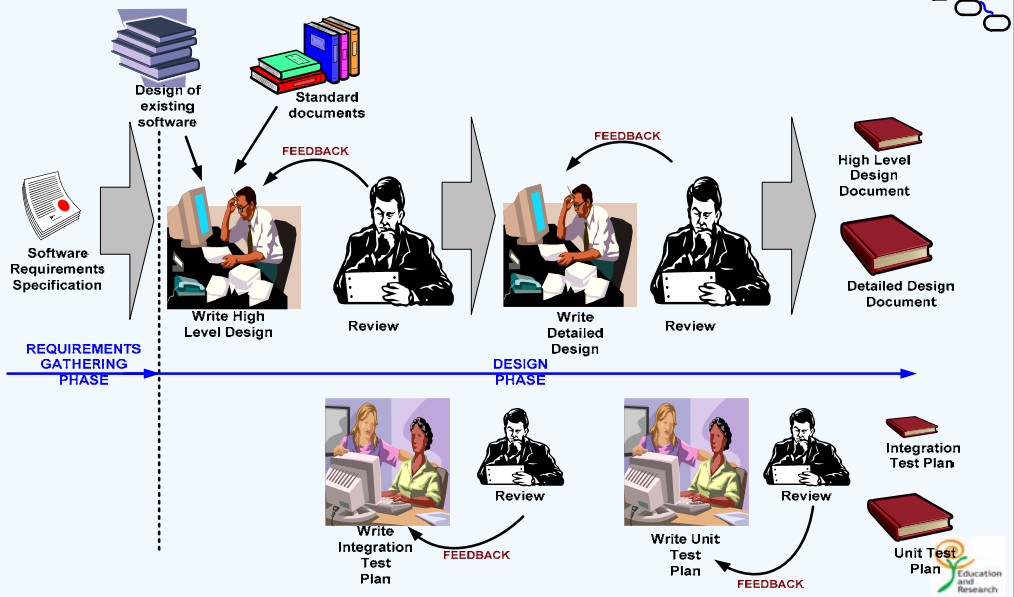
Integration Test Plan : contains test cases focusing on testing the interaction between various modules.

Detailed Design : takes HLD as input and goes one step further in giving detailed view of each of defined modules.

Unit Test Plan : contains test cases to verify the functionality within a particular module.

By the end of this phase the complete architecture of the software, the details of all functions and data to be used should be clear, reviewed and well represented.

Design...



Coding and Testing

- The program specifications are translated into a machine-readable form.
- The software engineers write the code that makes the software.
- The code will undergo different types of testing.
- Artifacts :
  - » Unit Tested Code
  - » Test Scripts for Integration Testing
  - » Test logs and reports after testing

Coding : The inputs to this phase are the documents from the design phase. Each software engineer is assigned a module. the steps in completing the module include :

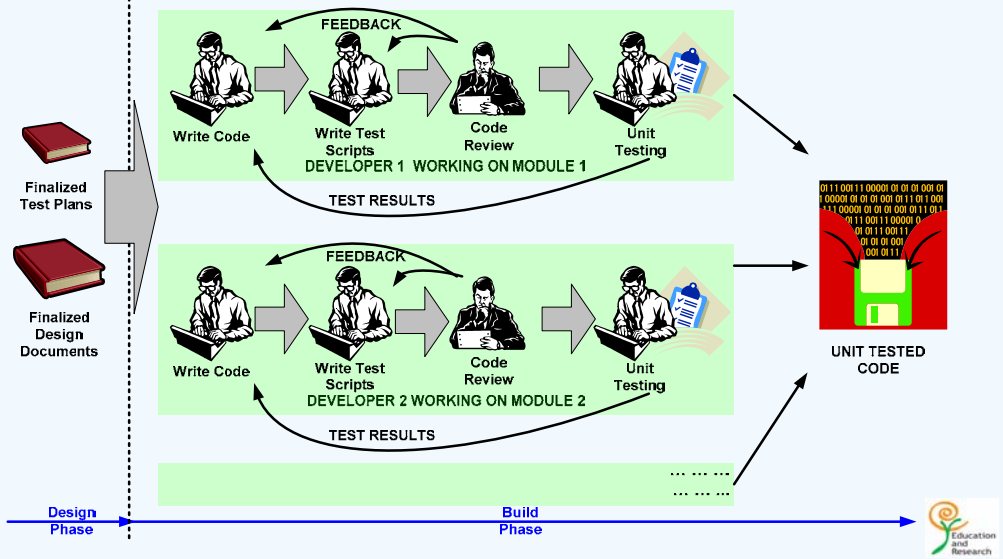
- •Write code
- •Write Unit Test Scripts which automates the steps identified for the test.
- •Code Review identifies the potential defects in the code and also assess whether or not the code adheres to the standards
- •Unit Testing tests the functionality within the module and done by the engineer who has written the code.

The output of this phase is the modular code, each of which has been unit tested and working and also test scripts for Integration Testing.

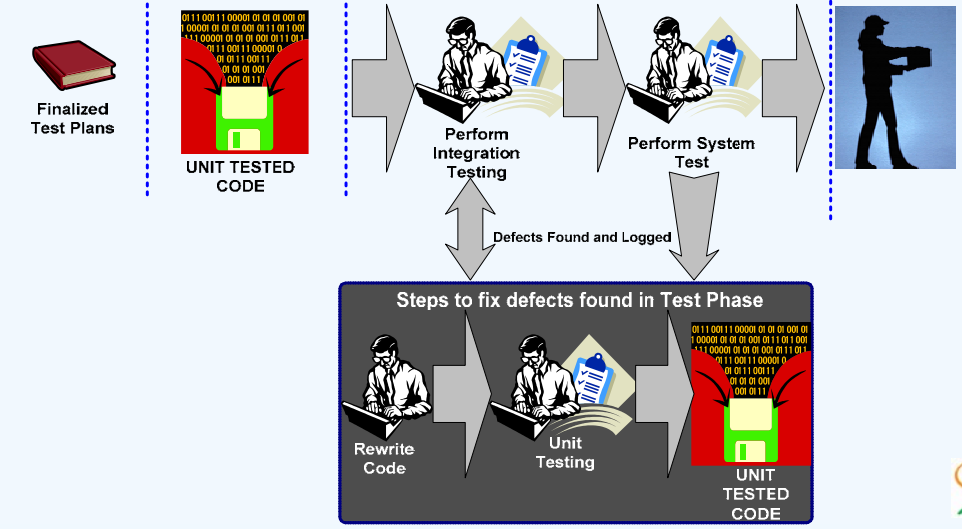
Testing : The entire unit tested code from the build phase must be integrated to form the complete software as foreseen in the design phase. After all the modules have been written and unit tests performed on them, they have to be put together; this is the process of integration. The integrated modules are then tested. This is called ***Integration Testing.***

Once all the modules are integrated and tested then the software is tested as a whole to ensure proper functionality of the software. This is called ***System Testing***. System Testing is to ensure that the software conforms to the stated business requirements.

Coding

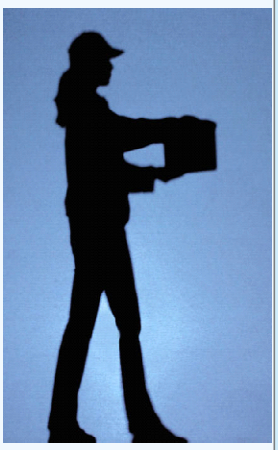


Testing



Deployment

- This phase mainly deals with packaging, installation, delivery and support of the developed software
- The complete delivery package needs to be assembled and tested
- Ensure that the plan for the support of the software is ready
- Early communication to customers about the changes that the software will bring about.
  - For example, reports might look different or location of information might have changed



Deployment : The final phase of the waterfall model is deployment. This phase mainly deals with packaging, installation and support of the developed software. The deployment of software represents an important milestone for any software project. A number of points need ro be kept in mind for the deployment of the software :

➢The complete delivery package needs to be assembled and tested.

➢Ensure that the plan for the support of the software is ready.

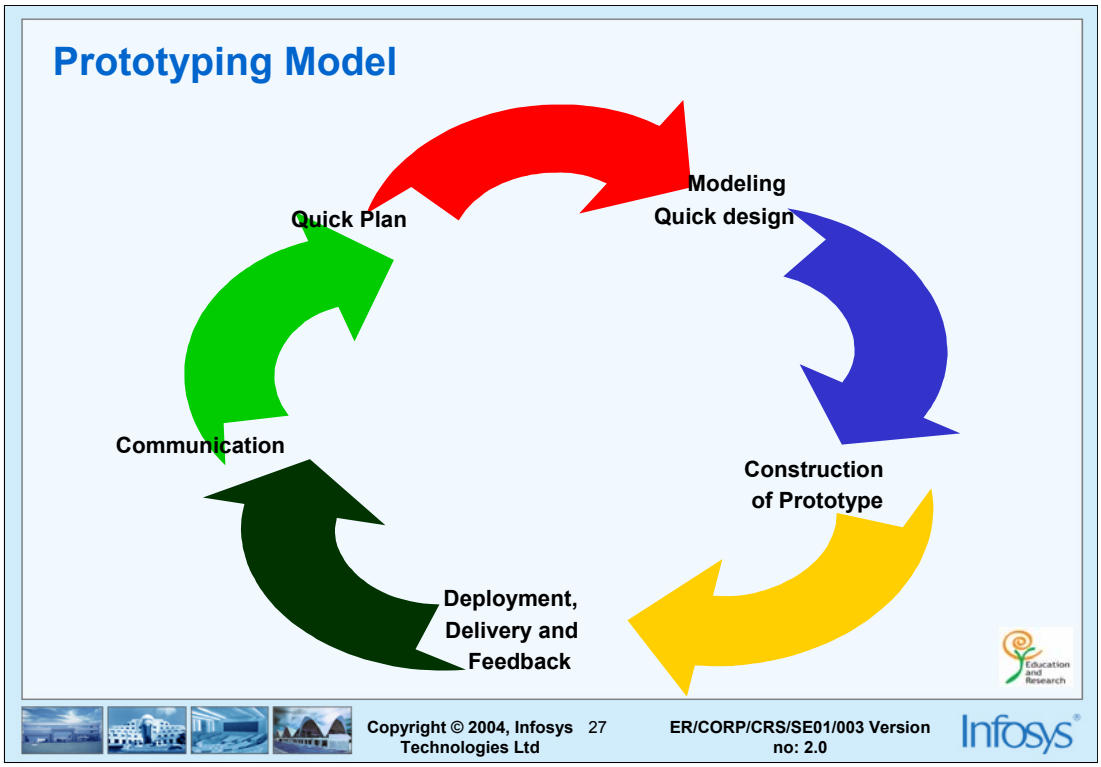
➢Early communication to customers about the changes that the software will bring about. For example – reports might look different or location of information might have changed.

Maintenance : This phase follows the Deployment phase. In this phase the deployed software had errors it is fixed and any enhancements that the customer wanted are done.

The complete scope of the project might be the maintenance of existing software. This could involve bringing in new features or upgrading the software and sometimes it would involve fixing the existing software to perform better.

Pros and Cons – Waterfall Model

- The Waterfall Model is very simple as the steps are quite logical.
- But the user may not be able to specify all the requirements in the beginning itself.
- Moreover, the final product is visible only at a very later stage.
- It does not capture potential risks.
- Major errors discovered at later stage leads to disastrous results.



In Prototyping model, the programmer first develops a prototype of the system.

•This prototype will not satisfy all the requirements of the client

•It may try to address only some of the important requirements

•The client, after evaluating this prototype will ask for modifications and additions

This cycle can be repeated to produce the final product.

The software engineer and the customer together define the overall objective for the software, identify the requirements that are known and outline area for further definition.

The iteration is quickly planned and modeled in quick design.The design focuses only on the aspects which are visible to the user.

The quick design leads to the construction of a prototype.

The prototype is evaluated by the customer.

The feedback is taken and used to refine the requirements for the software.

The iteration continue each time refining the software.

Prototyping Model...

- Throw Away
  - The model is thrown away once all requirements are understood.
- Evolving
  - Refine the model every time when the requirements are clearer and the prototype will evolve into the final software.

Irrespective of throwaway or evolving model, Prototyping starts with the communication phase.

Pros and Cons - Prototype Model

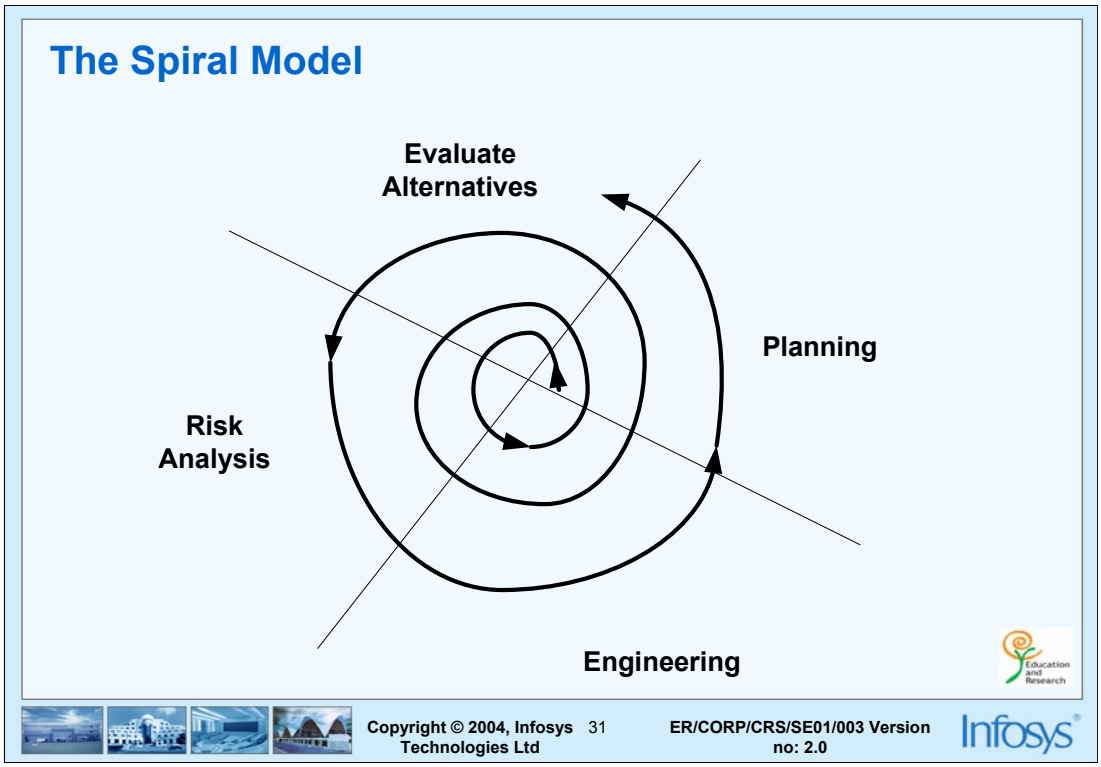
- The prototyping model has less technical risks
- There is scope for accommodating new requirements
- A part of the product is visible at an early stage itself
- This model may lead to indiscipline in software development

Undisciplined Development is something which the classic waterfall model tries to overcome. For example if some new requirements are received while in the construction phase, the engineer may drop construction and go back to planning or communication with customer. At some point of time the customer requirements has to be frozen in conformance with the customer or else it becomes an unending process, eating into project's time and resources.

The Spiral Model as Meta Model

- Spiral model is a meta-model, that is model for other models.
- Spiral Model gives room for risk analysis also
- Each phase, let it be Analysis or Design or Coding should iterate through four activities
  - Planning
  - Evaluate Alternatives
  - Analyze Risks
  - Engineering

The spiral model couples the iterative nature of the prototyping model and the controlled,systematic nature of the waterfall model. The model has a series of evolutionary releases of the software as it is being developed. Every iterative release might be a prototype. The later releases are more complete versions of the software to be produced.



How does this work for the Coding stage? One plans for coding by deciding who will code which module, and estimating how much time will be taken for the same. Next, evaluating alternatives for coding might include looking at the code generation tools that are available or using a different algorithm. Risk analysis could highlight facts like, the compiler version is not as recent as the version the client has or that the programmers are proficient in C but not proficient in SQL. Finally the software engineer would actually do the real work of writing code.

The Spiral Model…

- For example, in coding, we can plan who will code what and time required for coding.
- A different algorithm can be considered as an alternative.
- Risks can be non-availability of the latest compiler or inexperienced development team.
- The real execution of coding is done only after considering all these.

Pros and Cons – Spiral Model

- The Spiral Model gives room for risk analysis.
- It is more realistic as it is iterative.
- It requires good expertise in risk analysis and project management.

Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

Requirement Analysis and Design

Requirements Analysis and Design

- Requirement Analysis produces a specification of what a system should do.
- Design phase refines the findings from the analysis to give a clear picture of how the solution has to be implemented.

Requirement Analysis The intention of this phase is to provide a clear definition of the requirements of the system. The relationship between the inputs, the processes and the outputs are also specified. This is a very important phase because if the customer requirements are not clearly understood, the ambiguity can percolate to the other phases of development. Analysis may be performed in two contexts:

- A software system already exists, and the system has to be enhanced with new features
- There is no existing system and a new system must be built for the customer

Design The details of how the actual software implementation will be done are specified. The output of this phase is the design documents specifying the complete software architecture, data design, procedural detail and *interface design*

Structured Systems Analysis and Design

- SSAD is a methodology for doing Analysis and Design.
- SSAD considers any system as a composition of two components - Data and Processes which are considered separately
- The findings of SSAD need to be placed on paper. This is called modeling
- Building without modeling is difficult and expensive
- Building without modeling is similar to trying to construct a house with out drawing the plan

Why use models for analyzing requirements? These models are usually achieved by using graphical notations and principles which helps to minimize redundancy and also makes it easy to understand the requirements.

The following sections take a closer look at these models.

SSAD

- Processes are modeled using a technique called Process Modeling
- Process Modeling is used for understanding the requirements
- Data is modeled using Data Modeling
- Data Modeling is not only used for understanding the requirements but also for design

Analysis are used to achieve the following goals:

•Reduction in the risk of misinterpretation

•Resolution of all *ambiguous statements* in the requirements

•Easy incorporation of changes to requirements

•Enables the analyst to ask the right questions

The Analysis Phase results in the requirements specification, consisting of the following deliverables:

Process model: The process model takes into account the business needs of the system.

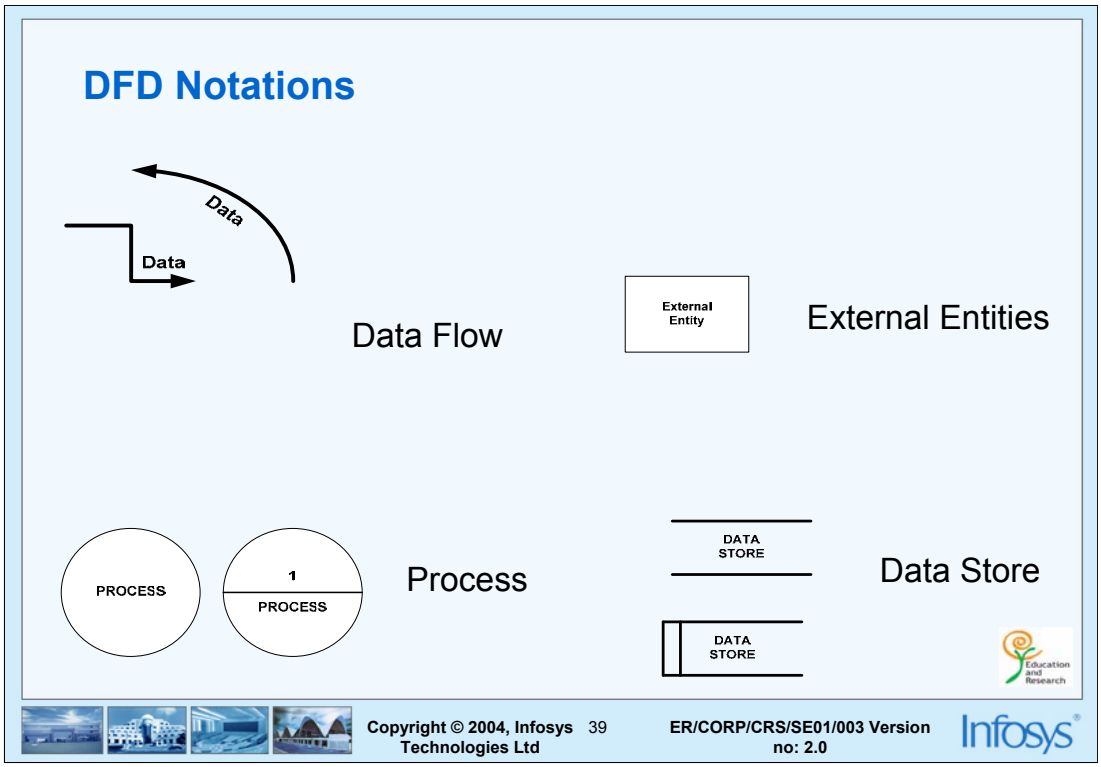
Process specification: The process specification is used to describe the process model once it has reached its final level of refinement.

*Data model: D*efines all the data objects that are processed within the system, the relationship between them and other information that is important for the relationship.

Data Dictionary: Dictionary is only a collection of data element definitions.

Process Model

- Analysis tool used for Process Model is DFD – Data Flow Diagrams
- The DFD is a graphical representation of the various processes that make up the system.



DFD are represented in different levels. The first diagram represents the system as a whole and is called Context-Level or Level 0 DFD. The Context level DFD represents the system as a whole, defining the external entities to the system. It clearly defines the system's inputs and outputs.

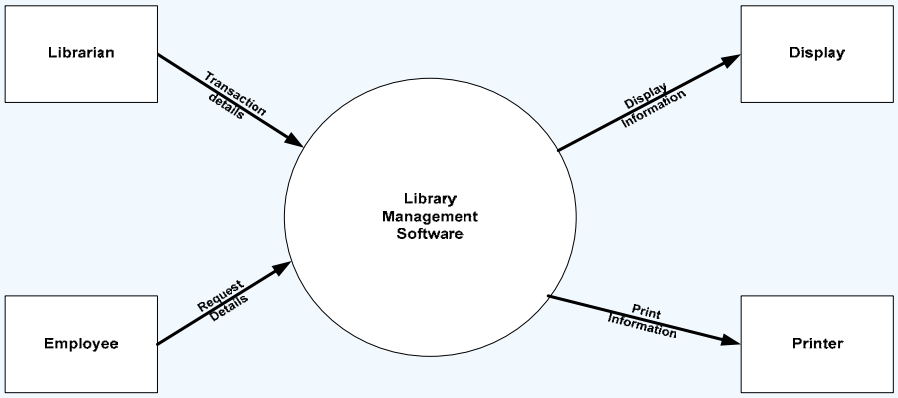
The next level - Level 1 will refine the context diagram providing more detail.

Level 2 DFD would show the refinement of a process in Level 1 DFD. The external entities are optional from Level 2 onwards.

As the levels increase each process becomes simple and well-defined, which clearly represents a business process.

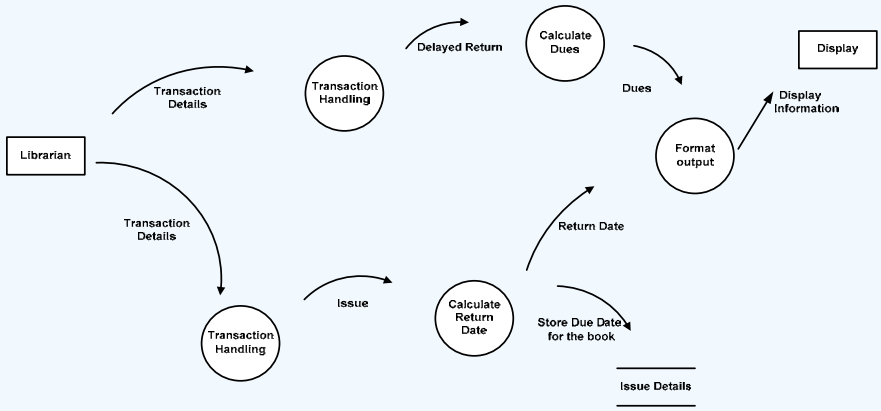
It is important to note that the Process model though it greatly helps to understand requirements, it may not be carried forward to the design phase. It can be used as a reference to help in design, but processes in terms of the programming language and the system being developed has to be identified separately in the design phase.

Level 0 DFD



Context Level DFD or Level 0 DFD

Level 1 DFD



Level 1 DFD

Process Specification

- The process specification is a statement of how the business needs are going to be met.

```
begin
 for every transaction in the Library
 begin
   if librarian has to issue a book
   {
      - Find if any books are not returned by employee
      - Calculate due date for the book to be issued
      - Update the information in the system - Output the details
   }
   if employee is returning a book
   {
      - Find if any books are not returned by employee
      - if book is returned after due-date
                              * Calculate amount due
      - Update system with amount collected
      - Output the details
   }
 end
end
```

The content of the process specification can include text, *pseudocode* **.**diagrams or charts. A process specification for each process in the refined DFD, will serve as a reference for design of the software component.

Data Model

Data Object

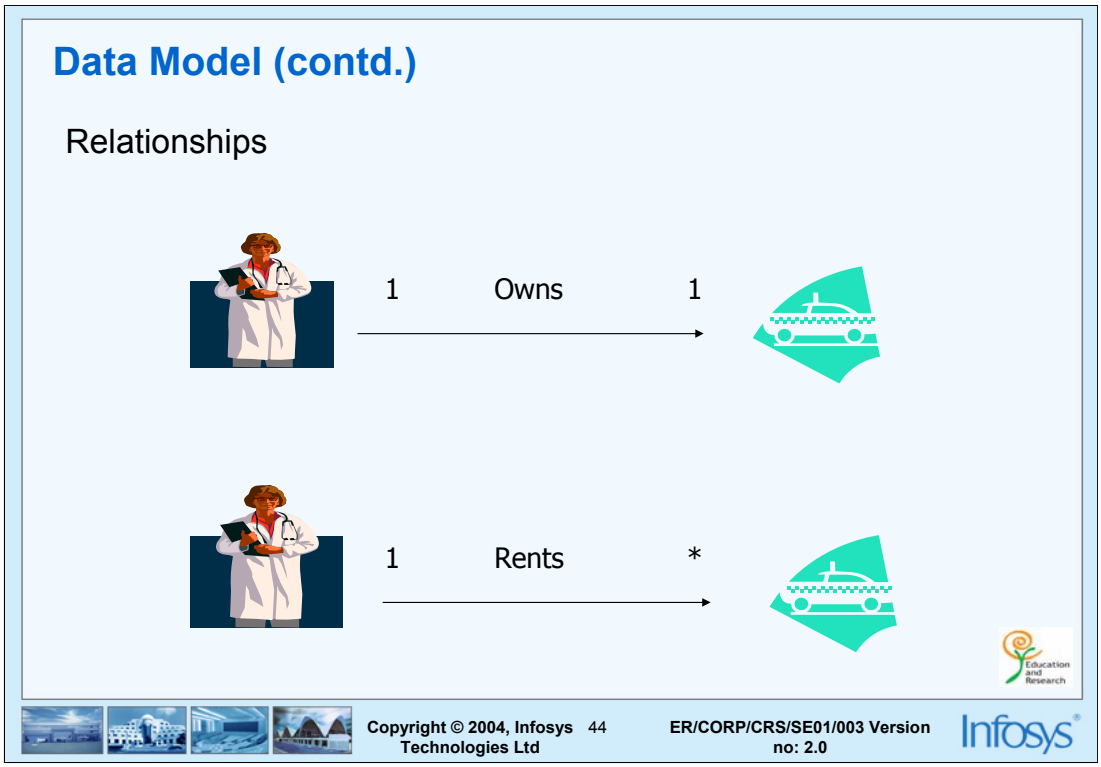
PERSON

CAR

Data Attributes

Car Model
Car Color
Car number

(Maruthi Zen,Black, KA 0489) is an instance



Data objects may be connected to each other in different ways. These connections are termed as relationships. To describe relationships, one must understand the role of the data objects within the system being created.

For example, consider two data objects "person" and "car". The relationship between a person and a car can be established in many ways.

A person owns a car

A person rents a car and so on

The relationship "owns" and "rents" define the relevant connections between a person and a car.

Cardinality refers to the number of occurrences of a data object that relates to a number of occurrences of another data object.

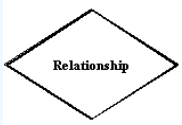
Modality defines a relationship by specifying whether or not the relation is optional. The modality of a relationship is 0 (zero) if it is optional and 1 if it is mandatory.

E-R Diagrams

An E-R diagram is a graphical tool used to model data; the modeling technique is called E-R Modeling.

An entity set is an object or concept about which you want to store information.

Attributes are the properties or characteristics of an entity.



Relationships illustrate how two entities are related

Six steps are followed to generate E-R Models:

Identify Entities – An entity can be classified into five categories; roles, events, locations, tangible things or concepts

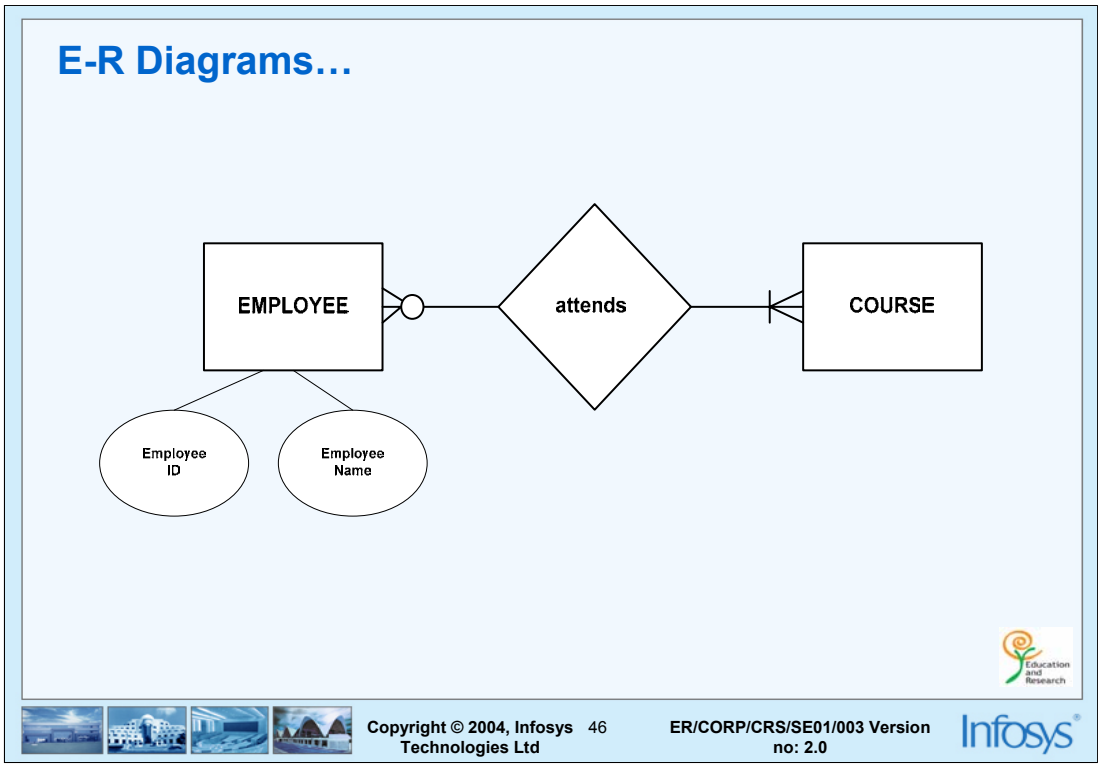
Identify Relationships – Identify the natural relationship, the cardinality and modality between the entities

Identify the P*rimary attributes*

Identify other relevant attributes

Draw the E-R diagram with all the identified entities, relationships and attributes

Review your results with your customers - look over the list of attributes associated with each entity to check for omissions or any modifications



Consider the scenario where an employee takes courses in E&R (Education and Research wing of Infosys). Follow the steps below to draw up a model.

- First the entities are identified. The entities in this case are “Course” and “Employee”. These will be represented as boxes in the E-R diagram.
- Next the relationships have to be determined. The relationship here is “Employee takes Courses”. The diamond shape is used to represent the relationship named “attends”.
- Next, the primary and other relevant attributes of the entities are identified. The Employee entity can have two attributes, employee ID and employee name. There can be more attributes like, address and telephone number of employee. Similarly the course entity can also have attributes like course ID, course name and description. The primary attribute for Employee could be employee ID and for Course could be course ID.
- Now, the E-R diagram must be drawn.

Having drawn the E-R diagram, the analyst now has a better understanding of the problem and starts contemplating different options. What is the minimum number of courses that an employee must take? Is there limit to the number of courses? To clarify these, the analyst meets the customer again and gets more information.

In this example, an employee attends one or more courses. These are represented in the E-R diagram by looking at the symbol at the “Course” end of the line. The symbols here indicate that an employee attends at least one course (mandatory). Questions could be raised with reference to the validity of this statement. Does every employee need to take a course? If the company recruits a new Vice President, now does she/he have to take a course? Is it optional for an employee to take a course? Well, the answer here is, yes it is optional. So the symbol must be wrong!

On clarifying the requirements given by the customer for the system, it is understood that the system needs to maintain only the information of employees who have taken at least one course.

So it turns out that the symbol on the “Course” end of the line is correct.

Data Dictionary

- A data dictionary defines the basic organization of data.
- The data dictionary is only a collection of data element definitions.

COURSE = **course id + course name + description**

course name = name of a course

EMPLOYEE **= emp id + name + address + telephone number + mark**
mark = mark obtained by an employee in a test

Design Phase

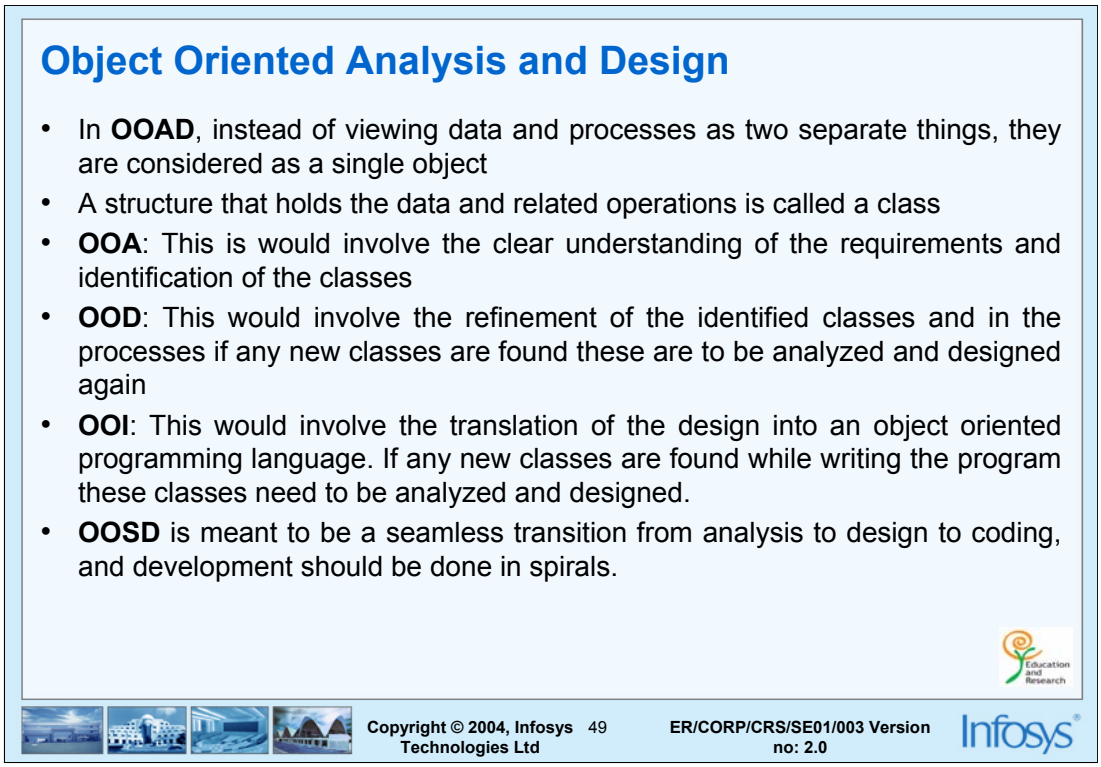
- Design produces a database design and a program design.
- Tools are available that can convert a Data Model into a database design

The Design Phase in SSAD produces the database design and program design. In effect the design phase refines the findings of the analysis phase to a greater extent, to ensure that it is easily understood by the software engineer.

The database design is arrived at by applying data modeling techniques to the E-R diagram produced during the analysis. This is discussed in detail in the RDBMS course.

The program design consists of flow charts and/or a set of program specifications. The flow chart graphically shows details of the sequence, conditions and iterations in each of the modules.

The program specification contains details on every module to enable the software engineer to write or generate code in the specified language. It describes the design and implementation constraints for every module that constitutes the software.



The focus of Requirement Analysis and Design phases of software development is to clearly analyze and understand the customer's requirement and then convert it into a design that will help in the creation of the software. When it is translated to Object Oriented Software Development (OOSD), it would be the following:

Object Oriented Analysis (OOA)

This is would involve the clear understanding of the requirements and identification of the classes.

Object Oriented Design (OOD)

This would involve the refinement of the identified classes and in the processes if any new classes are found these are to be analyzed and designed again.

Object Oriented Implementation (OOI)

This would involve the translation of the design into an object oriented programming language. If any new classes are found while writing the program these classes need to be analyzed and designed.

As it can be seen from the above definitions, OOSD has to be done iteratively, to achieve a clear understanding of the requirements, to establish a good design and to implement the same. OOSD is meant to be a seamless transition from analysis to design to coding, and development should be done in spirals. Each spiral should refine the earlier as depicted in Figure 3-9. With every iteration the final outcome should be in a better shape.

OOAD…

Class	Combines the data and procedures required to describe the content and behavior of real world entities
Attributes	Attributes are properties that describe a class
Operations	Operations are also called methods; these are the representations of one of the behaviors of a class
Objects	Instances of a particular class

Object Oriented Analysis

- OOA is to understand requirements and determine the classes with relationships and operations associated with them.
- Requirements must be gathered from the customer and analyzed.
- Classes must be identified with attributes and operations (methods).
- Relationship between the classes should be represented.

Use Case Driven Methodology

- It shows the user perspective of the system.
- Identify Actors
- Develop use cases
- Use Case must address the following :
  - Who are the actors?
  - What are the actor's goals?
  - What preconditions should exist before the story begins?
  - What exceptions can be considered as the story is described?
  - What system information will the actor access, change, delete or create?
  - What information does the actor require from the system?

Use Case Diagram

- A use-case template can be used for a detailed description of the scenario.

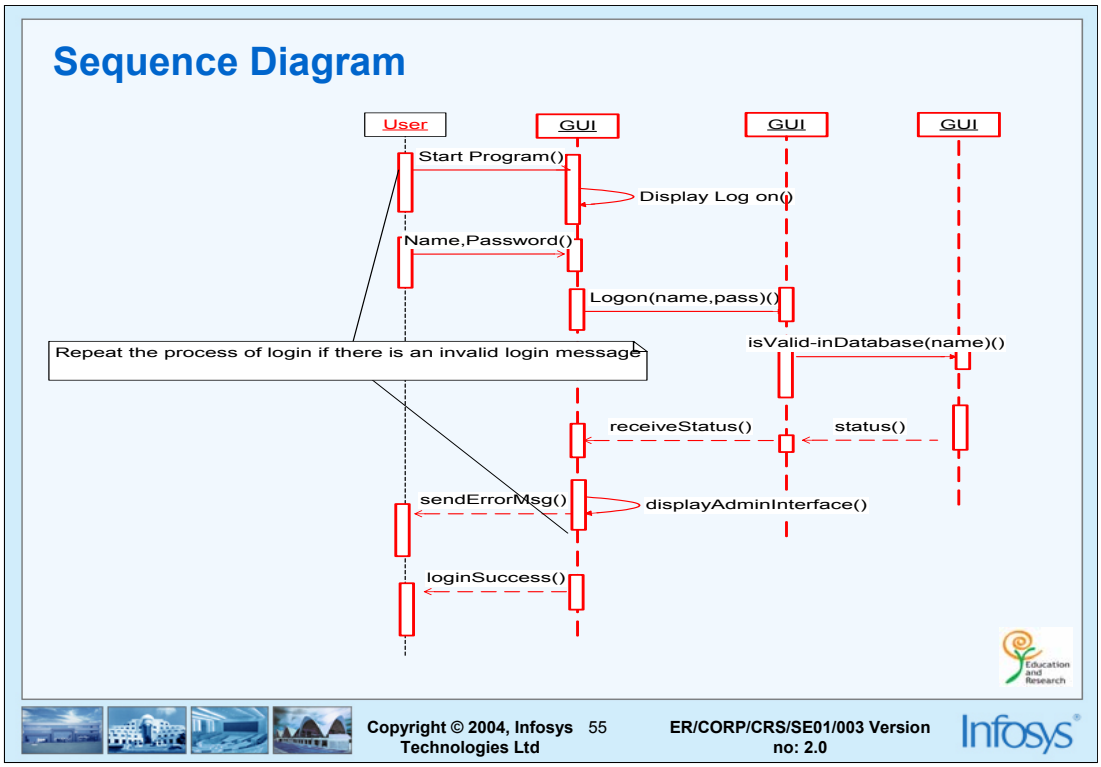
Use-case	Register for Course
Actors	Trainee and ILite
Objective	The trainee gets a list of available courses The trainee registers for a course
Precondition	The trainee has to be an employee of the company
Scenario	Trainee gets a list of courses from ILite Trainee registers for a course Trainee attends the course
Exceptions	The trainee registers but does not attend the course The course is cancelled

Consider a software application that helps trainees to register for courses in the organization. One of the requirements could be that the trainee uses the software to register for a course that is listed in *ILite*

To write the use-case the actors must be identified first. The actors here are the “Trainee” and “ILite”. One of the many use-cases for the system could be “Register for Course”. The basic use-case should present a high-level story that describes the interaction between the actors and the system. One way of describing the use-case is to use a template.
A use-case diagram can also be used to describe the interaction between the actors and the system. It is not always necessary to have a use-case diagram, but graphical representation can facilitate the understanding when a particular scenario is complex. In the figure, the stick person represents an actor (usually human user) and the ellipse represents a use-case. If the actor is another system then a rectangular box is used. The lines indicate the actor’s participation in the use-case.

Object Oriented Design

- Sequence Diagram/Time Line Diagram is the tool used to design the system
- OOA and OOD are iterative tasks, as the analyst goes through both the phases recursively to get clear and better requirements.



Sequence diagrams have to be drawn for all the identified use-cases to describe the sequence of events that happen in each use-case scenario.

The collection of all sequence diagrams will explain the participation of all identified objects and the sequence in which the different methods are invoked for all use-cases.

OOA and OOD are iterative tasks, as the analyst goes through both these phases recursively the requirements get clearer and better. This is to say that in the design phase the analyst could stumble on new objects that have to be analyzed.

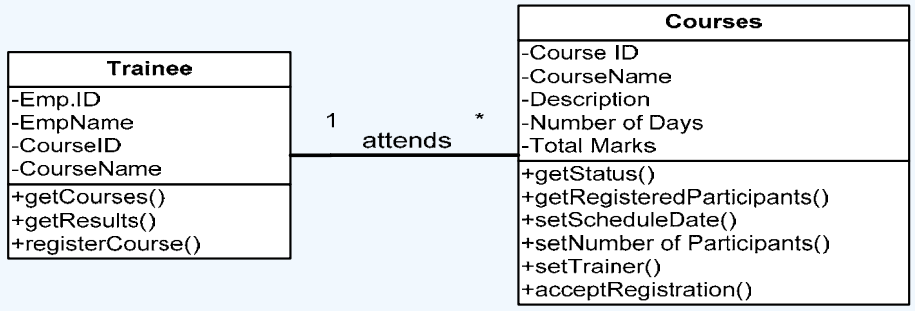
For example while drawing the above sequence diagram the designer would probably think of adding another object "Marks Card". This is again analyzed and then designed.

Class Diagrams

- All the noun phrases are listed as classes, in which the duplicated are removed after refinement.
- Specify the attributes that describe the class.
- Identify the behavior of the object as “operations”.
- Identify the relationships between classes.

Using all the use-cases and sequence diagrams, the requirements understood and many of the objects would have been identified. Now the analyst needs to group the objects into classes.

Class Diagrams…



Class Diagram Depicting Cardinality

Class Diagrams: A class is represented by a rectangle divided into three portions. Each of the portions is for a specific representation:

- The name of the class
- The Member variables in the class
- The Member functions or Methods in the class

The activities that are performed during OOAD would be the following:

Draw Use Cases to understand requirements

Use Sequence Diagrams to elaborate scenarios that are identified in the Use Cases

Identify the Classes involved

Class diagrams: Draw class diagrams

Relationships: Identify relationships between classes

Attributes: Identify the attributes of the classes

Operations: Decide on the operations that the class needs to perform.

There has to be review after each of the above steps. The steps are iterative and all notations need updating every time a new class or attribute or operations is identified. There can also be updates if the review identifies redundant or unwanted classes, attributes or operations, these have to be removed

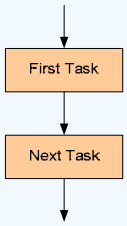
Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

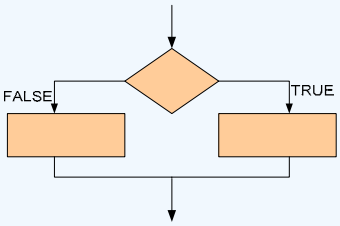
Software Construction

Structured Programming

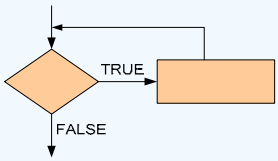
- Earlier people wrote programs that were not maintainable.
- Software Engineering analyzed the problems with these programs and introduced a disciplined way of programming called Structured Programming.
- Minimize the use of goto Statements - Use only sequence, condition and repetition.
- Write readable and modular programs.



SEQUENCE



CONDITION



REPETITION

Code Reviews

- Reviews “purify” the software engineering process, by filtering out defects.
- Self review is done by the programmer to ensure that there are no defects.
- Review can be an informal discussion between people.
- It can be a formal technical review where a presentation is given to a group consisting of the client and/or technical people.
- Reviews catch the error closest to their place of origin and makes the correction easy.
- Code Walkthrough is done at Infosys as a part of program review

Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

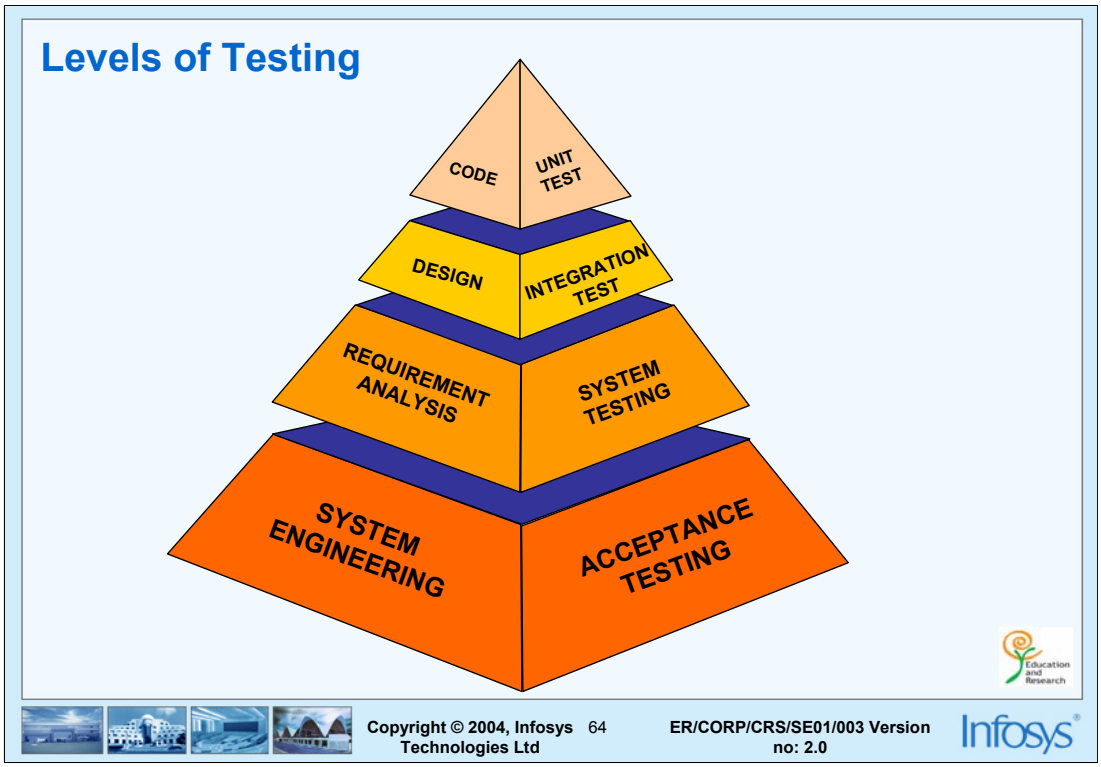
Software Testing

Software Testing

- Testing is for finding errors, not to demonstrate your software.
- It is better to identify a tester who has not developed the code
  - This is because programmer may not want to find error in the code he developed

Testing Terminologies

- **Error**: An error is an incorrect implementation of a system. Errors may exist in any phase of the software engineering process.
- **Fault:** A fault is an incorrect or unexpected behavior of a system in response to a real-world action. Faults result from errors in the system.
- **Failure:** A failure occurs when a system is prevented from achieving its mission. Failures result from faults.
- **Testing:** It is a process for analyzing software. It detects the differences between existing and required conditions and evaluates features of the software.



There are different levels of testing

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

Each of this level will test a phase of the software development life cycle

Unit Testing

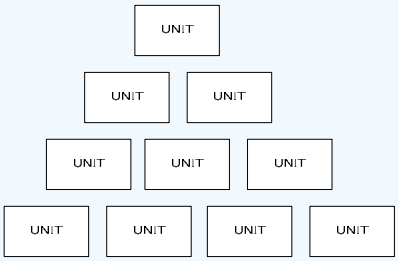
- In unit testing we check whether a particular module is implementing its specification.
- Unit testing checks whether coding is correct.

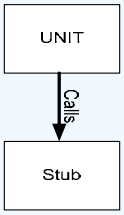
Integration Testing

- The modules that may work properly and independently may not work when they are integrated.
- Integration Testing will test whether the modules work well together.
- This will check whether the design is correct.
- Integration can be done in 4 different ways
  - Top Down
  - Bottom Up
  - Sandwich
  - Big Bang

Top Down Integration

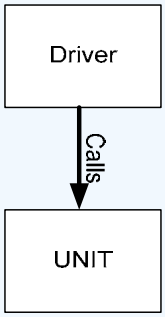
- In Top Down Integration, top level modules are developed and tested first, for testing this module we may require a dummy bottom level module – *stub.*





Bottom Up Integration

- In Bottom Up Integration, bottom level modules are developed and tested first, for testing this module we may require a dummy top level module – *Driver.*



Sandwich Integration

- Sandwich Integration combine both the previous techniques.
  - Instead of completely going for top down or bottom up, a layer is identified in between.
  - Above this layer we go for top down and below this layer bottom up.

Big Bang Integration

- The Big Bang Integration integrates all the unit tested module together at a go

System Testing

- System Testing will contain the following testing :
  - Functional Testing.
  - Performance Testing.
- Function Testing will test the implementation of the business needs.
- Performance Testing will test the non-functional requirements of the system like the speed, load etc.

System testing is the process of testing the completed software as a part of the environment it was created for. It is done to ensure that all the requirements specified by the customer are met. System testing involves functional testing and performance testing.

System Testing

- Performance Testing involves
  - Load Testing
    - Testing with many users accessing the system at the same time.
  - Endurance Testing
    - Testing for a long time for reliability.
  - Stress Testing
    - Testing to identify the number of users the system can handle at a time before breaking down.
  - Spike Testing
    - The system is stressed suddenly for a short duration.

- System testing will test the Customer Requirements

Acceptance Testing

- Acceptance Testing is done by the customer to test whether the system is meeting the requirements

Regression Testing

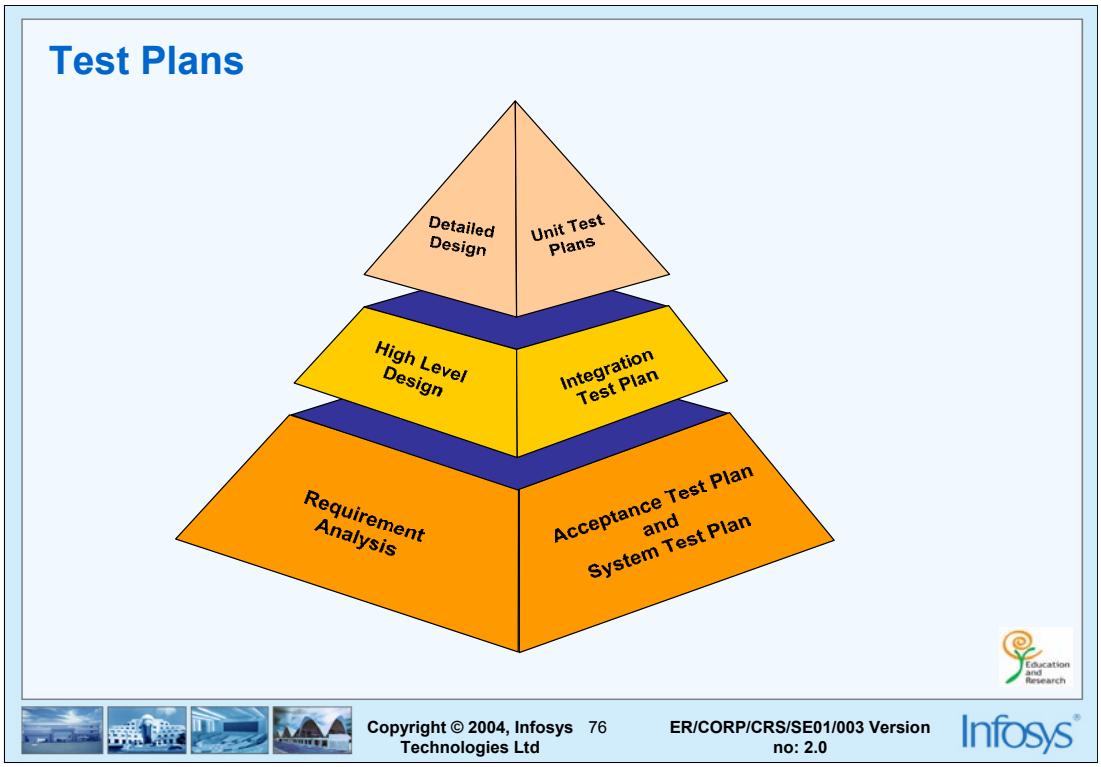
- The most frustrating experience for a user is to find that a feature that he has been using till yesterday has disappeared today.
- When a bug is fixed, ensure that the new version will pass all the test cases that it passed earlier.
- Maintenance projects require regression Testing.

Forms of Testing

- Verification Testing
  - This is the process of answering the question “Are we building it right?”
  - Unit And Integration Testing
- Validation Testing
  - This is the process of answering the question “Are we building the right thing?”
  - System And Acceptance Testing

Software testing is very often referred to as software verification and validation (V&V). Verification refers to the activities which ensure that software correctly implements a specific function. Validation refers to a different set of activities to ensure that software that has been built is traceable to the customer requirements.

Some believe that all testing is verification and that validation is only when the requirements are reviewed and approved by the user when the system is operational. Others are of the opinion that unit and integration testing are verification and all higher levels of testing such as System Testing or Acceptance Testing are validation testing.



A Test Plan is an exhaustive list of test cases. Testing is not an unplanned or informal process. It is based on well formulated test plans. Test plans are also reviewed before being used. Test plans should be written as early as possible in the software development life cycle.

The Acceptance Test Plan and System Test Plan are written immediately after requirements are gathered and analyzed. Both are artifacts that are finalized in the Requirement Analysis phase.

During High Level Design (HLD), the modules of the project and their interfaces are defined. So after HLD the Integration Test Plan is written.

All the program specifications are prepared during Detailed Design (DD). After Detailed Design, Unit Test Plans are written for each module.

Test Plans

- The System Test Plan and Acceptance Test Plan are also developed along with requirements analysis
- System Testing and Acceptance Testing are to test whether the software meets the users requirements.
- Unlike System Testing, Acceptance Testing is done by the customer.
- Integration Test Plan will be developed along with the High Level Design
- Unit Test Plan will be developed along with the Low Level Design

Philosophy of Test Case Design

Functional Testing	Structural Testing
Test cases selected based on the functional specification	Test cases selected based on software structure or implementation
Also termed as Black Box Testing	Also termed as White Box Testing
Example: System Testing	Example: Unit Testing

If we have some knowledge of the internal structure of the program, it is called a Grey Box Testing. Integration Testing can be Grey Box Testing.

Test Case Design

- A test plan should contain test cases i.e. each test case has the intended input and the expected output
- In Exhaustive Testing, we test the software with all possible inputs!
  - It is very obvious that this method is not practical.
- For designing the test cases, we use several methods
  - In Random Generation, we generate several test cases randomly.
    - It is very simple and can be easily automated.
    - But this is not an efficient method.

Test Case Design…

- In Equivalence Partitioning, we divide the inputs into different classes, so that if one member in a class is valid all are valid or the other way.
  - For example, if the requirement is that the input should be between 1 and 999, three equivalence classes are :
  - 1<= items <= 999 Valid Class
  - items < 1 Invalid Classes
  - items > 999

Test Case Design…

- In Boundary Value Analysis, we choose test cases that explore the boundary conditions.
  - Many errors are of the type “off-by-one”.
  - In the above example 0, 1, 999 and 1000 are boundary values.

Test Case Design…

- In Logic Coverage, first we define the coverage criteria.
  - Examples for coverage criteria are as follows :
    - Every statement should be executed at least once.
    - Every branch should be executed at least once.

Test Case Design…

- Error Guessing is an ad hoc approach guided by intuition and experience.
  - In a case where the program is writing into a file, the following can be some of the test cases
    - Having an empty file.
    - Having a file that is full.
    - Not at all having a file.
    - Having a file with read permission only

Alpha and Beta testing

- For software products like MS Word, a formal acceptance testing cannot be conducted
- Alpha Testing is a simulated acceptance testing conducted in the developer's environment
- The Beta Testing is conducted in the real world environment

The Alpha test is conducted in the developer's environment by the end-users. The environment might be simulated, with the developer and the typical end-user present for the testing. The end-user uses the software and records the errors and problems. Alpha test is conducted in a controlled environment.

The Beta test is conducted in the end-user's environment. The developer is not present for the beta testing. The beta testing is always in the real-world environment which is not controlled by the developer. The end-user records the problems and reports it back to the developer at intervals. Based on the results of the beta testing the software is made ready for the final release to the intended customer base.

Test Automation

- Testing can be automated with the help of testing tools.
- LoadRunner and TestManager are examples of testing tools.

The testing procedure requires the tester to repeat many tests. When the testing procedure needs to be repeated for many cycles it becomes a monotonous task. The testing process can be automated and tools can be used to do this work. They remove the monotony and make testing an easier task for the tester.

This can also be used to measure a lot of parameters like bandwidth usage, response time, data transferred etc. Tools to automate tests can be developed within the company or they can be bought.

Thank You!