Software Engineering Introduction & Software Requirements Analysis & Specifications UNIT I

Introduction to Software Engineering

Learning Objectives

- What is Software
- Documents
- Software vs. Hardware Characteristics
- Types of Software
- Good Software
- Need for Software Engineering
- Software Crisis
- Software Engineering

Learning Objectives..

- Quality issues of Software
  - Cost
  - Late schedule
  - Software Quality
- CASE
- Software Myths
- Software Process
- Terminologies

 U1. 4

What is Software?

- Software is a set of items or objects that form a "configuration" that includes
  - programs
  - documents
  - data ...

Documents

Consist of different type of manuals

- Documentation manuals
- Operating procedure manuals

 U1. 5

Documentation Manuals

- Analysis Specifications
  - Formal specification
  - Context diagram
  - Data flow diagrams
- Design
  - Flow charts
  - Entity Relationship Diagrams
- Implementation
  - Source code listing
  - Cross reference listing
- Testing
  - Test data
  - Test results

 U1. 6

Operating Procedural Manuals

- Consist of instructions to setup and use the software system and instructions on how to react to system failures
- User manuals
  - System overview
  - Beginner's Guide Tutorial
  - Reference Guide
- Operational manuals
  - Installation Guide
  - System Administration Guide

 U1. 7

The Nature of Software

Software is intangible
- Hard to understand development effort

Software is easy to reproduce
- Cost is in its *development*
  - ✓**in other engineering products, manufacturing is the costly stage**

The industry is labor-intensive
- Hard to automate

 U1. 8

The Nature of Software...

Chances of Hacking
- Quality problems are hard to notice

Software is easy to modify
- People make changes without fully understanding it

Software does not 'wear out'

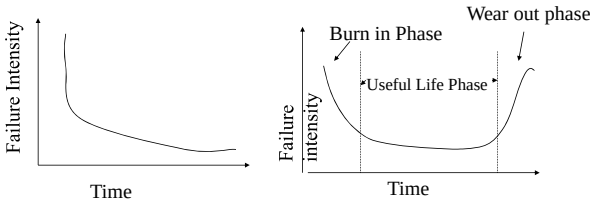
✓**in ways that were not anticipated, thus making it complex**

- Software doesn't wear out
- Software is not manufactured

 U1. 9

Software Characteristics

- Reusability of components
- Software is flexible



 U1. 10

Types of Software

Custom
- For a specific customer

Generic
- Sold on open market
- Often called
  - ✓ **COTS (Commercial Off The Shelf)**
  - ✓ **Shrink-wrapped**

Embedded
- Built into hardware
- Hard to change

- **System Software**
- **Application Software**

 U1. 11

Application Software

Real time software
- E.g. control and monitoring systems
- Must react immediately
- Safety often a concern

Data processing software
- Used to run businesses
- Accuracy and security of data are key

Some software has both aspects

 U1. 12

Applications Software

- Embedded software
- Business software
- Personal computer software
- Artificial Intelligence software
- Web based software
- Engineering and scientific software

 U1. 13

Attributes of Good Saoftware

- **Functionality**
  - to meet stated and implied need
- **Usability**
  - to be understood, learned and used
- **Reliability**
  - To maintain a specified level of performance
- **Portability**
  - To be adapted for different specified environment
- **Maintainability**
  - To be modified for the purposes of making corrections, improvements, or adaptations
- **Efficiency**
  - To provide appropriate performance relative to the amount of resources used

 U1. 14

Software Crisis

- **Failure to master the complexity of software results in projects that are late, over budget and deficient in their stated requirements**
- **Software crisis arise because:**
  - Informal methods to specify what software should do
  - Software tools are complicated and unreliable

 U1. 15

To Avoid Software Crises

- need to design software properly
  - To ease the verification
- need to maintain and upgrade software at a lower cost
  - Require Proper Documentation
- need to re-use components.
  - needs to precisely document what the software does
- important to have precise languages and tools
  - enable good documentation and communication of ideas at all stages
- standardized notations used to express specifications and designs
  - workers on a large project can collaborate without misunderstanding.

 U1. 16

What is Software Engineering?

- The process of solving customers' problems by the systematic development and evolution of large, **high-quality software** systems within **cost**, **time** and **other constraints**
- Solving customers' problems
  - Goal of software engineering
  - Sometimes the solution is to buy, not build
  - Adding unnecessary features does not help solve the problem
  - Software engineers must communicate effectively to identify and understand the problem

 U1. 17

What S/W Engineering is and is not..

- Software engineering is concerned with "engineering" software systems, that is, building and modifying software systems:
  - on time,
  - within budget,
  - meeting quality and performance standards,
  - delivering the features desired/expected by the customer.
- Software engineering is not…
  - Just building small or new systems.
  - Hacking or debugging until it works.
  - Easy, simple, boring or even pointless!

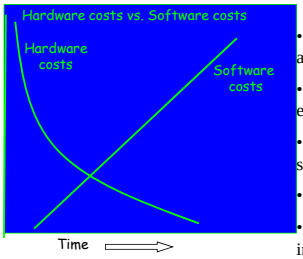
 U1. 18

S/W Engineering and Computer Science

- Computer science is concerned with theory and fundamentals;
- Software engineering is concerned with the practicalities of developing and delivering useful software
- Computer science theories are currently insufficient to act as a complete underpinning for software engineering

Software Development Costs

What can you infer from the graph?



Software costs are increasing as hardware costs continue to decline

- Hardware technology has made great advances
- Simple and well understood tasks are encoded in hardware
- Least understood tasks are encoded in software
- Demands of software are growing
- Size of the software applications is also increasing
- Hence "the software crisis"

Why are Software Projects Late? …

Does effort necessarily == progress?

Is one man working six months equal to six men working one month?

Unit of man-month implies that men and month are interchangeable.

- True only when a task can be partitioned among many workers
- with no communication between them.
- For sequential tasks, more effort has no effect on the schedule.
- Many tasks in software engineering have sequential constraints.

Our estimating techniques fallaciously confuse effort with progress, hiding the assumption that men and months are interchangeable.
- Fred Brooks, *The Mythical Man-Month*

Why Software Projects are Late?...

Managers do not monitor progress effectively

Schedule slips day-by-day.

Day-by-day slips are harder to recognize,
harder to prevent and harder to make up.

How does a software project get to be a year late?
One day at a time!
Fred Brooks, *The Mythical Man-Month*

 U1. 22

Why are Software Projects Late ?...

When we recognize slippage, should we add more people?

- Most tasks require communication among workers.
- Communication consists of:
  - Training.
  - Sharing information (intercommunication).
- Training affects effort at worst linearly, with the number of people.
- For *n* workers, intercommunication adds *n(n-1)/2* to effort.
  - If each worker must communicate with every other worker.
- Adding more people to an already late project is usually like "Adding gasoline to fire!"

Brooks' Law:
Adding manpower to a late software project makes it later.
Fred Brooks, *The Mythical Man-Month*

 U1. 23

Software Myths…

Myth : Sufficient literature full of standards and procedures for building the software

- Reality
  - –Is the literature is really used?
  - –Are software practitioners aware of its existence?
  - –Does it reflects modern software engineering practice?
  - –Is it complete?
  - –Is it streamline to improve time to delivery while still maintaining the focus on quality?

 U1. 24

Software Myths…

- Myth **:** Software is easy to change
- Myth **:** Computers provide greater reliability than the devices they replace
- Myth **:** Testing software or "proving" software correct can remove all the errors
- Myth **:** Reusing software increases safety
- Myth **:** Software can work right the first time

 U1. 25

Software Myths cont..

•Myth **:** Software with more features is better software
•Myth **:** If we get behind schedule, we can add more programmers and catch up Propagated misinformation and confusion
•Myth :According to customer A general statement of objective is sufficient to begin writing programs- we can fill in the details later
•Myth : Once we write the program and get it work, our job is done
•Myth : Until I get the program running I have no way of assessing its quality

•Myth : The only deliverable work product for a successful project is the working program
•Myth : Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down

 U1. 26

Software Process

- **Process :**Anything that operates for a period of time, normally consuming resources during that time and using them to create a useful result
- A set of activities whose goal is the development or evolution of software
- Generic activities in all software processes are:
  - **Specification** - what the system should do and its development constraints
  - **Development** - production of the software system
  - **Validation** - checking that the software is what the customer wants
  - **Evolution** - changing the software in response to changing demands

 U1. 27

Software Process Model

- A simplified representation of a software process, presented from a specific perspective
- Examples of process perspectives are
  - Workflow perspective - sequence of activities
  - Data-flow perspective - information flow
  - Role/action perspective - who does what
- Generic process models
  - Waterfall
  - Evolutionary development
  - Prototyping
  - Rapid Application development
  - Integration from reusable components

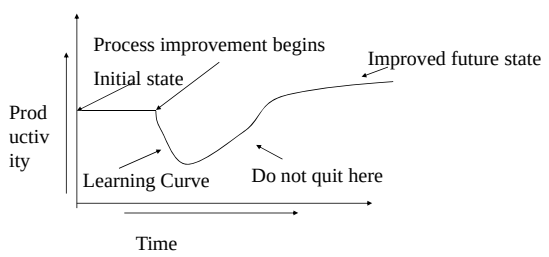
 U1. 28

Difficulty in S/W Process Improvement

- Not enough time
- Lack of knowledge
- Wrong Motivations
- Insufficient Commitment

 U1. 29

Process Improvement Learning Curve



 U1. 30

Some Terminologies

Deliverables and Milestones
- Deliverables generated during software development
- Milestones are the events

Product & Process
- What is delivered to customer
- Process is the way to produce software

Measure, Metrics and measurement
- Measure: quantitative indication
- Measurement: act of evaluating a measure
- Metric: degree to a given attribute

 U1. 31

Some Terminologies cont..

Software Process and Product Metrics
- Process: Productivity, quality, failure rate
- Product: size, reliability, complexity, functionality

Productivity (production per unit of effort) and Effort
- Quantity of output
- Period of time
- PMs LOC/PM

Module and software components
- Module: Work assignment for an individual developer
- Component: An independently deliverable piece of functionality providing access to its services through interfaces

Generic and customized software products

 U1. 32

What we Learnt

- ✓ Software Definition
- ✓ Different Software Documents
- ✓ Software vs. Hardware Characteristics
- ✓ Types of Software
- ✓ Property of Good Software
- ✓ Need for Software Engineering
- ✓ Software Crisis
- ✓ Software Engineering
- ✓ Software Myths
- ✓ Software Process
- ✓ Terminologies

 U1. 33

Software Life Cycle Model

Set of Processes that Results in Software Products

Learning Objectives

- Inherent Problems with Software Development
- What Do Programmers Do?
- Definitions
- Software Development sub processes
- Generic life cycle phases
- Processes, Activities and Tasks
- Modeling Dependencies in a Software Lifecycle
- Generic software process models

Generic Software Process Models

- Build-and-fix model
- Waterfall model
- Prototyping model
- Incremental model
- V Model
- Spiral model
- RAD

Inherent Problems With S/W Development

Requirements are complex

- The client usually does not know all the functional requirements in advance

Requirements may be changing

- Technology enablers introduce new possibilities to deal with nonfunctional requirements

Frequent changes are difficult to manage

- Identifying milestones and cost estimation is difficult

 U1. 37

Inherent Problems with S/W Development..

There is more than one software system

- New system must often be backward compatible with existing system ("legacy system")
- Phased development: Need to distinguish between the system under development and already released systems

 U1. 38

What Do Programmers Do?

- The Software Crisis led to a push to improve the way we develop software.
- Before we could do this, it was necessary to understand *how* software is developed.
- People soon recognized that what was commonly known as "programming" actually consisted of many more activities than just "programming".

 U1. 39

Definitions

- Software lifecycle modeling
  - Attempt to deal with complexity and change
- Software lifecycle
  - Set of activities and their relationships to each other to support the development of a software system
- Software development methodology
  - A collection of techniques for building models - applied across the software lifecycle

Definitions..

- Series of identifiable stages that a software product undergoes during its lifetime and these stages is called a **life cycle phase**
- Breaking software development down into a number of phases like these led to the idea of the Software Lifecycle
  - cf. butterfly's lifecycle (caterpillar, pupae, …)

S/W Development Sub Processes

Generating Request
- **System conception**

Discovering and documenting *what* the software system should do
- **Requirements Specification**

Deciding *how* the system is going to do it
- **Software Design**

Creating the software which implements it
- **Coding/Implementation/System Construction**
- **System Integration**

Software Development Activities cont..

Making sure that the software actually does what it is supposed to

- **Testing**
- **Software Quality Assurance**

Installing the software in the live environment

- **System Installation/System Conversion**

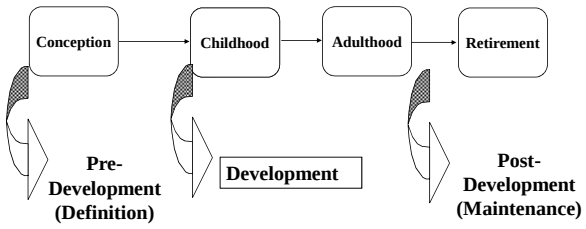
Keeping the software doing what it should

- **Software Maintenance/Evolution**

Phasing out the software when it is no longer of use

Generic Life Cycle Phases

Software construction goes through a progression of states



Pre Development Phase

Focuses on what?

- What information is to be processed?
- What functions and Performances are desired?
- What interfaces are to be established?
- What validation criteria are required to define a successful system?

Development Phase

Development phase focuses on the - how

- How data structure are to be designed?
- How Software architectures are to be designed?
- How procedure details are to be implemented?
- How the design will be translated in to a code?
- How testing will be performed?

Three specific steps in Development Phase:

- **Design**
- **Coding**
- **Testing**

 U1. 46

Post Development Phase

The Maintenance phase focuses on change that is associated with

- Corrective
- Adaptive
- Perfective
- Preventive

 U1. 47

S/W Development Activities

Requirements Analysis	**What is the problem?**	**Problem Domain**
System Design	**What is the solution?**	
Program Design	**What are the mechanisms that best implement the solution?**	**Implementation Domain**
Program Implementation	**How is the solution constructed?**	
Testing	**Is the problem solved?**	
Delivery	**Can the customer use the solution?**	
Maintenance	**Are enhancements needed?**	

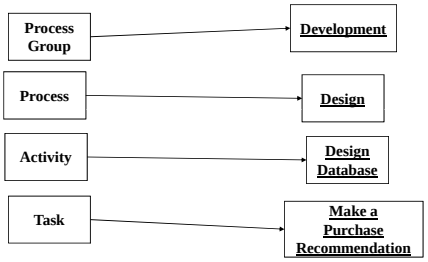
 U1. 48

Processes, Activities and Tasks

Process Group:
- Consists of Set of Processes

Process:
- Consists of Activities

Activity:
- Consists of sub activities and tasks



 U1. 49

Example

The Design Process is part of Development

The Design Process consists of the following Activities
- Perform Architectural Design
- Design Database (If Applicable)
- Design Interfaces
- Select or Develop Algorithsm (If Applicable)
- Perform Detailed Design (= Object Design)

The Design Database Activity has the following Tasks
- Review Relational Databases
- Review Object-Oriented Databases
- Make a Purchase recommendation
- ....

 U1. 50

Generic Software Process Models

- Build-and-fix model
- Waterfall model
- Rapid prototyping model
- Incremental model
- Spiral model
- RAD

 U1. 51

Software Engineering Life Cycle Models

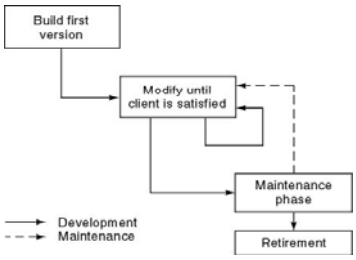
The life cycle model is selected based on:

- The nature of the Project and application
- The methods and tools to be used
- The deliverables that are required

 U1. 52

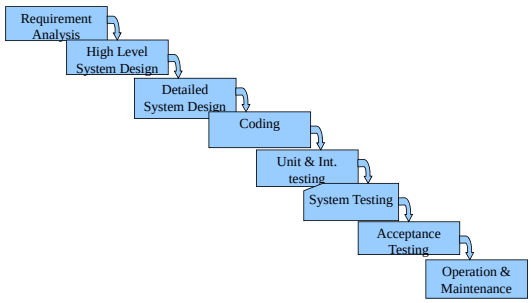
Build and Fix Model

- Problems
  - No specifications
  - No design
- Cost is high
- Maintenance is extremely difficult
- Totally unsatisfactory
- Need life-cycle model
  - "Game plan"
  - Phases
  - Milestones
- Work for small Projects



 U1. 53

Water Fall Model



 U1. 54

Typical Characteristics

- Sometimes called classic life cycle or the linear sequential model
- Each phase is considered to be completed with the production of certain *deliverables*
- For development of a large-scale system, each phase will typically be undertaken by a different set of people
  - different skills are required for each activity
- Communication between phases is principally by means of the deliverables

 U1. 55

Advantages

- Follows the usual engineering life cycle
- The Waterfall Model is simple to understand
  - even for non-technical managers!
- Its simplicity means that planning for a Waterfall development is relatively easy

 U1. 56

Disadvantages

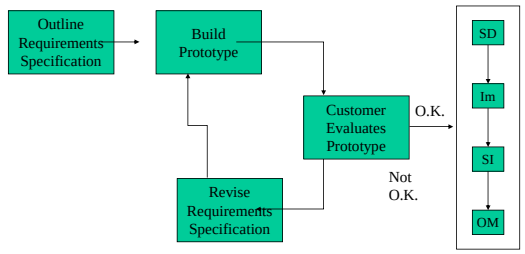
- Unfortunately, real projects are rarely so straightforward and sequential
- It is generally not possible to completely define (and freeze) all the requirements at the start of the project
- It is not until late in the process that something that can be demonstrated to the user is created
- In practice, "blocking states" occur, causing delays for some members of the team

 U1. 57

However ...

- … there is no question that even a poor model like the Waterfall model is significantly better than no model at all
- Variants of this sequential model are still widely used today, covering more or less of the activities that surround software development

Prototyping



Types of Prototypes

Illustrative Prototype/Exploratory

- Develop the user interface with a set of storyboards
- Implement them on a napkin or with a user interface builder (Visual C++, ...)
- Good for first dialog with client

Functional Prototype/Evolutionary

- Implement and deliver an operational system with minimum functionality
- Then add more functionality
- Order identified by risk

Types of Prototypes..

Exploratory Prototype

- Implement part of the system to learn more about the requirements.
- Good for paradigm breaks

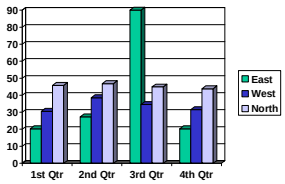
Evolutionary Prototyping

- The prototype is used as the basis for the implementation of the final system
- Advantage: Short time to market
- Disadvantage: Can be used only if target system can be constructed in prototyping language

 U1. 61

Advantages

- Prototype system is much easier to evaluate than a dry SRS document
- Customers and users can give immediate feedback on the requirements specification
- The implications of requirements can be judged within the "live" environment
- Construction of the prototype can help developers to make better decisions when implementing the full system



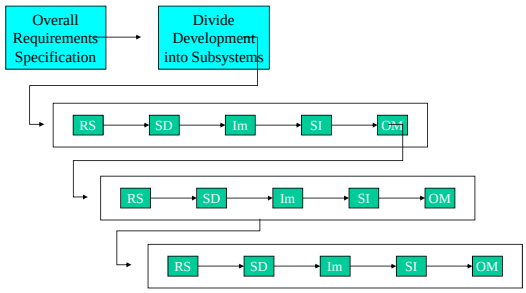
 U1. 62

Disadvantages

- The customer may think that the prototype is nearly the finished product
- As a result, the customer may not be prepared to wait another 6 months (or whatever) while the system is "rebuilt"
- Requires extensive participation and involvement of the customer, which is not always possible

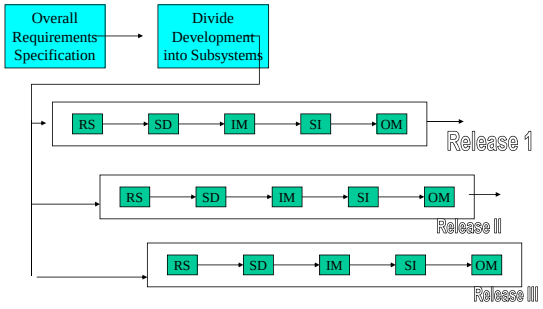
 U1. 63

The Incremental Model



 U1. 64

Iterative Enhancement Model



 U1. 65

Characteristics

- Requirements are prioritize
- Conducted in several cycles
- Usable product released at the end of the each cycle
- Each release providing additional functionality

 U1. 66

Advantages

- Markets created before development
- Core capabilities can be delivered "quickly" to customer
- Training and concepts can begin in an early release
- Core capabilities can be evaluated "quickly" by customer
- Frequent releases help developers to swiftly fix other unanticipated bugs
- Enables good use of available resources (e.g. staffing, hardware, customer time)
- A very safe approach
- Focus on different areas of expertise in different releases

 U1. 67

Disadvantages

- Every increment must be developed through to production standard
- Extra time spent on testing, documenting and maintaining a "temporary" product
- Can be difficult to split the problem up into appropriate increments
- Expensive

 U1. 68

Evolutionary Development Model

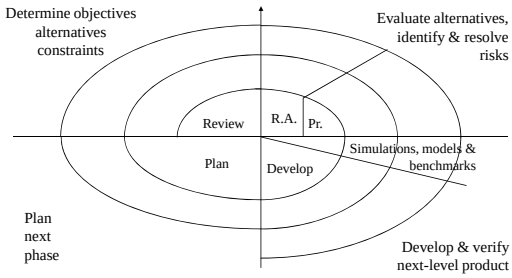
- Resembles iterative enhancement model
- Does not require a useable product at the end of each cycle
- Requirements are implemented by category rather than by priority
- Used when it is not necessary to provide a minimal version of the system quickly
- Useful for projects using new technology or complex projects

 U1. 69

Boehm's Spiral Model (1986)

 U1. 70

Boehm's Spiral Model (1986)



 U1. 71

Spiral Model Components

- Planning- Determination of objectives, alternatives, and constraints.
- Risk Analysis- Analyzes alternatives and attempts to identify and resolve the risks involved.
- Engineering- Development of the product as well as the incorporation of testing.
- Customer Evaluation- Assessment of the products of the engineering element.

 U1. 72

Characteristics

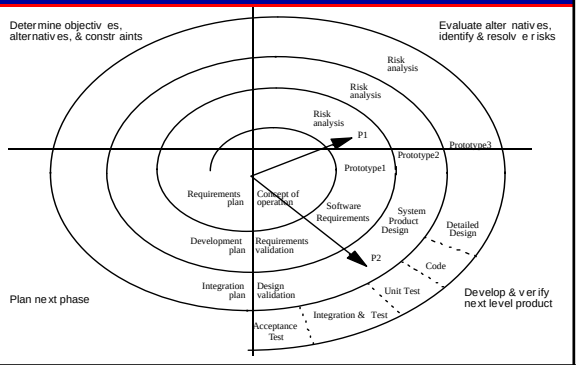
- *risk*-driven *process model* generator
- answers two main questions:
  - What should be done next?
  - For how long should it continue?
- encompasses features of the phased lifecycle as well as the prototype lifecycle
- uses risk analysis as one of its elements
- each cycle is completed with a review by the people concerned with the project
- overcomes major sources of project risk with the Risk Management Plan
- Radial dimension shows cumulative cost
- Angular dimension shows progress made
- helps in being more compatible with other model

Activities ("Rounds")

- Concept of Operations
- Software Requirements
- Software Product Design
- Detailed Design
- Code
- Unit Test
- Integration and Test
- Acceptance Test
- Implementation

- For each cycle go through these steps
  - Define objectives, alternatives, constraints
  - Evaluate alternative, identify and resolve risks
  - Develop, verify prototype
  - Plan next "cycle"

Spiral Model…



Strengths

- has a wide range of options to accommodate the good features of other lifecycle models.
- the risk-avoidance approach keeps from having additional difficulties.
- prepares for lifecycle evolution, growth, and changes of the software product.
- incorporates software quality objectives into software product development. Emphasis is placed on identifying all objectives and constraints during each round.
- The risk analysis and validation steps eliminate errors early on.
- Great amounts of detail are not needed except in the case where this lack of detail jeopardizes the project.

 U1. 76

Weaknesses

- Lack of explicit process guidance in determining objectives, constraints, alternatives
- The risk-driven model is dependent on the developers' ability to identify project risk
- Provides more flexibility than required for many applications

 U1. 77

The Limitations of the Waterfall and Spiral Models

- Neither of these model deals well with frequent change
  - The Waterfall model assume that once you are done with a phase, all issues covered in that phase are closed and cannot be reopened
  - The Spiral model can deal with change between phases, but once inside a phase, no change is allowed
- What do you do if change is happening more frequently? ("The only constant is the change")

 U1. 78

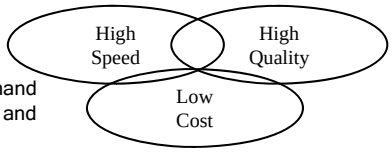
Rapid Application Development (RAD)

- Topical in 1990's after
- Book Rapid Application Development by Martin, J (1991)
- a 'tool kit' methodology
- can utilize a wide range of techniques and tools
- Incremental, plus reliance on many standard modules
- usually very small team.
- emphasis on user involvement and responsibility throughout whole development
- Quality definition in a RAD environment put by James Martin
- ***"meeting the true business (or user) requirements as effectively as possible at the time the system comes into operation"***

RAD Goals

Radically changes way systems are developed with goals of.

- High quality systems
- fast development and delivery
- low costs

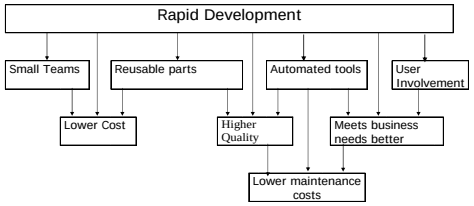


should go hand in hand when the right tools and techniques are used

RAD Properties

- Must be delivered in 2 - 6 months
- split into increments if too large to enable this
- each increment is implemented separately with frequent delivery of working parts of system.

Rapid Development



Rapid development, high quality and lower costs go hand in hand if an appropriate development methodology is used, Martin 1991

RAD - Essentials

Tools
- Code generators, CASE tools, prototyping tools and 4GLs

Methodology
- to use tools as effectively as possible

People
- right skills and talents. Well selected and motivated. End users

Management
- not place obstacles, facilitate fast development

Infrastructure
- In which fast development can take place

Advantages

- Quick initial view is possible
- Less development time
- User involvement increases the acceptability

Disadvantages

- User involvement is difficult through out the life cycle
- Difficult to reduce the development time significantly
- Reusable components may not be available
- Availability of highly skilled personnel is difficult
- Not effective if system is not modularized properly

 U1. 85

Selection of a Life Cycle Model

- Requirement
- Development Team
- Users
- Project type & Associated Risk

 U1. 86

Based on Requirements

Requirements	Waterfall	Prototype	Iterative Enhance	Evolut. Develop.	Spiral	RAD
Easily understandable and defined	Yes	No	No	No	No	Yes
Change requir.	No	Yes	No	No	Yes	No
Define requir. early	Yes	No	Yes	Yes	No	Yes
Indicating a complex system to be built	No	Yes	Yes	Yes	Yes	No

 U1. 87

Based on Development Team

Development Team	Waterfall	Prototype	Iterative Enhance	Evolut. Develop.	Spiral	RAD
Less experience on similar projects	No	Yes	No	No	Yes	No
Less domain knowledge (new to technology)	Yes	No	Yes	Yes	Yes	No
Less Experience on tools	Yes	No	No	No	Yes	No
Availability of training, if required	No	No	Yes	Yes	No	Yes

Based on User Involvement

User Involvement	Waterfall	Prototype	Iterative Enhance	Evolut. Develop.	Spiral	RAD
In all phases	No	Yes	No	No	No	Yes
Limited participation	Yes	No	Yes	Yes	Yes	No
No previous experience of participation in similar projects	No	Yes	Yes	Yes	Yes	No
Expert in problem domain	No	Yes	Yes	Yes	No	Yes

Based on Project Type & Risk

Project type & Risk	Waterfall	Prototype	Iterative Enhance	Evolut. Develop.	Spiral	RAD
Enhancement of existing system	No	No	Yes	Yes	No	Yes
Funding is stable for the project	Yes	Yes	No	No	No	Yes
High reliability requirements	No	No	Yes	Yes	Yes	No
Tight project schedule	No	Yes	Yes	Yes	Yes	Yes
Use of reusable components	No	Yes	No	No	Yes	Yes
Resources scarce	No	Yes	No	No	Yes	No

What we learnt

- ✓ Inherent Problems with Software Development
- ✓ Generic life cycle phases
- ✓ Processes, Activities and Tasks
- ✓ Modeling Dependencies in a Software Lifecycle
- ✓ Generic software process models
  - ✓ Build-and-fix model
  - ✓ Waterfall model
  - ✓ Prototyping model
  - ✓ Incremental model
  - ✓ V Model
  - ✓ Spiral model
  - ✓ RAD
- ✓ Selection of Life Cycle Model

 U1. 91

Practical Problems

1. A simple data Processing System
2. A data entry system for office staff that has never used computer before. The user interface and user friendliness are extremely important.
3. A new system for comparing fingerprints. It is not clear if the current algorithms can compare fingerprints in the given response time constraints
4. A spreadsheet system that has some basic features and many other desirable features that use this basic features.
5. A new missile tracking system. It is not known if the current hardware/software technology is mature enough to achieve the goals.

 U1. 92

Practical Problems

1. An on-line inventory management system for an automobile industry.
2. A flight control system with extremely high reliability. There are many potential hazards with such a system.
3. A website for online store which always has a list of desired features it wants to add and add them quickly.

 U1. 93

Problem

- Suppose that you have to build a product to determine the inverse of 3.748571 to four decimal places. Once the product has been implemented and tested, it will be thrown away. Which life-cycle model would you use? Give reason for your answer.

Problem

- You are a software engineering consultant and have been called in by the vice president for finance of Deplorably Decadent Desserts, a corporation that manufactures and sells a variety of desserts to restaurants. She wants your organization to build a product that will monitor the company's product, starting with the purchasing of the various ingredients and keeping track of the desserts as they are manufactured and distributed to the various restaurants. What criteria would you use in selecting a life cycle model for the project?

Problem

- Your development of the stock control product for Deplorably Decadent Desserts is highly successful. As a result Deplorably Decadent Desserts wants the product to be rewritten as a COTS package to be sold to a variety of different organizations that prepare and sell food to restaurants as well as to retail organizations. The new product therefore must be portable and easily adapted to new hardware and operating systems. How do the criteria you would use in selecting a life cycle model for this project differ from those in your answer to problem 2

Problem

- You are a software-engineering consultant. The executive vice president of a publisher of paperback books wants you to develop a product that will carry out all the accounting functions of the company and provide online information to the head office staff regarding orders and inventory in the various company warehouses. Terminals are required for 15 accounting clerks, 32 order clerks and 42 warehouse clerks. In addition, 18 managers need access to the data. The president is willing to pay $30,000 for the hardware and the software together and wants the complete product in 4 weeks. What do you tell hem? Bear in mind that, as a consultant, you want his business, no matter how unreasonable his request.

Software Requirements Analysis and Specification

Learning Objectives

- **What are requirements?**
- **What is requirements engineering?**
- **What happens if the requirements are wrong?**
- **Why is requirements engineering difficult?**
- **Requirement Engineering Process Steps**
- **Type of requirements**
- **Requirements Document**
- **Requirements Phase: Deliverables**

Learning Objectives..

Requirement Elicitation Methods

- Onsite Observation
- Questionnaire
- Interviews
- Brainstorming Sessions
- Facilitated Application Specification Technique (FAST)
- Quality Function Deployment (QFD)
- Viewpoint-oriented elicitation
- Ethnography
- Scenarios
- Use Case Approach
- Prototyping

 U1. 100

Learning Objectives..

- Requirement Analysis
- Context diagram
- Model the requirements
- Data Flow Diagram
- Data Dictionary
- Guidelines for Writing Requirements
- The Requirements Document
- Nature of the SRS
- Characteristics of a good SRS
- IEEE requirements standard

 U1. 101

What are Requirements?

- Defined, during the early stages of a system development as a **specification** of what should be implemented or as a constraint of some kind on the system.
- Can be defined as
  - a user-level facility description,
  - a detailed specification of expected system behavior,
  - a general system property,
  - a specific constraint on the system,
  - information on how to carry out some computation,
  - a constraint on the development of the system.
- inevitable as requirements may serve a dual function
  - the basis for a bid for a contract - therefore must be open to interpretation
  - the basis for the contract itself - therefore must be defined in detail

 U1. 102

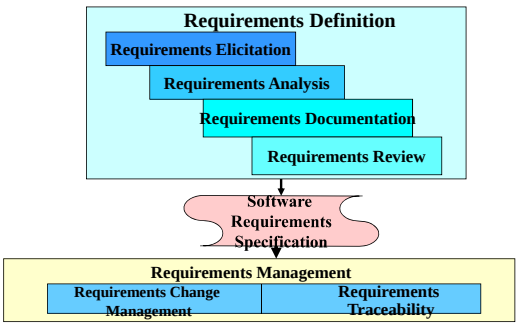
What happens if the Requirements are Wrong?

- The system may be delivered late and cost more than originally expected.
- The customer and end-users are not satisfied with the system. They may not use its facilities or may even decide to scrap it altogether.
- The system may be unreliable in use with regular system errors and crashes disrupting normal operation.
- If the system continues in use, the costs of maintaining and evolving the system are very high.

Why is Requirements Engineering Difficult?

- Businesses operate in a rapidly changing environment so their requirements for system support are constantly changing.
- Multiple stakeholders with different goals and priorities are involved in the requirements engineering process.
- System stakeholders do not have clear ideas about the system support that they need.
- Requirements are often influenced by political and organizational factors that stakeholders will not admit to publicly.
- Over-reliance on CASE tools
- Tight project schedule
- Communication barriers
- Lack of resources

Requirement Engineering Process Steps



Definitions and Specifications

Requirement definition

The software must provide a means of representing and accessing external files created by other tool

Requirement Specifications

1. The user should be provided with facilities to define the type of external files.
2. Each external file type may have an associated tool which may be applied to the file.
3. Each external file type may be represented as a specific icon on the user display.
4. Facilities should be provided for the icon representing an external file to be defined by the user
5. When a user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon

Type of Requirements-I

Functional requirements

- Statements of services the system should provide,
- how the system should react to particular inputs
- how the system should behave in particular situations.

Non-functional requirements

- constraints on the services or functions offered by the system such as
- timing constraints, constraints on the development process, standards, etc.

Domain requirements

- Requirements that come from the application domain of the system
- reflect characteristics of that domain

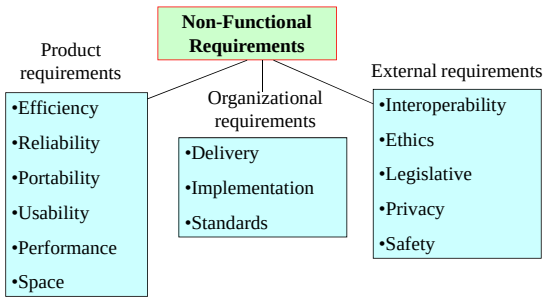
Examples of Functional Requirements

- The user shall be able to search either all of the initial set of databases or select a subset from it.
- The system shall provide appropriate viewers for the user to read documents in the document store.
- Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account's permanent storage area.

Non-functional Requirements

- Define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless

Non-Functional Requirements Classifications



Non-functional Requirements Examples

Product requirement

- 4.C.8 It shall be possible for all necessary communication between the APSE and the user to be expressed in the standard Ada character set

Organisational requirement

- 9.3.2 The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95

External requirement

- 7.6.5 The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system

Non-functional Requirements Measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K Bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Type of Requirements-II

Known requirements
- Something a stakeholder believes to be implemented

Unknown requirements
- Forgotten by the stakeholder because they are not needed right now or needed only by another stakeholder

Undreamt requirements
- Stakeholder may not be able to think of new requirements due to limited domain knowledge

Known, Unknown and Undreamt requirement may be functional or nonfunctional

Type of Requirements-III

- User requirements
- System requirements

User Requirements

- Should describe functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge
- User requirements are defined using natural language, tables, and diagrams
- Problems with natural language
  - Precision vs. understand ability
  - Functional vs. non-functional requirements confusion
  - Requirements amalgamation

 U1. 115

System Requirements

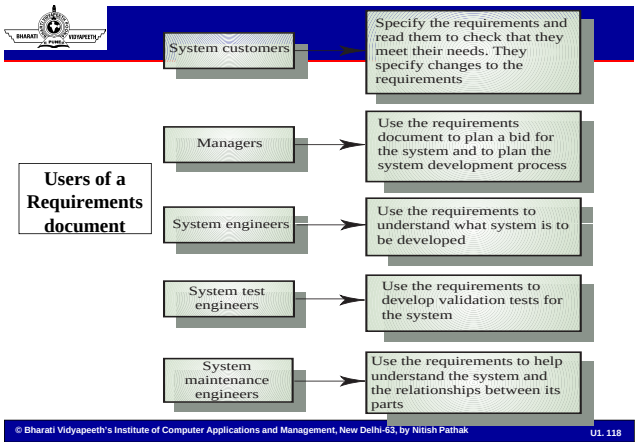
- More detailed specifications of user requirements
- Serve as a basis for designing the system
- May be used as part of the system contract

 U1. 116

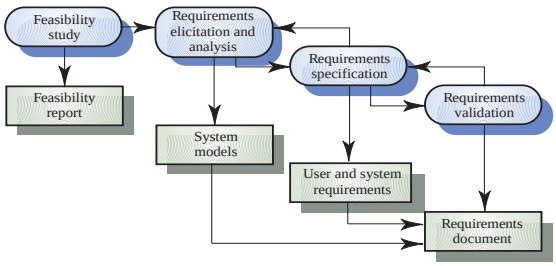
Requirement Document

- Specify external system behaviour
- Easy to change
- Serve as reference tool for maintenance
- Record forethought about the life cycle of the system
  - i.e. predict changes
- Characterise responses to unexpected events

 U1. 117



The Requirements Engineering Process



Requirements Elicitation and Analysis

Requirements Elicitation:

- Definition of the system in terms understood by the client ("Problem Description")
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc.
- These are called *stakeholders.*

Requirements Analysis:

- Technical specification of the system in terms understood by the developer ("Problem Specification")

Requirement Elicitation Methods

- Onsite Observation
- Questionnaire
- Interviews
- Brainstorming Sessions
- Facilitated Application Specification Technique (FAST)
- Quality Function Deployment (QFD)
- Viewpoint-oriented elicitation
- Ethnography
- Scenarios
- Use Case Approach
- Prototyping

 U1. 121

Interviews

- Face-to-face interpersonal meeting designed to identify relations or verify information and capture information as it exists

Advantages

- Flexible tool
- Offering better opportunity than questionnaire
- Effective for complex subjects
- People enjoy being interviewed

Drawbacks

- Needs preparation time and money to conduct

 U1. 122

Interviews cont..

The art of interviewing

- Creating permissive situation in which the answers offered are reliable

Arranging the interview

- Physical location, time of the interview and order of interviewing assures privacy and minimal interruption

Guides to a successful interview

- Set the stage for the interview
- Establish rapport; put the interviewee at ease
- Phrase questions clearly and succinctly
- Be a good listener; avoid arguments
- Evaluate the outcome of the interview

Interviews may be open-ended or structured

 U1. 123

Selection of Stakeholder

- Must be selected based on their technical expertise, domain knowledge, credibility and accessibility
- Several groups to be considered for interview
  - Entry Level personnel
  - Mid level stakeholders
  - Managers or other Stakeholders
  - Users of the Software

Brainstorming Sessions

- A group technique to understand the requirements
- Requirements in the long list can be categorized, prioritized and pruned
- The facilitator required to handle group bias and group conflicts

Basic Guidelines

- Arrange a meeting at a neutral site for developers and customers
- Establishment of rules for preparation and participation
- Prepare an informal agenda that encourages free flow of ideas
- Appoint a facilitator
- Prepare a definition mechanism
- Participants should not criticize or debate

FAST Session Preparations

Each FAST attendee is asked to make a list of objects that are:

- Part of the environment that surrounds the system
- Produced by the system
- Used by the system
- List of services that interact or manipulate the objects
- List of constraint
- Performance criteria

 U1. 127

Activities of FAST session

- Participants presents the list of objects, services, constraints and performance for discussion
- Prepare the combine list for each topic
- Discuss the consensus combined list and finalized by facilitator
- Team are divided in subteams, each works for mini specifications
- Each subteam presents the mini specifications to all FAST attendee
- Prepare the issue list
- Each attendee prepares a list of validation criteria to finalize the consensus validation criteria list
- Subteam write the complete draft specifications using all inputs from the FAST meeting

 U1. 128

QFD steps

- Identify all the stakeholders and any initial constraints identified by customer that affect requirement development
- List out customer requirements
- Assign a value to each requirement indicating the degree of importance
- Final list of requirements may be reviewed by requirement engineers and categorize like
  - It is possible to achieve
  - It should be deferred and the reason thereof
  - It is impossible and should be dropped from consideration

 U1. 129

Use cases

•a scenario based technique in the UML which identify the actors in an interaction and which describe the interaction itself

•A set of use cases should describe all possible interactions with the system

•Sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system

Purpose

•To define the scope of the system — what will be handled by the system and what will be handled outside the system.

•To define who and what will interact with the system.

•To outline the functionality of the system.

Use Case Modeling: Overview

The *Use Case Model* consists of the following:

- Actors
- Use cases
- Relationships
- System boundary

Steps of use case modeling:

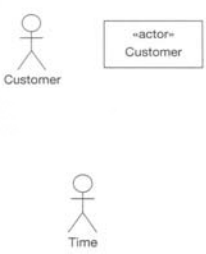
- Find the system boundary
- Find the actors
- Find the use cases:
- Describe how Actors and Usecase interacts
- Package Usecases nad Actors
- Draw Usecase diagrams
- Evaluate your Results

Use Case Model- Characteristics

- Need to be verified by users/managers
- Will be used as basis for rest of the development
- Therefore, It must be
  - Simple
  - Correct
  - Complete
  - Consistent
  - Verifiable, Modifiable, Traceable
  - Rankable (for iteration)

Actors

- a role taken by an external entity when interacting with the system directly
- a stereotype of class with its own icon
- An actor:
  - Is always external to the system
  - Interacts directly with the system
  - Represents a role played by people or things, not specific people or specific things



Use Case

- According to Rumbaugh, a *use case* is "a specification of sequences of actions, including variant sequences and error sequences, that a system, subsystem, or class can perform by interacting with outside actors"
- Use cases:
  - Are always started by an actor
  - Are always written from an actor's point of view

Describe How Actors and Use Cases Interact

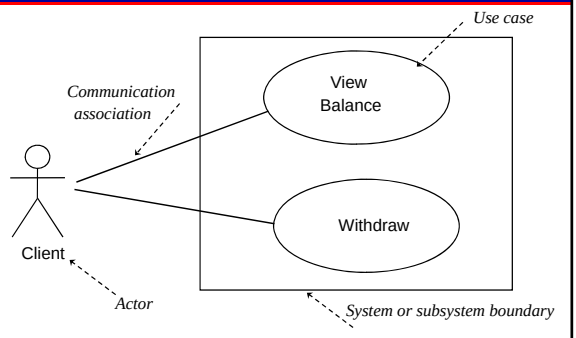
- to show how actors relate to the use case,
- must define a communicates-association
- navigable in the same direction as the signal transmission between the actor and the use case

ATM example- Candidate Requirements

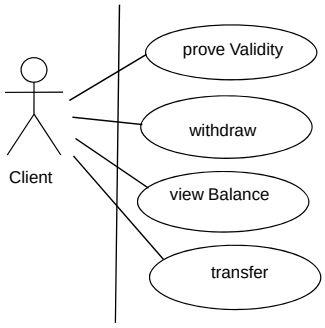
Apply for a card (??)
See the balance
Withdraw money
Change the PIN
Enter a new CARD detail
Add another account to a CARD
Block a card (Manager)
Issue Money (Money Dispenser)
Print summary (at 2 PM)

Actors
Client
Bank Clerk
Bank Manager
Money Dispenser
2 PM

Use Case Diagrams- Notation



Transactions



Usecase Withdraw:
For this, client must have logged-on already. The Client may withdraw money from an account upto the specified limit

Ranking:
(1) prove Validity
(2) View Balance
(3) Withdraw

Etc.

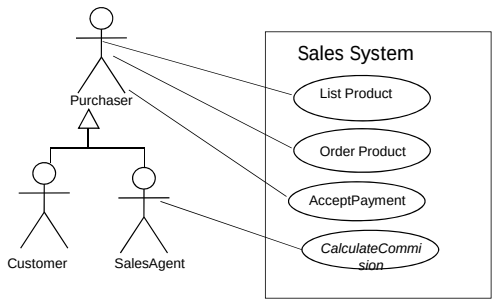
Advanced Use Case Modelling

Start with the **priority** ones

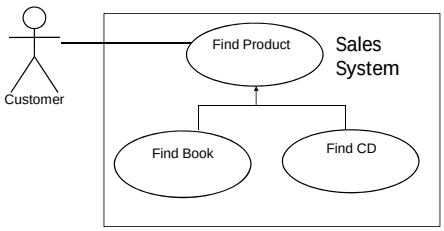
Add structure to the use case model:

- identify generalization in Actors
- identify generalization in use cases
- include and extend relationships

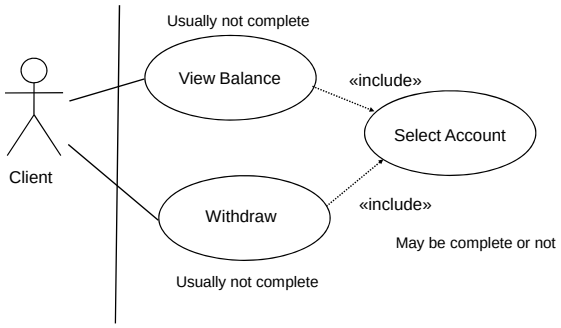
Generalization Actor



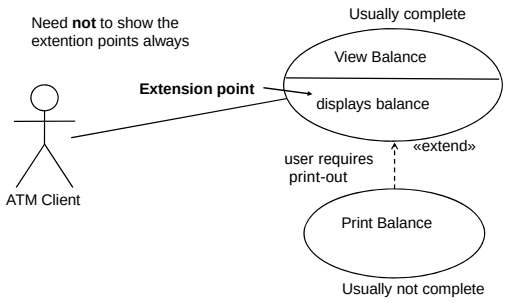
Generalization Usecase



Include



Extend



Template Use Case Descriptions

Many projects use templates

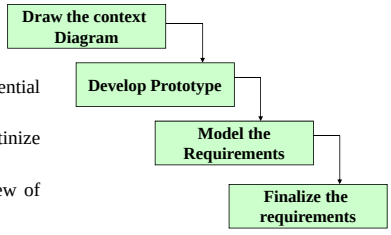
- name of use case
- pre-conditions
- post-conditions
- purpose
- description
- alternative courses
- errors

Templates - Example

Name : Withdraw
Actor : Client
Pre-conditions : User already logged-in
Post-conditions : Amount is deducted from user's account
Purpose : To allow the client to withdraw money
Description:
(1) Client initiates this usecase by selecting 'withdraw'
(2) System displays all the accounts and prompts to select any one
(3) Client selects one account
(4) System prompts for the amount (fast cash or --)
- - -- - -

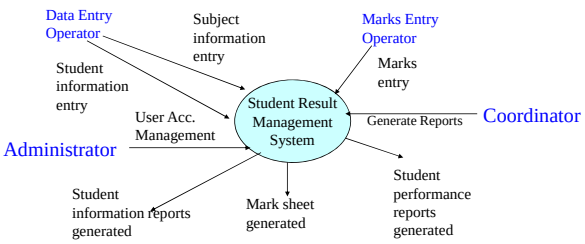
Requirement Analysis

- Very important and essential activity after elicitation
- Analyze refine and scrutinize the gathered information
- Provides a graphical view of the entire system



Context diagram

- Simple model that defines the boundaries and interfaces of the proposed system with external world



Model the requirements

- Consists of various graphical representations of functions, data entities, external entities and their relationships
- Help to find incorrect, incorrect, inconsistent, missing requirements
- Models include DFD, ERD, DD, State Transition Diagrams etc.

Data Flow Diagram

- modeling tool that allows us to picture a system as a network of functional processes connected to one another by "pipelines" and "holding tanks" of data
- synonyms for dataflow diagram:
  - Bubble chart
  - DFD (the abbreviation we will use throughout this book)
  - Bubble diagram
  - Process model (or business process model
  - Business flow model
  - Work flow diagram
  - Function model
  - "a picture of what's going on around here"

Components Of DFD

- Process
- Flow
- Store
- Terminator

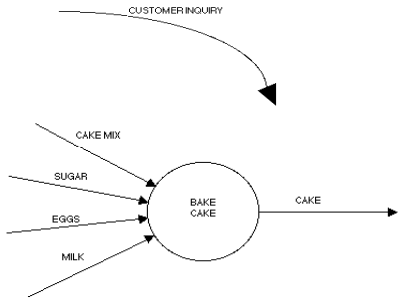
Process

- First component of the DFD, shows a part of the system that transforms inputs into outputs
- Common synonyms are a bubble, a function, or a transformation
- named or described with a single word, phrase, or simple sentence that describes *what* the process does

The Flow

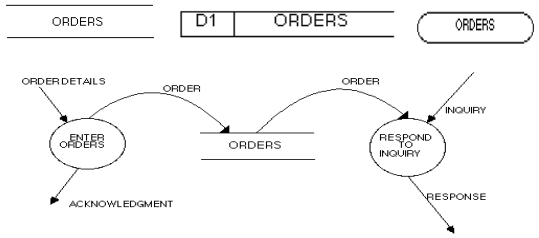
- used to describe the movement of chunks, or packets of information from one part of the system to another part
- name represents the meaning of the packet that moves along the flow
- flows show *direction*
- data moving along that flow will either travel to another process (as an input) or to a store or to a terminator

The Flow cont..



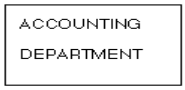
The Store

- used to model a collection of data packets at rest.
- Stores are connected by flows to processes



The Terminator

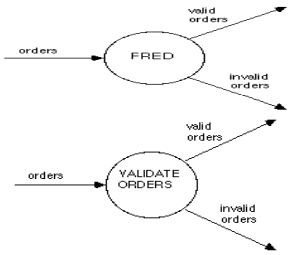
- Represent external entities with which the system communicates
- a person or a group of people or department, or another system but outside the control of the system being modeled
- Represent the interface between our system and the outside world
- The systems analyst cannot change the contents, or organization, or internal procedures associated with the terminators
- Any relationship that exists between terminators will not be shown in the DFD model.



Guidelines for Constructing DFDs

- Choose meaningful names for processes, flows, stores, and terminators
- Number the processes from left to right, top to bottom
- Redraw the DFD as many times as necessary
- Avoid overly complex DFDs
- Make sure the DFD is internally consistent and consistent with any associated DFDs
  - Technically correct
  - Acceptable to the user
  - Neatly enough drawn that you wouldn't be embarrassed to show it to the board of directors in your organization

Guidelines for Constructing DFDs cont..

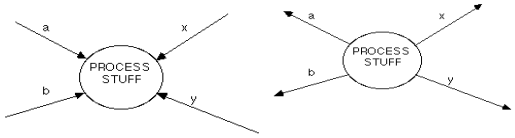


Inappropriate Name

Appropriate Name

Logical Consistency of DFD

- Avoid infinite sinks, bubbles that have inputs but no outputs
- Avoid spontaneous generation bubbles
- Beware of unlabeled flows and unlabeled processes

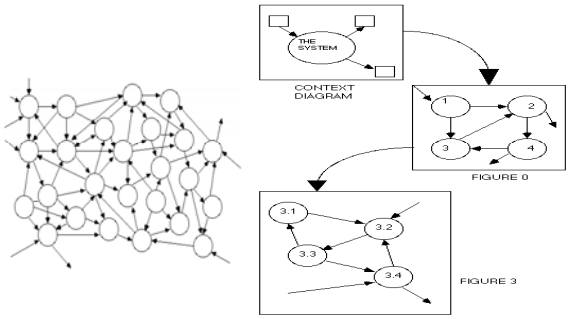


Leveled DFDs

- The numbers serve as a convenient way of relating a bubble to the next lower level DFD which more fully describes that bubble
- simple system
  - find two to three evels
- medium-size system
  - typically have three to six levels;
- a large system l
  - have five to eight levels.

If, for example, each figure has seven bubbles, then there will be 343 bubbles at the third level, 16,807 bubbles at the fifth level, and 40,353,607 bubbles at the ninth level

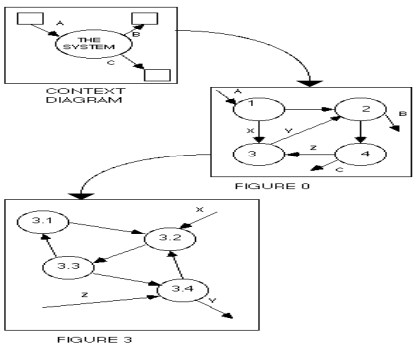
Leveled DFDs cont..



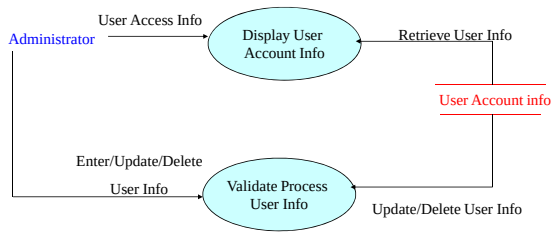
Leveled DFDs cont..

- Must all parts of the system be partitioned to the same level of detail? **No**
- How do you ensure that the levels of DFDs are consistent with each other?
- the dataflows coming into and going out of a bubble at one level must correspond to the dataflows coming into and going out of an entire figure at the next lower level which describes that bubble

Leveled DFDs cont..



Level-2 DFD for User Account Maintenance



Data Dictionary

- A catalog-repository of the elements in a system
- List of all the elements composing the data flowing through a system i.e Data Flows, Data Stores, Processes

- **Why DD**
- Manage details in large system
- Documentation
- Improved analyst/user communication
- Maintain control information
- Useful to locate errors and omissions
- Important step in building a data base

Data Dictionary Contents

- Data Elements
- Data Structures
- Data Flow
- Data Store
- Process

Data Elements in DD

- **Format for Data Element Description in DD**

Data Element Name_____
Description_____
Type _____
Length_____
Aliases_____
Range of Values_____To_____
Typical Value_____
List of Specific Values (If Any)
Valid Prefixes Meaning
Other Editing Details_____

Data Structures in DD

- Grouping of data elements
- Set of data items that are related to one another and that collectively describe a component in the system
- Built on four relationships that may be either data items or other data structure
  - Sequence Relationship
  - Selection relationship (Either/Or) represented as [] e.g Std. No./SSN, Parent/Guardian
  - Iterative relationship {} e.g one to six subjects
  - Optional relationship () e.g. middle name

Data Structures Examples

STUDENTDATA=NAME+STREET_ADDRESS+CITY+STATE+POSTAL_CODE+TNO+[STUDENTNO/SSN]+{COURSE NO+C.NAME}+(SECTION NO+TIME+DAY)

Name = first-name+(middle-name)+last-name
courtesy-title = [Mr. | Miss | Mrs. | Ms. | Dr. | Prof.]

first-name = **{**legal-character**}**

last-name = **{**legal-character**}**

legal-character = [A-Z | a-z | ' | - | |]

Data Flow in DD

DATA FLOW_____

DESCRIPTION_____

FROM PROCESS_____

TO PROCESS_____

DATA STRUCTURE_____

Data Store in DD

DATA STORE_____
DESCRIPTION_____
INBOUND DATAFLOW_____
OUTBOUND DATAFLOW_____
DATA DESCRIPTION_____
VOLUME_____
ACCESS_____

Process in DD

PROCESS_____
DESCRIPTION_____
INPUT_____
OUTPUT_____
LOGIC SUMMARY_____

Constructing Data Dictionary

- Each unique Data Flow, Data Store, Process in the DFD must have entry in DD
- Definitions must be readily accessible by name
- Avoid redundancy
- Simple to update
- Use straightforward but specific procedures to write DD
- DD is an integral component of the structured specifications
- Correlation between DFD and DD is important

 U1. 172

Guidelines for Writing Requirements

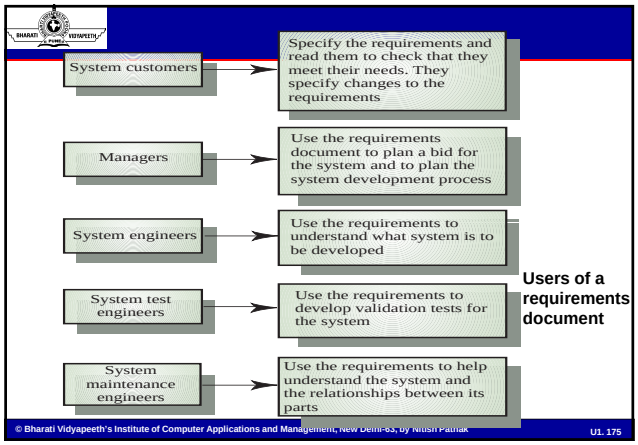
- Invent a standard format and use it for all requirements
- Use language in a consistent way.
- Use text highlighting to identify key parts of the requirement
- Avoid the use of computer jargon

 U1. 173

The Requirements Document

- The official statement of what is required of the system developers
- Should include both a definition and a specification of requirements
- NOT a design document.
- set of WHAT the system should do rather than HOW it should do it

 U1. 174



Nature of the SRS

- Functionality
- External Interfaces
- Performance
- Attributes
- Design Constraints imposed on an implementation

U1. 176

Characteristics of a Good SRS

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

U1. 177

IEEE Requirements standard

Introduction
- 1.1 Purpose
- 1.2 Scope
- 1.3 Definition, acronyms and abbreviations
- 1.4 References
- 1.5 overview

 U1. 178

IEEE Requirements Standard

Overall description
- 2.1 Product perspective
  - 2.1.1 System Interfaces
  - 2.1.2 Interfaces
  - 2.1.3 H/W Interfaces
  - 2.1.4 S/W Interfaces
  - 2.1.5 Communication Interfaces
  - 2.1.6 Memory Constraints
  - 2.1.7 Operations
  - 2.1.8 Site Adaption Requirements

 U1. 179

IEEE Requirements Standard cont..

- 2.2 Product functions
- 2.3 User characteristics
- 2.4 Constraints
- 2.5 Assumptions and dependencies
- 2.6 Apportioning of Requirements

 U1. 180

IEEE Requirements Standard cont..

Specific requirements

- 3.1 External interfaces (details of inputs & outputs)
- 3.2 Functions
- 3.3 Performance requirements
- 3.4 Logical database requirements
- 3.5 Design constraints
  - 3.5.1 Standard Compliance

IEEE Requirements Standard cont..

- 3.6 Software system attributes
  - 3.6.1 Reliability
  - 3.6.2 Availability
  - 3.6.2 Security
  - 3.6.3 Maintainability
  - 3.6.4 Portability
- 3.7 Additional Comments

IEEE Requirements Standard cont..

3.2 Organizaing the Specific Requirements

a. System Mode
b. User Classs
c. Objects
d. Feature
e. Stimulus
f. Response
g. Functional Hierarchy

Requirement Management

- Process of understanding and controlling changes to system requirements
- Two Categories of Requirement
  - Enduring Requirements
  - Volatile Requirements

Requirement Change Management

- Allocating adequate resources
- Analysis of Requirement Changes
- Documenting Requirements
- Creating requirement traces throughout the project life cycle from inception to the final work products
- Establishing team Communication
- Establishing a baseline for requirement specification
  - Approval of changes from competent authority
  - Communicate the approval to all stakeholders
  - History of changes is maintained
  - All documents are updated to reflect the change

ER Diagram

- specialized graphic that illustrates the interrelationships between entities in a database.
- Three Types of information
  - Entities
  - Attributes
  - Relationship

Example

residents of a city

Example

- A movie has a title, a year and a length. Since some movies have the same title, it takes a title and a year to uniquely identify a movie.
- Some movies are remakes of others.
- A star has a name and an address. A star's name uniquely identifies the star.
- A star can appear in any number of movies.
- Some movies have many stars and some have none.
- A studio has a name and an address, and is uniquely identified by its name.
- A star can belong to at most one studio.
- A studio can own any number of movies.
- A movie is always owned by at most one studio, but some are not owned by any studio.

cont..

- A studio may or may not have a president, but nobody can be a studio president without being the president of some studio.
- Studio presidents are uniquely identified by their name, but they also have an address.
- No one can be the president of more than one
- studio.
- Stars and studio presidents are both examples of movie people, but there are other types of movie people as well.
- No star can be a studio president.

Solution

From the domain description we derive the entity types and their attributes (including primary keys):

• Entity types and their attributes:
– Movie
• title, year, length
– Star
• name, address
– Studio
• name, address
– Studio President
• name, address
– Movie People
• name, address

cont..

- From the domain description we derive the relationships between the entity types and their cardinalities:
- Relationships:
  - some movies are **remakes** of (exactly 1) movie; each movie has (0 or more) remakes
  - studios **own** (0 or more) movies; each movie is **owned by** (at most 1) studio
  - stars **appear in** (0 or more) movies; movies **have** (0 or more) stars appearing in them

cont..

- stars **belong to** (at most 1) studio; studios **have** (0 or more) stars belonging to them
- studio presidents **are presidents of** (exactly 1) studio; studios **have** (at most 1) president
- stars and studio presidents are (disjoint but incomplete) subtypes of movie people

1.1 Purpose (SAMPLE)

The purpose of this document is to describe the external requirements for a course scheduling system for an academic department in a University. It also describes the interfaces for the system.

1.2 Scope (SAMPLE)

This document is the only one that describes the requirements of the system. It is meant for use by the developers and will be the basis for validating the final delivered system. Any changes made to the requirements in the future will have to go through a formal change approval process. The developer is responsible for asking for clarifications, where necessary, and will not make any alterations without the permission of the client.

2 Overall Description (SAMPLE)

In the computer science department there are a set of classrooms. Every semester the department offers courses, which are chosen from the set of department courses. A course has expected enrollment and could be for graduate students or undergraduate students. For each course, the instructor gives some time preferences for lectures. The system is to produce a schedule for the department that specifies the time and room assignments for the different courses. Preference should be given to graduate courses, and no two graduate courses should be scheduled at the same time. If some courses cannot be scheduled, the system should produce a "con ict report" that lists the courses that cannot be scheduled and the reasons for the inability to schedule them.

What we Learnt

- ✓ Understand meaning of requirement
- ✓ Importance of Requirement Engineering
- ✓ Requirement Engineering Process Steps
- ✓ Type of requirements
- ✓ Requirements Phase: Deliverables
- ✓ Requirement Elicitation Methods
- ✓ Requirement Analysis
- ✓ Data Flow Diagram
- ✓ Writing Requirement Document

What we Learnt..

- ✓ **Requirement Elicitation Methods**
  - ✓ Onsite Observation
  - ✓ Questionnaire
  - ✓ Interviews
  - ✓ Brainstorming Sessions
  - ✓ Facilitated Application Specification Technique (FAST)
  - ✓ Quality Function Deployment (QFD)
  - ✓ Viewpoint-oriented elicitation
  - ✓ Ethnography
  - ✓ Scenarios
  - ✓ Use Case Approach
  - ✓ Prototyping

Review Question Objective

Q1. Which is the oldest Profession? Why?
a) Physician b) Civil Engineer c) Computer Scientist
Q2. What is Software?
Q3. What are different types of Documents?
Q4. Define Documentation Manual?
Q5. Define Procedure Manuals?
Q6. What are Documentation Manuals?
Q7. What are Operating Procedure Manuals?
Q8. Why Software is Intangible?
Q9. Why Software is easy to reproduce?
Q10. Why Software industry is Labor Intensive?
Q11. Why Software is easy to modify?

Review Question Objective..

Q12. Software doesn't wear out. Give reason.
Q13. What are different Software Characteristics?
Q14. Define Failure Intensity.
Q15. Compare Failure intensity of Hardware and Software.
Q16. What are different types of software?
Q17. Define Custom, Generic and Embedded Software.
Q18. For which type of software maximum number of copies are used
a) Custom b) Generic c) Embedded
Q19. For which type of software lowest processing time is devoted
a) Custom b) Generic c) Embedded
Q20. For which type of software annual development efforts are highest
a) Custom b) Generic c) Embedded

Review Question Objective

Q21. How you will decide the Software is Good or Bad?
Q22. What are the different types of maintenance?
Q23. Differentiate Corrective and perfective maintenance?
Q24. What type of maintenance is least in practice?
Q25. Define Following Software Attributes
i) Maintainability ii) Dependability iii) Efficiency iv) Usability
Q26. Define Software Crises?
Q27. Give three suggestions to avoid the situation of Software Crises.
Q28. What Are Quality Issues in Software?
Q29 Why Software Cost are increasing as Hardware Costs Continue to decline?
Q30. What are the primary drivers of Software Cost?

Review Question Objective..

Q31. Why Cost to fix an error increases as it is found later and later in the software lifecycle?
Q32. Why Software Project late? Give any two reasons.
Q33. When we recognize slippage in the IT Project, should we add more people? Give Reason.
Q34. Define Quality, Quality Control, Quality Assurance and Quality Management System.
Q29. Q35. Differentiate Quality Control and Quality Assurance.
Q36. Define CASE
Q37. Differentiate Lower CASE and Upper CASE
Q38. Write at least two software myths.
Q39. Define Process
Q40. What are generic activities in all Software Process?

Review Question Objective..

Q41. Why Software process improvement is difficult?
Q42. Draw and label Process Improvement Learning Curve.
Q43. Define the terms with example
I. Deliverables & Milestones
II. Product
III. Process
IV. Measure, Metric, Measurement
V. Software Process & Product Metric
VI. Productivity
VII. Effort
VIII. Software Component
IX. Verification &Validation

Review Question Objective..

Q44. Define Software Lifecycle Modeling, Software Lifecycle, Software Development Methodology, Software Life Cycle Phase

Q45. What must be focus in Pre Development Phase of any Software?

Q46. What must be focus in Development Phase of any Software?

Q47. What are the three specific steps in Development Phase of any Software?

Q48. What must be focus in Post Development Phase of any Software?

Q49. Define Process, Activity and Task with example.

Q50. Project risk factor is consider in (i) Waterfall model (ii) Spiral model (iii) Quick & Fix model (iv) (ii) and (iii).

 U1. 202

Review Question Short Type

Q1. Describe the characteristics of software contrasting it with the characteristics of hardware?

Q2. Why researchers identify the necessity to engineer software?

Q3. Why we are in perpetual "software crisis"?

Q4. Discuss the history of software crises with examples. Why are the case tools not normally able to control it?

Q5. What is software engineering? Is it an art, craft or a science? Discuss.

Q6. What is more important : product or process? Justify your answer.

Q7. What are the key challenges facing software engineering?

Q8. Why is the primary goal of software development now shifted from producing good quality software to good quality maintainable software?

Q9. Discuss the inherent problems with software development.

 U1. 203

Review Question Short Type

Q10. List down the documents after each activity of Software Development Activity.

Q11. What do you understand by the expression "life cycle model of software development"? Why is it important to adhere to a life cycle model during the development of a larger software product?

Q12. What problems will a software development organization face if it does not adequately document its software process?

Q13. Q14. Compare iterative enhancement model and evolutionary development model.

Q15. What do you understand by the expression "life cycle model of software development"? Why is it important to adhere to a life cycle model during the development of a larger software product?

Q16. What problems will a software development organization face if it does not adequately document its software process?

 U1. 204

Review Question Long Type

Q1. Discuss different types of Software Documents.

Q2. Discuss the nature of software

Q3. What are various type of Software?

Q4. How are the software myths affecting software process? Explain with the help of example.

Q5. What do you mean by a software process? What is the difference between a methodology and a process? What problems will a software development house face if it does not follow any systematic process in its software development efforts?

Q6. Discuss the selection process parameters for a life cycle model.

Research Problems

Q1. Draw a DFD for MCA admission system

Q2. Write SRS for BVICAM Time Table Management System

Q3. Prepare Questionnaire to gather requirements for your project.

Q4. Suppose you are the facilitator for a brain storming session to discuss the student related issues in your institute describe your role and prepare all required document for the preparation of brain storming session and also compile the reports of your learnings

Research Problems

Q1. Q5. Discuss the characteristics, pros and cons of the following Software Development Life Cycle Model. Identify the parameter for the comparison of the listed model and compare them.

- Joint Applications Design
- Rapid Application Development (RAD)
- Extreme Programming (XP); extension of earlier work in Prototyping and RAD.
- Open Source Development
- End-user development
- Synchronize and Stablize Model

References

1. K. K. Aggarwal & Yogesh Singh, “Software Engineering”, 2nd Ed., New Age International, 2005.
2. R. S. Pressman, “Software Engineering – A practitioner’s approach”, 5th Ed., McGraw Hill Int. Ed., 2001.
3. Pankaj Jalote, “An Integrated Approach to Software Engineering”, Narosa, 3rd Ed., 2005.
4. Stephen R. Schach, “Classical & Object Oriented Software Engineering”, IRWIN, 1996.
5. James Peter, W. Pedrycz, “Software Engineering: An Engineering Approach”, John Wiley & Sons.
6. Sommerville, “Software Engineering”, Addison Wesley,8th Ed., 2009.

 U1. 208

References

7. Frank Tsui and Orlando Karan, “Essentials of Software Engineering”, Joes and Bartlett, 2nd Ed., 2010.
8. Rajib Mall, “Fundamrntal of Software Engineering”, PHI, 3rd Ed., 2009.
9. Carlo Ghizzi , Mehdi Jazayeri and Dino Mandrioli, “ Fundamental of Software Engineering”, PHI, 2nd Ed.,2003.
10. Carol L. Hoover, Mel Rosso-Llopart and Gil Taran, “Evaluating Project Decision Case Studies in Software Engineering”, Pearson, 2010.
11. Sommerville, “Software Engineering”, Addison Wesley, 2002 http://www.cse.iitk.ac.in/JaloteSEbook/
12. http://www.csse.monash.edu.au/~jonmc/CSE2305/Topics/07.14.SWEng2/html/text.html

 U1. 209