REQUIREMENTS ENGINEERING

Course Objectives

- To establish the importance / relevance of Requirement Specification
- To enlist the problems involved in specifying requirements
- To use modelling techniques to minimise problems in specifying requirements

What are Requirements ?

- A condition or capability needed by a user to solve a problem or achieve an objective
- A condition or capability that must be met or possessed by a system to satisfy a contract, standard, specification, or other formally imposed document
- A software requirements specification (SRS) is a document containing a complete description of what the software will do without describing how it will do it.

Requirements…

- **May range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification**
- **This is inevitable as requirements may serve a dual function**
  - May be the basis for a bid for a contract - therefore must be open to interpretation
  - May be the basis for the contract itself - therefore must be defined in detail
  - Both these statements may be called requirements

Requirements…

- **Software requirements specify what the software must do to meet the business needs.**
- **For example, a stores manager might state his requirements in terms of efficiency in stores management.**
- **A bank manager might state his requirements in terms of time to service his customers.**
- **It is the analyst's job to understand these requirements and provide an appropriate solution.**
- **To be able to do this, the analyst must understand the client's business domain: who are all the stake holders, how they affect the system, what are the constraints, what are the alterables, etc**

Requirements…

- The analyst should not blindly assume that only a software solution will solve a client's problem.
- He should have a broader vision.
- Sometimes, re-engineering of the business processes may be required to improve efficiency
- A detailed statement of what the software must do to meet the client's needs should be prepared. This document is called Software Requirements Specification (SRS) document.

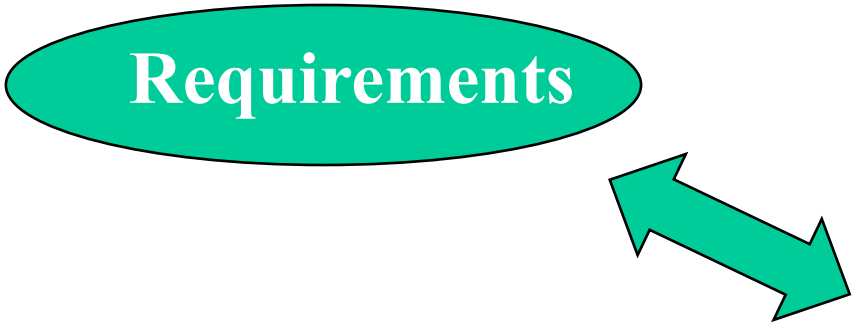
Types of requirement

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers
- System requirements
  - A structured document setting out detailed descriptions of the system services. Written as a contract between client and contractor
- Software specification
  - A detailed software description which can serve as a basis for a design or implementation. Written for developers

Requirements engineering

Requirements engineering is the process of establishing

- the services that the customer requires from a system
- the constraints under which it operates and is developed



The descriptions of the system services and constraints

that are generated during the requirements engineering process

Why the Term Requirements Engineering ?

Engineering, not just Analysis

- Standards/Guidelines, Tools
- Breaking down of requirements, building up of solutions
- (IEEE)
  - Analysis
  - Specification
  - Management

Library Problem

- Check out a copy of a book, return a copy of a book
- Add a copy of a book to the Library
- Remove a copy of a book from the library
- Get the list of books by a particular author or in a particular subject area

Library Problem …Contd…

- Find out the list of books currently checked out by particular borrower
- Find out which borrower last checked out a particular copy of a book
- There are two types of users : Staff Users and Ordinary Borrowers. Transactions 1 and 2 are restricted to staff users

Constraints

1. Copies in the library must be available for checkout or be checked out

2. Copy of the book may be both available and checked out at the same time

3. A borrower may not have more than a predefined number of books checked out at one time

Ambiguities

- What is a Library ?
- Who is a user ?
- Confusion regarding “ *Book* ” and “ *Copy* ”
- What does “ *Available* ” mean ?
- What do “ *Last Checked Out* ” and “ *Currently* ” mean ?

Incompleteness

- Initialization
- Addition of a new book
- Remove all copies of a book
- Create a library
- Add a staff user / ordinary user
- Error Handling
- Missing Constraints
  - Period of Lending
  - Not more than one copy of the same book can be borrowed

Further Examples for Bad Specification

- The counter value is picked up from the last record.
- Inversion of a square, matrix ‘M’ of size ‘n’ such as LM ML = I_n, where ‘L’ is the inverse matrix and I_n is the identity matrix of size ‘n’.
- The software should be highly user friendly
- The output of the program shall usually be given within 10 seconds.
- The software should be developed on DOS system, but should be portable to other systems.

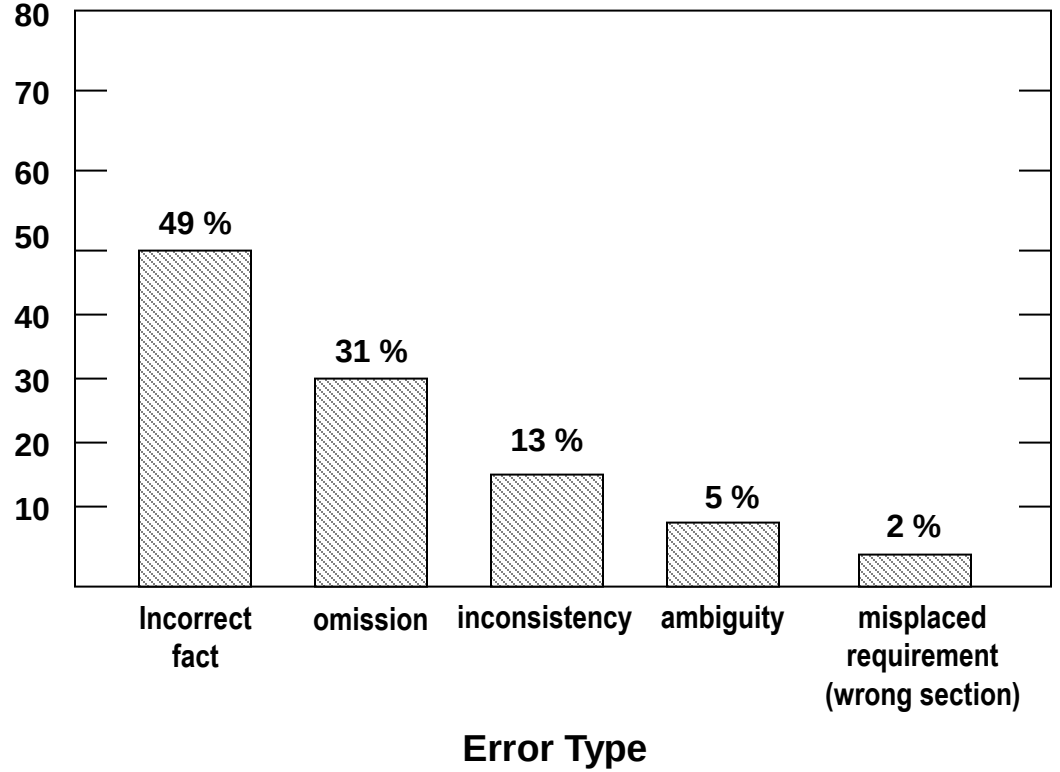
Further Examples for Bad Specification

- In first statement, the word 'last' is ambiguous. It could mean the last accessed record, which could be anywhere in a random access file, or, it could be physically the last record in the file
- Second statement though appears to be complete, is missing on the type of the matrix elements. Are they integers, real numbers, or complex numbers. Depending on the answer to this question, the algorithm will be different.
- How does one determine, whether this requirement is satisfied or not.
- What are the exceptions to the 'usual 10 seconds' requirement?

Characteristics of a Good SRS

- Correct (No errors)
- Unambiguous
- Complete
- No use of TBDs *(To Be Determined)*
- Verifiable (words like “highly”, “usually” not to be used)
- Consistent (non-conflict)
- Modifiable
- Traceable

Distribution of Errors in Requirements Specification



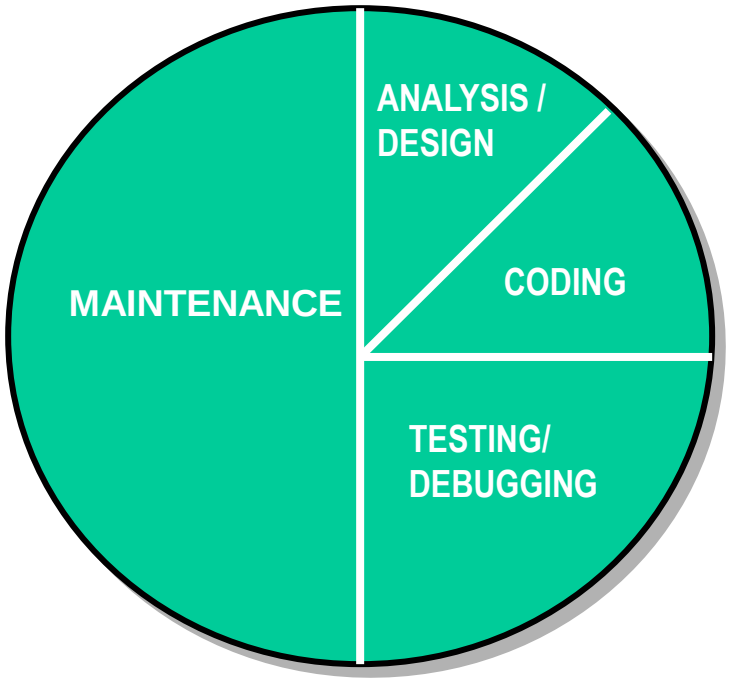
[NAVY A - 7E AIRCRAFTS OPERATIONAL FLIGHT PROGRAM; 5th INT. CONF. ON SOFTWARE ENGINEERING, 1981]

System Development Life Cycle

- Analysis (Requirements Definition, User Requirements Specification and Software Requirements Specification)
- System Design and Specification
- Construction and Testing
- System Acceptance
- Production and Maintenance

Distribution of Effort

- Half the effort in a System Life Cycle is devoted to maintenance
- Half the effort in system development is devoted to testing and debugging



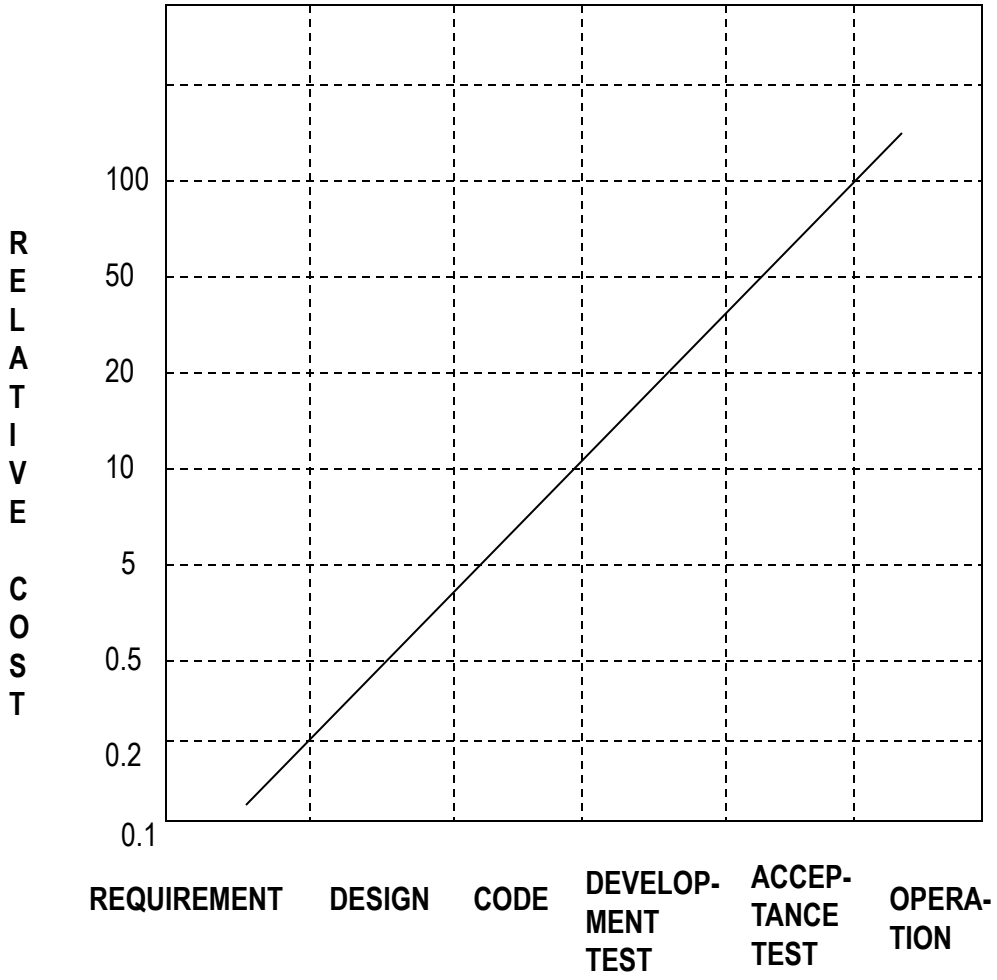
Distribution of Effort …Contd…

- Over the years, we have been building sophisticated and integrated software and thereby increasing the efforts of software development, and ...
- Maintenance efforts have also been going up

Therefore,

We ought to be placing an emphasis on decreasing software development & maintenance costs if we want to have minimum cost systems

Relative Cost To Fix An Error



The Requirements Definition and Analysis Phase is Critical to keep the Development and Maintenance Costs to a Minimum


2. Understanding
Requirements

Types Of Requirements

- Functional Requirements
  - what system should do
- Non-Functional Requirements
  - specify the overall quality attributes the system must satisfy.

Functional Requirements

The requirements that specify the Inputs (Stimuli) to the system, the Outputs (Responses) from the system, and the behavioural relationship between them

Functional Requirements (Examples)

- To calculate the compound interest @ 8% per annum on a Fixed Deposit for a period of three years
- To calculate the Tax @ 30% on an annual income equal to and above Rs.2,00,000 but less than Rs.3,00,000
- To invert a Square Matrix (Maximum size 100 X 100) of Real Numbers

Non-Functional Requirements

- The requirements that describe the overall attributes of the system :
  - Portability
  - Reliability
  - Performance
  - Testability
  - Modifiability
  - Security
  - Presentation
  - Reusability
  - Understandability
  - Acceptance Criteria

Non-Functional Requirements (Examples)

Performance

- Number of significant digits to which accuracy should be maintained in a numerical solution
- Maximum response time in a transaction processing system
- Delays and task completion time limits in a real-time system

Non-Functional Requirements (Examples) …Contd...

Environment

- To run the software under a given operating system or use a specific programming language

Security

- The addition and deletion of the book in the system can be carried out by the Library-Chief only

Non-Functional Requirements (Examples) …Contd…

Testability

- All the off-diagonal elements, equal to or more than 10^{-3}, should be printed out by the matrix diagonalisation program

Understandability / Usability

- Experienced Officers should be able to use all of the system functions after a total training of two hours. After this training, the average number of errors made by experienced officers should not exceed two per day

Categories of Requirements

- Satisfiability
- Service
- Criticality
- Stability
- User Categories

Types of Satisfiability

- Normal Requirements
- Expected Requirements
- Exciting Requirements

Normal Requirements

- User responses to specific questions
- Satisfy / dissatisfy in proportion to their presence / absence in system
- The satisfaction is linear and Bi-Directional

Expected Requirements

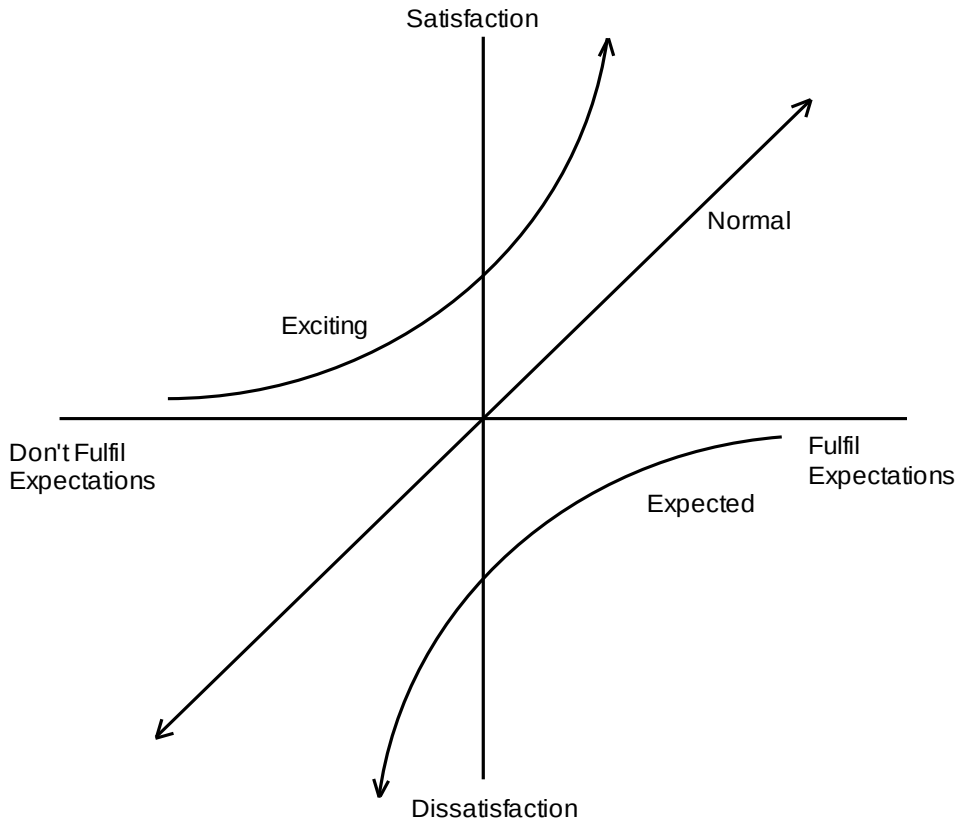
Expected requirements may not be stated by the users, but the developer is expected to meet them. **S**o basic that users may neglect to mention them

- Their presence in solution may meet expectation but not enhance satisfaction
- Absence is very dissatisfying
- "Most Important" for analyst

Exciting Requirements

- Beyond the users' expectations; cannot be elicited from users
- Highly satisfying when met
- Their absence doesn't dissatisfy because they are not expected

Satisfiability



Normal, Expected and Exciting Requirements

The Trend over the Years

- Exciting requirements often become normal requirements

- Some normal requirements become expected requirements
- For example, on-line help feature was first introduced in the UNIX system in the form of *man* pages. At that time, it was an exciting feature. Later, other users started demanding it as part of their systems. Now a days, users do not ask for it, but the developer is expected to provide it.

Type of Services

Function-Related Requirements

- *Users' Information Needs*
- *Information Processing Aspects*

Presentation-Related Requirements

- *Manner or form of presenting information*

Type of Services …Contd…

Performance-Related Requirements

- *Time-criticality of Information*
- *Optimum Use of Resources*

Administration-Related Requirements

- *Controls on the information processing*
- *Organizational Climate*

Types of Stability

- Stable
- Unstable
  - Indicate the changes and their expected dates

Criticality

- Classification
  - *Mandatory*
  - *Desirable*
  - *Non-Essential*
- Decision
  - *In consultation with the users by the application of appropriate prioritization rules*
- These define the problem domain
- Can help in phased development, based on criticality

User Categories

- Compile requirements against each defined user category to assure completeness of requirements
- Broadly they are of two kinds. Those who dictate the policies of the system and those who utilise the services of the system. All of them use the system. It is important that all stakeholders are identified and their requirements are captured.

Modeling Requirements

- Every software system has the following essential characteristics:
- It has a boundary. The boundary separates what is with in system scope and what is outside
- It takes inputs from external agents and generates outputs
- It has processes which collaborate with each other to generate the outputs
- These processes operate on data by creating, modifying, destroying, and querying it
- The system may also use data stores to store data which has a life beyond the system

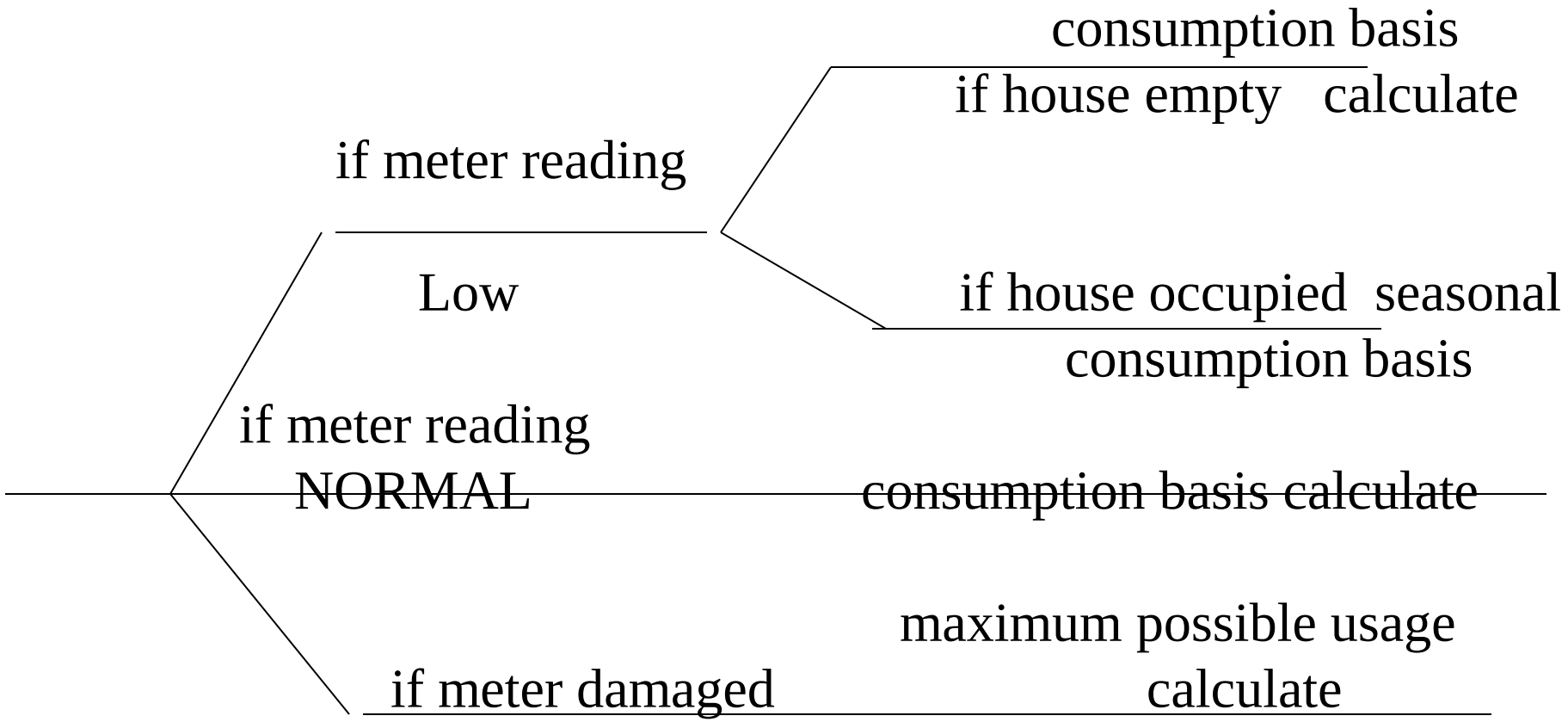
SSADM tools

- Data Flow Diagram (DFD) for modelling processes and their interactions.
- Entity Relationship Diagram (ERD) for modelling data and their relationships.
- Data Dictionary to specify data
- Decision Tables and Decision Trees to model complex decisions.
- Structured English to paraphrase process algorithms.
- State Transition Diagram to model state changes of the system.

DECISION TREE

- **A decision tree represents complex decisions in the form of a tree. Though visually it is appealing, it can soon get out of hand when the number and complexity of decisions increase. An example is given below.**
  First the textual statement is given and then the corresponding decision tree is given:

  Rules for electricity billing are as below:
- **If the meter reading is "OK", calculate on consumption basis(i.e. meter reading)**
  If the meter reading appears "LOW", then check if the house is occupied
  If the house is occupied, calculate on seasonal consumption basis otherwise calculate on consumption basis
  If the meter is damaged, calculate based on maximum possible electricity usage


consumption basis
if house empty calculate
if meter reading
Low
if house occupied seasonal
consumption basis
if meter reading
NORMAL
consumption basis calculate
maximum possible usage
if meter damaged
calculate

DECISION TABLE

- **There are two types of decision tables, binary-valued(yes or no) and multi-valued. An example follows:**

  ELECTRICITY BILL CALCULATION BASED ON CUSTOMER CLASS

  If a customer uses electricity for domestic purposes and if the consumption is less than 300 units per month then bill with minimum monthly charges.
  Domestic customers with a consumption of 300 units or more per month are billed at special rate.
  Non-domestic users are charged double that of domestic users (minimum and special rates are double).

BINARY VALUED DECISION TABLE

Domestic Customer	**Y**	**Y**	**N**	**N**
Consumption < 300 units/month	**Y**	**N**	**Y**	**N**
Minimum Rate	**Y**	**N**	**N**	**N**
Special Rate	**N**	**Y**	**N**	**N**
Double Minimum Rate	**N**	**N**	**Y**	**N**
Double Special Rate	**N**	**N**	**N**	**Y**

MULTIVALUED DECISION TABLE

CUSTOMER	D	D	N	N
CONSUMPTION	>=300	<300	>=300	<300
RATE	S	M	2S	2M

- **Like decision trees, binary-value decision tables can grow large if the number of rules increase. Multi-valued decision tables have an edge. In the above example, if we add a new class of cutomers, called Academic, with the rules:**

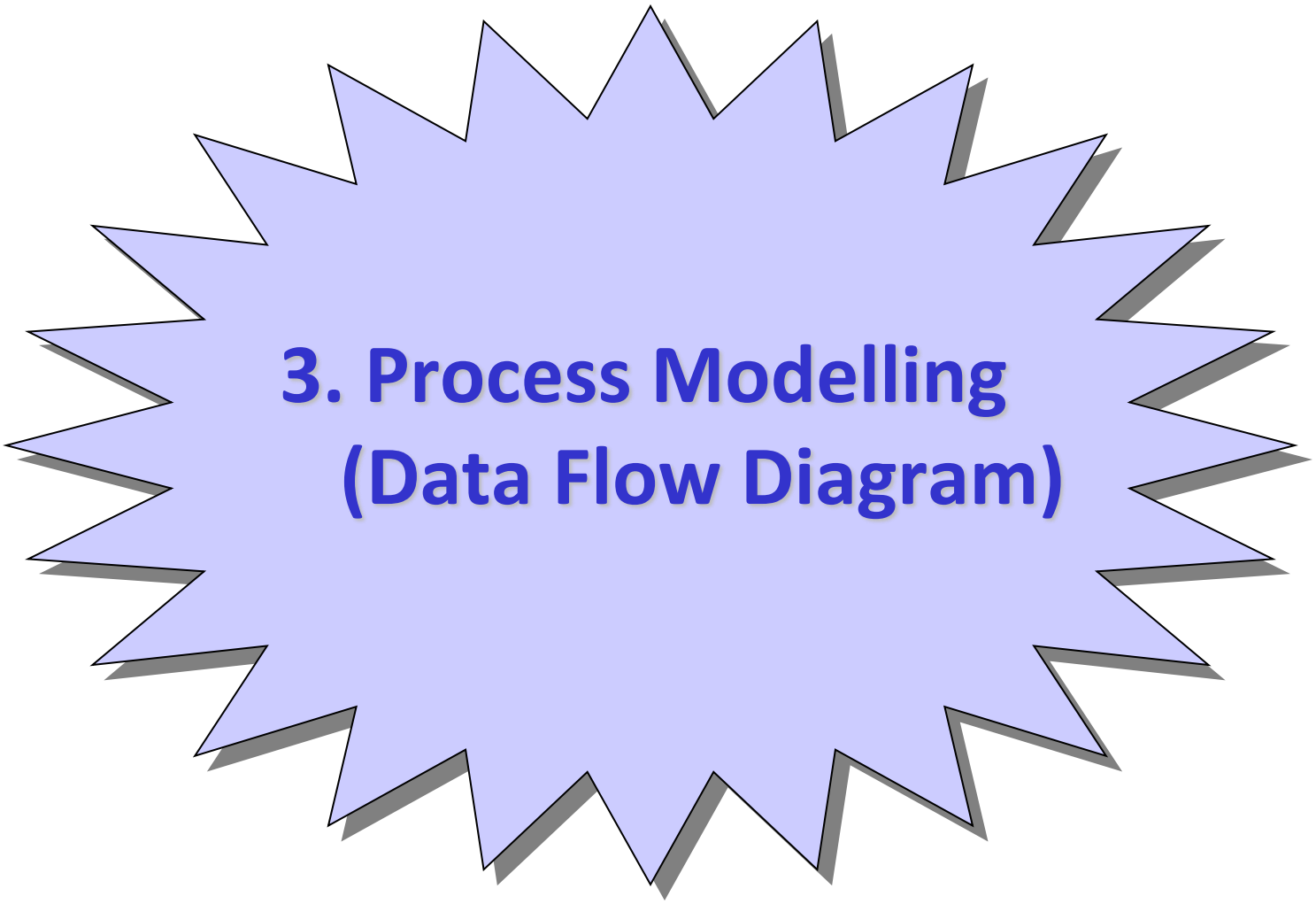
  If the consumption is less than 300 units per month then bill with concessional rates. Otherwise bill with twice the concessional rates. then new tables will look like the following:

BINARY VALUED DECISION TABLE

ACADEMIC	N	N	N	N	Y	Y
DOMESTIC CUSTOMER	Y	Y	N	N	N	N
CONSUMPTION < 300UPM	Y	N	Y	N	Y	N
MINIMUM RATE	Y	N	N	N	N	N
SPECIAL RATE	N	Y	N	N	N	N
TWICE MINUMUM RATE	N	N	Y	N	N	N
TWICE SPECIAL RATE	N	N	N	Y	N	N
CONCESSION RATE	N	N	N	N	Y	N
TWICE CONCESSION RATE	N	N	N	N	N	Y

MULTIVALUED DECISION TABLE

CUSTOMER	DOMESTIC	DOMESTIC	NONDOMESTIC	NONDOMESTIC	ACADEMIC	ACADEMIC
CONSUMPTION	>=300	<300	>=300	<300	>=300	<300
RATE	SPL	MIN	TWICE SPL	TWICE MIN	TWICE CONC	CONCESSIONAL


3. Process Modelling
(Data Flow Diagram)

Prototype

- Prototyping is the technique of constructing a partial implementation of a system so that customers, users or developers can learn more about a problem or a solution to that problem.

- The principal function of the development of “Prototype” is to establish the requirements

Implementation of a Subset

- A subset is also a partial implementation. The purpose of a subset is to provide early functionality, whereas the purpose of the prototype is to learn more about the problem or its solution.

Classes of Prototyping

- Throwaway
- Evolutionary
- Incremental

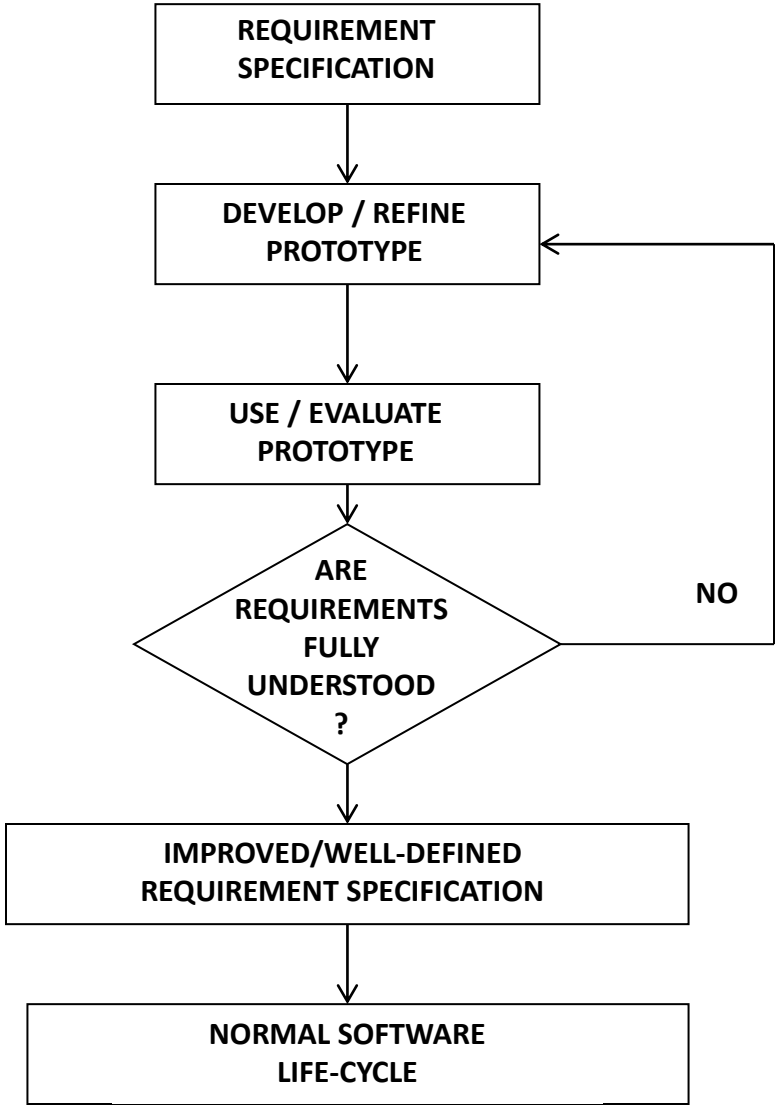
Throwaway Prototype

- Prototype Software is constructed
- Used to learn more about problem / it's solution
- Discarded after desired knowledge is gained
- The final system, adapting normal Software Life Cycle, using the Prototype as a Model, is coded afresh.

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

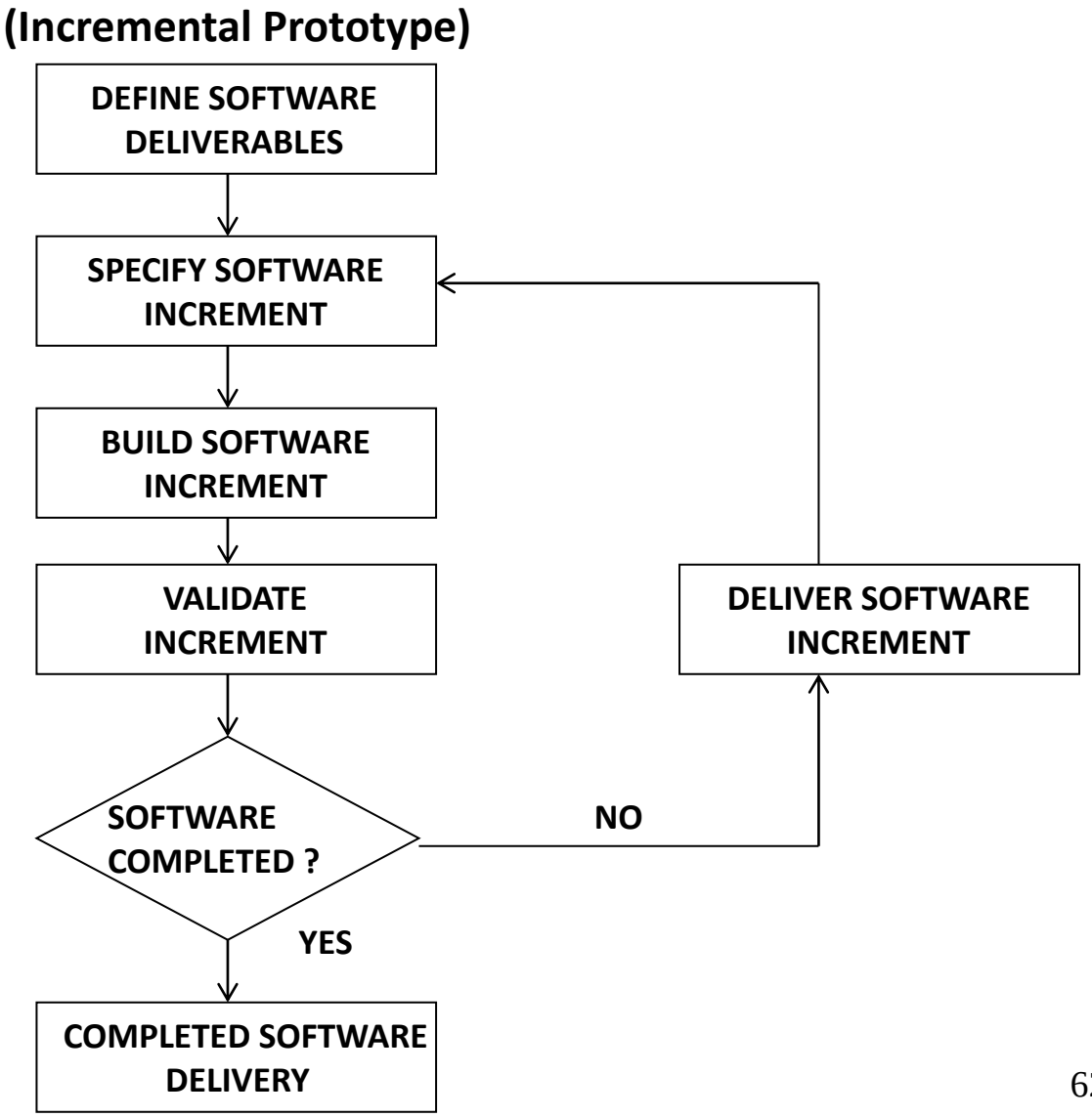
Software Development Life Cycle Using Throwaway Prototype



Evolutionary Prototype

- Prototype Software is constructed incorporating major functions
- Used and inputs for refinement/enhancement are provided by the user
- Prototype is refined to meet the now better understood needs
- Steps 3 and 2 are repeated until the prototype system satisfies all needs and has thus evolved into the real system

Variation of Evolutionary Prototype



Comparison

Throwaway Vs. Evolutionary

Subject	Throw-Away	Evolutionary
Development Approach	Quick and Dirty; No Rigour	No Sloppiness Rigourous
What to Build	Build only Difficult Parts	Build Understood parts first build on solid foundation
Design Drivers	Optimize Development Time	Optimize Modifiability
Ultimate Result	Throw it away, once the requirements are understood	Evolve it to be delivered as the Final System

Advantages

- Systems can be developed much faster
- Maintenance / Development Cost is minimised
- Better Satisfaction on Functional Requirements

Disadvantages

- Undue User Expectation
- Inconsistencies Between Prototype and Final System
- Encouragement of End-User Computing
- Final System Inefficiencies
- Lack of Attention to Human-Factors Guidelines
- Inattention to Proper Analysis

When to Go for Prototyping ?

- **System**
  - is Dynamic
  - Extensive User Dialogues
  - is On-Line

When Not To Go For Prototyping ?

- **System**
  - is Stable
  - is Batch
  - makes Little Use of User Dialogues
  - does Extensive Number Crunching

Guidelines for writing requirements

- **Invent a standard format and use it for all requirements**
- **Use language in a consistent way. Use**

  shall **for mandatory requirements,**

  should **for desirable requirements**
- **Use text highlighting to identify key parts of the requirement**

Avoid the use of computer jargon !!!

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**
- **System requirements may be expressed using system models**

Requirements and design

- In principle, **requirements should state** what the system should **do and the design should describe** how it does this
- In practice, **requirements and design are inseparable**
  - A system architecture may be designed to structure the requirements
  - The system may inter-operate with other systems that generate design requirements
  - The use of a specific design may be a domain requirement

Requirements document requirements

- Specify external system behaviour
- Specify implementation constraints
- 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


6. Computer
Aided
Software
Engineering (CASE)

What is CASE ?

- "CASE" is the Discipline of Computer based Assistance for development and maintenance of software

- Case Tools are available for different phases of software development life cycle

What Case Tools Can Do ?

- Graphic Tool
  - *Data Flow Diagram*
  - *Flowchart*
  - *Entity Relationship Diagram*
  - *Structure Charts*
  - *State-Transition Diagrams*
- Dictionary Tools
  - *Contents of Files*
  - *Inputs and Outputs*
  - *Properties of Data Elements*
  - *Logic Rules for Processes*

What Case Tools Can Do ? …Contd…

- Prototyping Tools
  - *External Design of Inputs*
  - *Screens*
  - *Forms*
  - *Outputs*

- Quality Checking Tools
  - *Dictionary Specification*
  - *Correctness of DFD*
  - *Correctness of ERD*
  - *Consistency Errors*
  - *Completeness Errors*

Cross-Referencing Through Central Repository

What Case Tools Can Do ? … Contd …

- Code Generators
- Cost-Benefit Analysis Tools
- Project Management Tools
- Configuration Management Tools
- Documentation Assemblers
  - Technical and Non-Technical
  - With Interfaces to popular Word Processors
- Test Data Generators
- Path Coverage Analyzers

Popular Case Tools

- Excelerator
- Design Aid
- Information Engineering Facility
- Structured Architect
- Analyst/Designer Toolkit
- Prokit-Workbench
- Foundation
- Teamwork

TCS - Case Tools

- **For Requirement Analysis**
  - ER Modeler
  - Process Modeler
  - MasterCraft
  - Revine (Reverse Engineering Tool)

Caution

Do not be lured into displacing
your goal with another !
Understanding the Problem

Following the tool

NO, NEVER !

TYPES OF REQUIREMENTS

Physical Environment

- Where is the equipment to function ?
- Is there one location or several ?
- Are there any environmental restrictions, such as temperature, humidity, or magnetic interference ?

Interfaces

* Is the input coming from or more other systems ?

* Is the output to one or more other systems ?

- Is there a prescribed way in which the data must be formatted ?
- Is there a prescribed medium that the data must use ?

Users and Human factors

* Who will use the system ?

- Will there be several types of users ?
- What is the skill level of each type of user ?
- What kind of training will be required for each type of user ?
- How easy will it be for a user to understand and use the system?
- How difficult will it be for a user to misuse the system ?

TYPES OF REQUIREMENTS

Physical Environment

- Where is the equipment to function ?
- Is there one location or several ?
- Are there any environmental restrictions, such as temperature, humidity, or magnetic interference ?

Interfaces

* Is the input coming from or more other systems ?

* Is the output to one or more other systems ?

- Is there a prescribed way in which the data must be formatted ?
- Is there a prescribed medium that the data must use ?

Users and Human factors

* Who will use the system ?

- Will there be several types of users ?
- What is the skill level of each type of user ?
- What kind of training will be required for each type of user ?
- How easy will it be for a user to understand and use the system?
- How difficult will it be for a user to misuse the system ?

TYPES OF REQUIREMENTS

Physical Environment

- Where is the equipment to function ?
- Is there one location or several ?
- Are there any environmental restrictions, such as temperature, humidity, or magnetic interference ?

Interfaces

* Is the input coming from or more other systems ?
* Is the output to one or more other systems ?
- Is there a prescribed way in which the data must be formatted ?
- Is there a prescribed medium that the data must use ?

Users and Human factors

* Who will use the system ?
- Will there be several types of users ?
- What is the skill level of each type of user ?
- What kind of training will be required for each type of user ?
- How easy will it be for a user to understand and use the system?
- How difficult will it be for a user to misuse the system ?

Functionality

* What will the system do ?
* When will the system do it ?
* Are there several modes of operation ?
* How and when can the system be changed or enhanced ?
* Are there constraints on execution speed, response time ?

Documentation

*How much documentation is required?
*Should it be on-line, in book format or both ?
* To what audience is each type of documentation addressed ?

Data

For both input and output, what should be format of the data be ?

How often will they be received or sent ?

How accurate must they be ?

How much data flow through the system ?

Must any data be retained for any period of time ?

Resources

- What materials, personnel or other resources are required to build, use and maintain the system ?
- What skills must the developers have ?
- What are the requirements for power, heating or air conditioning ?
- Is there a limit on the amount of money to be spent on development or on hardware and software ?

Security

Must access to the system or to information be controlled ?

How often will the system be backed up ?

Should precautions be taken against fire, water damage or theft ?

Quality Assurance

What are the requirements for reliability, availability, maintainability, security and other quality attributes ?

How the characteristics of the system be demonstrated to others ?

Must the system isolate and detect faults ?

What is the MTBF ?

Is there a maximum time allowed for restarting the system after a failure?

How Users and Developers view each other ?

How developer see users	**How users see developers ?**
Users don't know what they want.	Developer's don't understand operational needs
Users want everything right now.	Developer's can't translate clearly stated needs into a successful system
User's can't prioritize needs.	Developers say no all the time.
Users refuse to take responsibility for the system	Developers are always late.
Users are unwilling to compromise	Developers are unable to respond
User's can't remain on schedule	quickly to legitimately changing needs
User's are unable to provide a Usable statement of needs	Developers try to tell us how to do our job
	Developer's can't translate clearly stated needs into a system