

## UNIT II SOFTWARE

### REQUIREMENTS

- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed
- Requirements may be functional or non-functional
  - Functional requirements describe system services or functions
  - Non-functional requirements is a constraint on the system or on the development process

#### Types of requirements

- User requirements
  - Statements in natural language (NL) 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

### Functional and Non-Functional

#### Functional requirements

- Functionality or services that the system is expected to provide.
- Functional requirements may also explicitly state what the system shouldn't do.
- Functional requirements specification should be:
  - Complete: All services required by the user should be defined
  - Consistent: should not have contradictory definition (also avoid ambiguity → don't leave room for different interpretations)

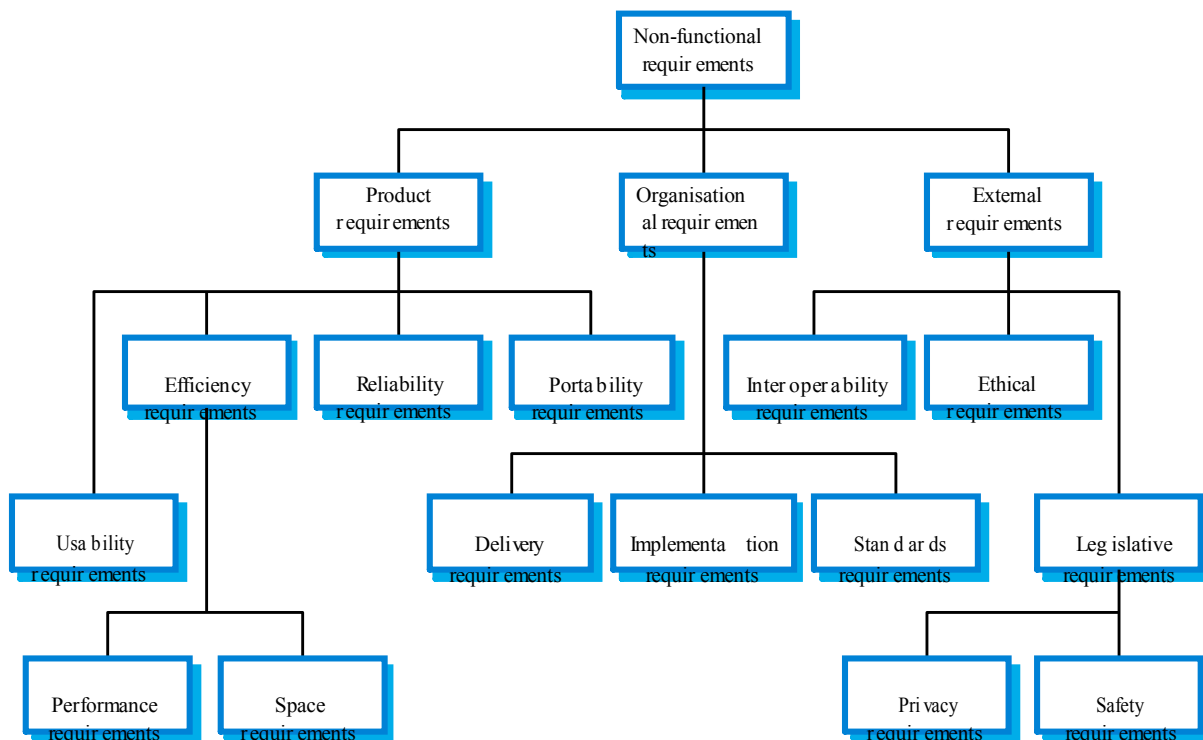
#### Examples of functional requirements

- The LIBSYS system
- A library system that provides a single interface to a number of databases of articles in different libraries.
- Users can search for, download and print these articles for personal study.
- 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

- Requirements that are not directly concerned with the specific functions delivered by the system
- Typically relate to the system as a whole rather than the individual system features
- Often could be deciding factor on the survival of the system (e.g. reliability, cost, response time)

### Non-Functional requirements classifications:



### Domain requirements

- Domain requirements are derived from the application domain of the system rather than from the specific needs of the system users.
- May be new functional requirements, constrain existing requirements or set out how particular computation must take place.
- Example: tolerance level of landing gear on an aircraft (different on dirt, asphalt, water), or what happens to fiber optics line in case of sever weather during winter Olympics (Only domain-area experts know)

### Product requirements

- Specify the desired characteristics that a system or subsystem must possess.
- Most NFRs are concerned with specifying constraints on the behaviour of the executing system.

### Specifying product requirements

- Some product requirements can be formulated precisely, and thus easily quantified
  - Performance
  - Capacity

- Others are more difficult to quantify and, consequently, are often stated informally
  - Usability

### Process requirements

- Process requirements are constraints placed upon the development process of the system
- Process requirements include:
  - Requirements on development standards and methods which must be followed
  - CASE tools which should be used
  - The management reports which must be provided

### Examples of process requirements

- The development process to be used must be explicitly defined and must be conformant with ISO 9000 standards
- The system must be developed using the XYZ suite of CASE tools
- Management reports setting out the effort expended on each identified system component must be produced every two weeks
- A disaster recovery plan for the system development must be specified

### External requirements

- May be placed on both the product and the process
- Derived from the environment in which the system is developed
- External requirements are based on:
  - application domain information
  - organisational considerations
  - the need for the system to work with other systems
  - health and safety or data protection regulations
  - or even basic natural laws such as the laws of physics

### Examples of external requirements

- Medical data system The organisation's data protection officer must certify that all data is maintained according to data protection legislation before the system is put into operation.
- Train protection system The time required to bring the train to a complete halt is computed using the following function:
- The deceleration of the train shall be taken as:

$$g_{\text{train}} = g_{\text{control}} + g_{\text{gradient}}$$

where:

$g_{\text{gradient}} = 9.81 \text{ ms}^{-2} * \text{compensated gradient} / \alpha$  and where the values of  $9.81 \text{ ms}^{-2} / \alpha$  are known for the different types of train.

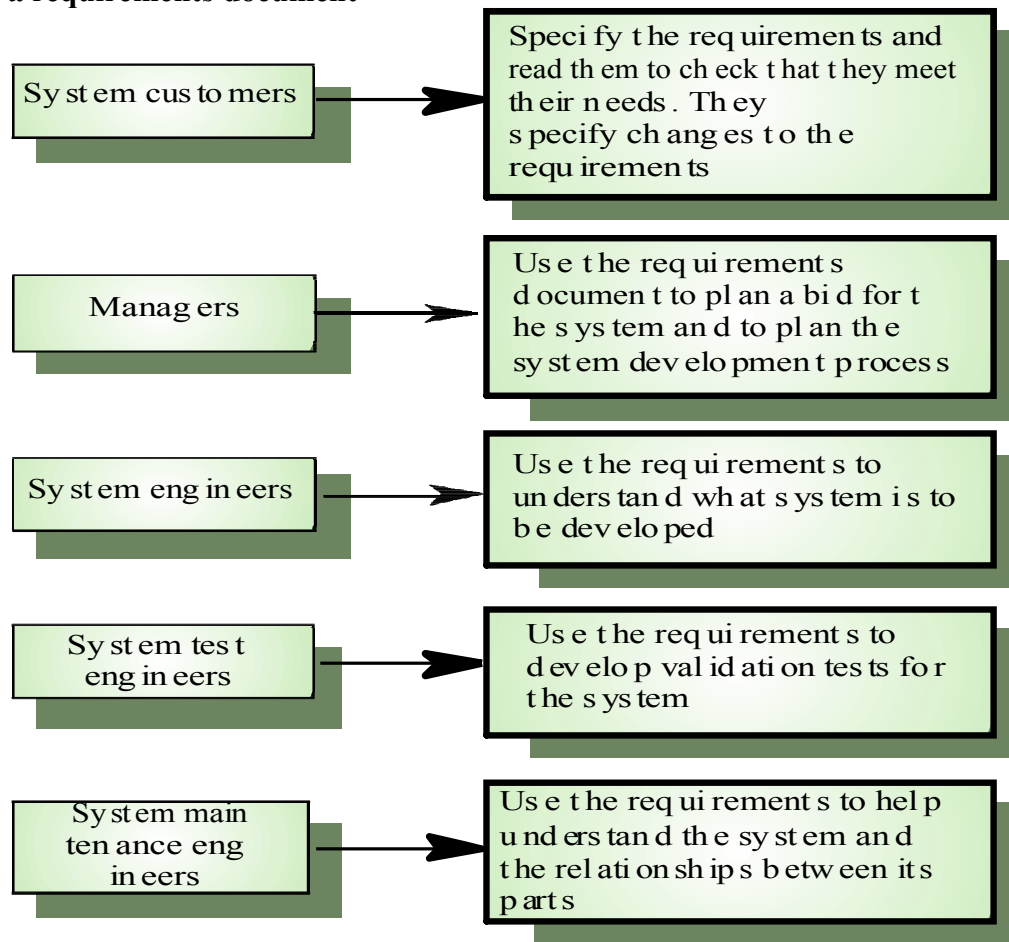
$g_{\text{control}}$  is initialised at  $0.8 \text{ ms}^{-2}$  - this value being parameterised in order to remain adjustable. The illustrates an example of the train's deceleration by using the parabolas derived from the above formula where there is a change in gradient before the (predicted) stopping point of the train.

### Software Document

- Should provide for communication among team members

- Should act as an information repository to be used by maintenance engineers
- Should provide enough information to management to allow them to perform all program management related activities
- Should describe to users how to operate and administer the system
- 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

### Users of a requirements document



### Process Documentation

- Used to record and track the development process
  - Planning documentation
  - Cost, Schedule, Funding tracking
  - Schedules
  - Standards

- This documentation is created to allow for successful management of a software product
- Has a relatively short lifespan
  - Only important to internal development process
  - Except in cases where the customer requires a view into this data
- Some items, such as papers that describe design decisions should be extracted and moved into the *product* documentation category when they become implemented
  - Product Documentation
- Describes the delivered product
- Must evolve with the development of the software product
- Two main categories:
  - System Documentation
  - User Documentation

### **Product Documentation**

- System Documentation
  - Describes how the system works, but not how to operate it
- Examples:
  - Requirements Spec
  - Architectural Design
  - Detailed Design
  - Commented Source Code
    - Including output such as JavaDoc
  - Test Plans
    - Including test cases
  - V&V plan and results
  - List of Known Bugs
- User Documentation has two main types
  - End User
  - System Administrator
    - In some cases these are the same people
  - The target audience must be well understood!
- There are five important areas that should be documented for a formal release of a software application
  - These do not necessarily each have to have their own document, but the topics should be covered thoroughly
- Functional Description of the Software
- Installation Instructions
- Introductory Manual
- Reference Manual
- System Administrator's Guide

### **Document Quality**

- Providing thorough and professional documentation is important for any size product development team

- The problem is that many software professionals lack the writing skills to create professional level documents

### **Document Structure**

- All documents for a given product should have a similar structure
  - A good reason for product standards
- The IEEE Standard for User Documentation lists such a structure
  - It is a superset of what most documents need
- The authors –best practices II are:
- Put a cover page on all documents
- Divide documents into chapters with sections and subsections
- Add an index if there is lots of reference information
- Add a glossary to define ambiguous terms

### **Standards**

- Standards play an important role in the development, maintenance and usefulness of documentation
- Standards can act as a basis for quality documentation
  - But are not good enough on their own
    - Usually define high level content and organization
- There are three types of documentation standards

#### **1.Process Standards**

- Define the approach that is to be used when creating the documentation
- Don't actually define any of the content of the documents

#### **2. Product Standards**

- Goal is to have all documents created for a specific product attain a consistent structure and appearance
  - Can be based on organizational or contractually required standards
- Four main types:
  - Documentation Identification Standards
  - Document Structure Standards
  - Document Presentation Standards
  - Document Update Standards
- One caveat:
  - Documentation that will be viewed by end users should be created in a way that is best consumed and is most attractive to them
  - Internal development documentation generally does not meet this need

#### **3. Interchange Standards**

- Deals with the creation of documents in a format that allows others to effectively use
  - PDF may be good for end users who don't need to edit
  - Word may be good for text editing

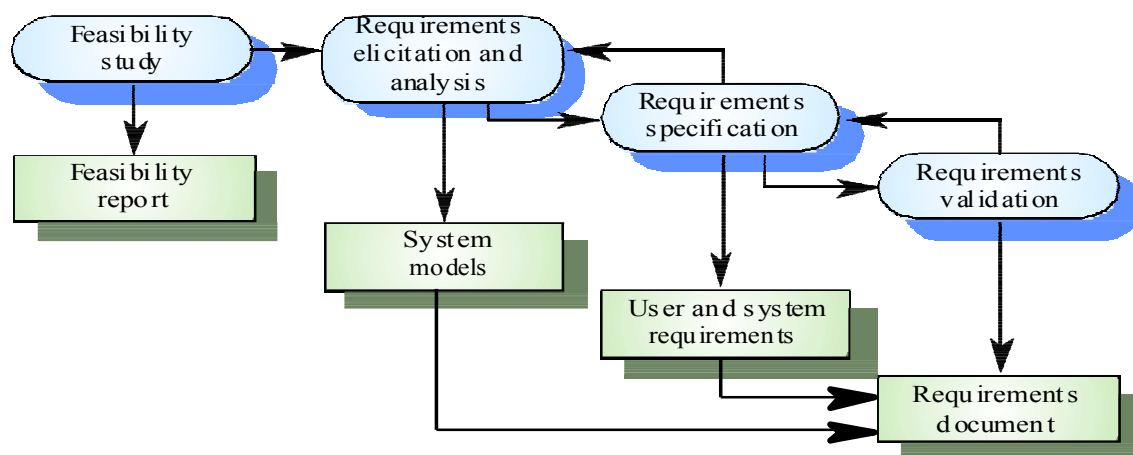
- Specialized CASE tools need to be considered
- This is usually not a problem within a single organization, but when sharing data between organizations it can occur
  - This same problem is faced all the time during software integration

### Other Standards

- IEEE
  - Has a published standard for user documentation
  - Provides a structure and superset of content areas
  - Many organizations probably won't create documents that completely match the standard
- Writing Style
  - Ten -best practices II when writing are provided
  - Author proposes that group edits of important documents should occur in a similar fashion to software walkthroughs

### Requirement Engineering Process

- The requirements engineering process includes a feasibility study, requirements elicitation and analysis, requirements specification and requirements management



### Feasibility Studies

- A feasibility study decides whether or not the proposed system is worthwhile
- A short focused study that checks
  - If the system contributes to organisational objectives
  - If the system can be engineered using current technology and within budget
  - If the system can be integrated with other systems that are used
- Based on information assessment (what is required), information collection and report writing
- Questions for people in the organisation
  - What if the system wasn't implemented?
  - What are current process problems?
  - How will the proposed system help?

- What will be the integration problems?
- Is new technology needed? What skills?
- What facilities must be supported by the proposed system?

### **Elicitation and analysis**

- Sometimes called requirements elicitation or requirements discovery
- Involves technical staff working with customers to find out about
  - the application domain
  - the services that the system should provide
  - the system's operational constraints
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc.
  - These are called stakeholders

### **Problems of requirements analysis**

- Stakeholders don't know what they really want
- Stakeholders express requirements in their own terms
- Different stakeholders may have conflicting requirements
- Organisational and political factors may influence the system requirements
- The requirements change during the analysis process
  - New stakeholders may emerge and the business environment change

### **System models**

- Different models may be produced during the requirements analysis activity
- Requirements analysis may involve three structuring activities which result in these different models
  - Partitioning – Identifies the structural (part-of) relationships between entities
  - Abstraction – Identifies generalities among entities
  - Projection – Identifies different ways of looking at a problem
- System models will be covered on January 30

### **Scenarios**

- Scenarios are descriptions of how a system is used in practice
- They are helpful in requirements elicitation as people can relate to these more readily than abstract statement of what they require from a system
- Scenarios are particularly useful for adding detail to an outline requirements description

### **Ethnography**

- A social scientist spends a considerable time observing and analysing how people actually work
- People do not have to explain or articulate their work
- Social and organisational factors of importance may be observed
- Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models



**Requirements validation**

- Concerned with demonstrating that the requirements define the system that the customer really wants
- Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error
- Requirements checking
  - Validity
  - Consistency
  - Completeness
  - Realism
  - Verifiability

**Requirements validation techniques**

- Reviews
  - Systematic manual analysis of the requirements
- Prototyping
  - Using an executable model of the system to check requirements.
- Test-case generation
  - Developing tests for requirements to check testability
- Automated consistency analysis
  - Checking the consistency of a structured requirements description

**Requirements management**

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development
- Requirements are inevitably incomplete and inconsistent
  - New requirements emerge during the process as business needs change and a better understanding of the system is developed
  - Different viewpoints have different requirements and these are often contradictory

**Software prototyping**

Incomplete versions of the software program being developed. Prototyping can also be used by end users to describe and prove requirements that developers have not considered

**Benefits:**

The software designer and implementer can obtain feedback from the users early in the project. The client and the contractor can compare if the software made matches the software specification, according to which the software program is built.

It also allows the software engineer some insight into the accuracy of initial project estimates and whether the deadlines and milestones proposed can be successfully met.

**Process of prototyping****1. Identify basic requirements**

Determine basic requirements including the input and output information desired. Details, such as security, can typically be ignored.

## 2. Develop Initial Prototype

The initial prototype is developed that includes only user interfaces. (See Horizontal Prototype, below)

## 3. Review

The customers, including end-users, examine the prototype and provide feedback on additions or changes.

## 4. Revise and Enhance the Prototype

Using the feedback both the specifications and the prototype can be improved. Negotiation about what is within the scope of the contract/product may be necessary. If changes are introduced then a repeat of steps #3 and #4 may be needed.

## Dimensions of prototypes

### 1. Horizontal Prototype

It provides a broad view of an entire system or subsystem, focusing on user interaction more than low-level system functionality, such as database access. Horizontal prototypes are useful for:

- Confirmation of user interface requirements and system scope
- Develop preliminary estimates of development time, cost and effort.

### 2 Vertical Prototypes

A vertical prototype is a more complete elaboration of a single subsystem or function. It is useful for obtaining detailed requirements for a given function, with the following benefits:

- Refinement database design
- Obtain information on data volumes and system interface needs, for network sizing and performance engineering

## Types of prototyping

Software prototyping has many variants. However, all the methods are in some way based on two major types of prototyping: Throwaway Prototyping and Evolutionary Prototyping.

### 1. Throwaway prototyping

Also called close ended prototyping. Throwaway refers to the creation of a model that will eventually be discarded rather than becoming part of the final delivered software. After preliminary requirements gathering is accomplished, a simple working model of the system is constructed to visually show the users what their requirements may look like when they are implemented into a finished system.

The most obvious reason for using Throwaway Prototyping is that it can be done quickly. If the users can get quick feedback on their requirements, they may be able to refine them early in the development of the software. Making changes early in the development lifecycle is extremely cost effective since there is nothing at that point to redo. If a project is changed after a considerable work has been done then small changes could require large efforts to implement since software systems have many dependencies. Speed is crucial in implementing a throwaway prototype, since with a limited budget of time and money little can be expended on a prototype that will be discarded.

Strength of Throwaway Prototyping is its ability to construct interfaces that the users can test. The user interface is what the user sees as the system, and by seeing it in front of them, it is much easier to grasp how the system will work.

## 2. Evolutionary prototyping

Evolutionary Prototyping (also known as breadboard prototyping) is quite different from Throwaway Prototyping. The main goal when using Evolutionary Prototyping is to build a very robust prototype in a structured manner and constantly refine it. "The reason for this is that the Evolutionary prototype, when built, forms the heart of the new system, and the improvements and further requirements will be built.

Evolutionary Prototypes have an advantage over Throwaway Prototypes in that they are functional systems. Although they may not have all the features the users have planned, they may be used on a temporary basis until the final system is delivered.

In Evolutionary Prototyping, developers can focus themselves to develop parts of the system that they understand instead of working on developing a whole system. To minimize risk, the developer does not implement poorly understood features. The partial system is sent to customer sites. As users work with the system, they detect opportunities for new features and give requests for these features to developers. Developers then take these enhancement requests along with their own and use sound configuration-management practices to change the software-requirements specification, update the design, recode and retest.

## 3. Incremental prototyping

The final product is built as separate prototypes. At the end the separate prototypes are merged in an overall design.

## 4. Extreme prototyping

Extreme Prototyping as a development process is used especially for developing web applications. Basically, it breaks down web development into three phases, each one based on the preceding one. The first phase is a static prototype that consists mainly of HTML pages. In the second phase, the screens are programmed and fully functional using a simulated services layer. In the third phase the services are implemented. The process is called Extreme Prototyping to draw attention to the second phase of the process, where a fully-functional UI is developed with very little regard to the services other than their contract.

## Advantages of prototyping

**1. Reduced time and costs:** Prototyping can improve the quality of requirements and specifications provided to developers. Because changes cost exponentially more to implement as they are detected later in development, the early determination of what the user really wants can result in faster and less expensive software.

**2. Improved and increased user involvement:** Prototyping requires user involvement and allows them to see and interact with a prototype allowing them to provide better and more complete feedback and specifications. The presence of the prototype being examined by the user prevents many misunderstandings and miscommunications that occur when each side believe the other understands what they said. Since users know the problem domain better than anyone on the development team does, increased interaction can result in final product that has greater tangible and intangible quality. The final product is more likely to satisfy the users' desire for look, feel and performance.

### **Disadvantages of prototyping**

**1. Insufficient analysis:** The focus on a limited prototype can distract developers from properly analyzing the complete project. This can lead to overlooking better solutions, preparation of incomplete specifications or the conversion of limited prototypes into poorly engineered final projects that are hard to maintain. Further, since a prototype is limited in functionality it may not scale well if the prototype is used as the basis of a final deliverable, which may not be noticed if developers are too focused on building a prototype as a model.

**2. User confusion of prototype and finished system:** Users can begin to think that a prototype, intended to be thrown away, is actually a final system that merely needs to be finished or polished. (They are, for example, often unaware of the effort needed to add error-checking and security features which a prototype may not have.) This can lead them to expect the prototype to accurately model the performance of the final system when this is not the intent of the developers. Users can also become attached to features that were included in a prototype for consideration and then removed from the specification for a final system. If users are able to require all proposed features be included in the final system this can lead to conflict.

**3. Developer misunderstanding of user objectives:** Developers may assume that users share their objectives (e.g. to deliver core functionality on time and within budget), without understanding wider commercial issues. For example, user representatives attending Enterprise software (e.g. PeopleSoft) events may have seen demonstrations of "transaction auditing" (where changes are logged and displayed in a difference grid view) without being told that this feature demands additional coding and often requires more hardware to handle extra database accesses. Users might believe they can demand auditing on every field, whereas developers might think this is feature creep because they have made assumptions about the extent of user requirements. If the developer has committed delivery before the user requirements were reviewed, developers are between a rock and a hard place, particularly if user management derives some advantage from their failure to implement requirements.

**4. Developer attachment to prototype:** Developers can also become attached to prototypes they have spent a great deal of effort producing; this can lead to problems like attempting to convert a limited prototype into a final system when it does not have an appropriate underlying architecture. (This may suggest that throwaway prototyping, rather than evolutionary prototyping, should be used.)

**5. Excessive development time of the prototype:** A key property to prototyping is the fact that it is supposed to be done quickly. If the developers lose sight of this fact, they very well may try to develop a prototype that is too complex. When the prototype is thrown away the precisely developed requirements that it provides may not yield a sufficient increase in productivity to make up for the time spent developing the prototype. Users can become stuck in debates over details of the prototype, holding up the development team and delaying the final product.

**6. Expense of implementing prototyping:** the start up costs for building a development team focused on prototyping may be high. Many companies have development methodologies in place, and changing them can mean retraining, retooling, or both. Many companies tend to just jump into the prototyping without bothering to retrain their workers as much as they should.

A common problem with adopting prototyping technology is high expectations for productivity with insufficient effort behind the learning curve. In addition to training for the use of a prototyping technique, there is an often overlooked need for developing corporate and project

specific underlying structure to support the technology. When this underlying structure is omitted, lower productivity can often result.

### **Best projects to use prototyping**

It has been found that prototyping is very effective in the analysis and design of on-line systems, especially for transaction processing, where the use of screen dialogs is much more in evidence. The greater the interaction between the computer and the user, the greater the benefit is that can be obtained from building a quick system and letting the user play with it.

Systems with little user interaction, such as batch processing or systems that mostly do calculations, benefit little from prototyping. Sometimes, the coding needed to perform the system functions may be too intensive and the potential gains that prototyping could provide are too small.

Prototyping is especially good for designing good human-computer interfaces. "One of the most productive uses of rapid prototyping to date has been as a tool for iterative user requirements engineering and human-computer interface design.

### **Methods**

There are few formal prototyping methodologies even though most Agile Methods rely heavily upon prototyping techniques.

#### **1. Dynamic systems development method**

Dynamic Systems Development Method (DSDM) is a framework for delivering business solutions that relies heavily upon prototyping as a core technique, and is itself ISO 9001 approved. It expands upon most understood definitions of a prototype. According to DSDM the prototype may be a diagram, a business process, or even a system placed into production. DSDM prototypes are intended to be incremental, evolving from simple forms into more comprehensive ones.

DSDM prototypes may be throwaway or evolutionary. Evolutionary prototypes may be evolved horizontally (breadth then depth) or vertically (each section is built in detail with additional iterations detailing subsequent sections). Evolutionary prototypes can eventually evolve into final systems.

The four categories of prototypes as recommended by DSDM are:

- **Business prototypes** – used to design and demonstrate the business processes being automated.
- **Usability prototypes** – used to define, refine, and demonstrate user interface design usability, accessibility, look and feel.
- **Performance and capacity prototypes** - used to define, demonstrate, and predict how systems will perform under peak loads as well as to demonstrate and evaluate other non-functional aspects of the system (transaction rates, data storage volume, response time)
- **Capability/technique prototypes** – used to develop, demonstrate, and evaluate a design approach or concept.

The DSDM lifecycle of a prototype is to:

1. Identify prototype
2. Agree to a plan
3. Create the prototype
4. Review the prototype

## 2. Operational prototyping

Operational Prototyping was proposed by Alan Davis as a way to integrate throwaway and evolutionary prototyping with conventional system development. "[It] offers the best of both the quick-and-dirty and conventional-development worlds in a sensible manner. Designers develop only well-understood features in building the evolutionary baseline, while using throwaway prototyping to experiment with the poorly understood features."

Davis' belief is that to try to "retrofit quality onto a rapid prototype" is not the correct approach when trying to combine the two approaches. His idea is to engage in an evolutionary prototyping methodology and rapidly prototype the features of the system after each evolution.

The specific methodology follows these steps:

- An evolutionary prototype is constructed and made into a baseline using conventional development strategies, specifying and implementing only the requirements that are well understood.
- Copies of the baseline are sent to multiple customer sites along with a trained prototyper.
- At each site, the prototyper watches the user at the system.
- Whenever the user encounters a problem or thinks of a new feature or requirement, the prototyper logs it. This frees the user from having to record the problem, and allows them to continue working.
- After the user session is over, the prototyper constructs a throwaway prototype on top of the baseline system.
- The user now uses the new system and evaluates. If the new changes aren't effective, the prototyper removes them.
- If the user likes the changes, the prototyper writes feature-enhancement requests and forwards them to the development team.
- The development team, with the change requests in hand from all the sites, then produce a new evolutionary prototype using conventional methods.

Obviously, a key to this method is to have well trained prototypers available to go to the user sites. The Operational Prototyping methodology has many benefits in systems that are complex and have few known requirements in advance.

## 3. Evolutionary systems development

Evolutionary Systems Development is a class of methodologies that attempt to formally implement Evolutionary Prototyping. One particular type, called Systems craft is described by John Crinnion in his book: Evolutionary Systems Development.

Systemscraft was designed as a 'prototype' methodology that should be modified and adapted to fit the specific environment in which it was implemented.

Systemscraft was not designed as a rigid 'cookbook' approach to the development process. It is now generally recognised[sic] that a good methodology should be flexible enough to be adjustable to suit all kinds of environment and situation...

The basis of Systemscraft, not unlike Evolutionary Prototyping, is to create a working system from the initial requirements and build upon it in a series of revisions. Systemscraft places heavy emphasis on traditional analysis being used throughout the development of the system.

## 4. Evolutionary rapid development



Evolutionary Rapid Development (ERD) was developed by the Software Productivity Consortium, a technology development and integration agent for the Information Technology Office of the Defense Advanced Research Projects Agency (DARPA).

Fundamental to ERD is the concept of composing software systems based on the reuse of components, the use of software templates and on an architectural template. Continuous evolution of system capabilities in rapid response to changing user needs and technology is highlighted by the evolvable architecture, representing a class of solutions. The process focuses on the use of small artisan-based teams integrating software and systems engineering disciplines working multiple, often parallel short-duration timeboxes with frequent customer interaction. Key to the success of the ERD-based projects is parallel exploratory analysis and development of features, infrastructures, and components with and adoption of leading edge technologies enabling the quick reaction to changes in technologies, the marketplace, or customer requirements.

To elicit customer/user input, frequent scheduled and ad hoc/impromptu meetings with the stakeholders are held. Demonstrations of system capabilities are held to solicit feedback before design/implementation decisions are solidified. Frequent releases (e.g., betas) are made available for use to provide insight into how the system could better support user and customer needs. This assures that the system evolves to satisfy existing user needs.

The design framework for the system is based on using existing published or de facto standards. The system is organized to allow for evolving a set of capabilities that includes considerations for performance, capacities, and functionality. The architecture is defined in terms of abstract interfaces that encapsulate the services and their implementation (e.g., COTS applications). The architecture serves as a template to be used for guiding development of more than a single instance of the system. It allows for multiple application components to be used to implement the services. A core set of functionality not likely to change is also identified and established.

The ERD process is structured to use demonstrated functionality rather than paper products as a way for stakeholders to communicate their needs and expectations. Central to this goal of rapid delivery is the use of the "time box" method. Timeboxes are fixed periods of time in which specific tasks (e.g., developing a set of functionality) must be performed. Rather than allowing time to expand to satisfy some vague set of goals, the time is fixed (both in terms of calendar weeks and person-hours) and a set of goals is defined that realistically can be achieved within these constraints. To keep development from degenerating into a "random walk," long-range plans are defined to guide the iterations. These plans provide a vision for the overall system and set boundaries (e.g., constraints) for the project. Each iteration within the process is conducted in the context of these long-range plans.

Once architecture is established, software is integrated and tested on a daily basis. This allows the team to assess progress objectively and identify potential problems quickly. Since small amounts of the system are integrated at one time, diagnosing and removing the defect is rapid. User demonstrations can be held at short notice since the system is generally ready to exercise at all times.

## **5. Scrum**

Scrum is an agile method for project management. The approach was first described by Takeuchi and Nonaka in "The New New Product Development Game" (Harvard Business Review, Jan-Feb 1986).

## **Tools**

Efficiently using prototyping requires that an organization have proper tools and a staff trained to use those tools. Tools used in prototyping can vary from individual tools like 4th generation programming languages used for rapid prototyping to complex integrated CASE tools. 4th generation programming languages like Visual Basic and ColdFusion are frequently used since they are cheap, well known and relatively easy and fast to use. CASE tools are often developed or selected by the military or large organizations. Users may prototype elements of an application themselves in a spreadsheet.

### **1. Screen generators, design tools & Software Factories**

Commonly used screen generating programs that enable prototypers to show users systems that don't function, but show what the screens may look like. Developing Human Computer Interfaces can sometimes be the critical part of the development effort, since to the users the interface essentially is the system.

Software Factories are Code Generators that allow you to model the domain model and then drag and drop the UI. Also they enable you to run the prototype and use basic database functionality. This approach allows you to explore the domain model and make sure it is in sync with the GUI prototype.

### **2. Application definition or simulation software**

It enables users to rapidly build lightweight, animated simulations of another computer program, without writing code. Application simulation software allows both technical and non-technical users to experience, test, collaborate and validate the simulated program, and provides reports such as annotations, screenshot and schematics. To simulate applications one can also use software which simulate real-world software programs for computer based training, demonstration, and customer support, such as screen casting software as those areas are closely related.

### **3. Sketchflow**

Sketch Flow, a feature of Microsoft Expression Studio Ultimate, gives the ability to quickly and effectively map out and iterate the flow of an application UI, the layout of individual screens and transition from one application state to another.

- Interactive Visual Tool
- Easy to learn
- Dynamic
- Provides environment to collect feedback

### **4. Visual Basic**

One of the most popular tools for Rapid Prototyping is Visual Basic (VB). Microsoft Access, which includes a Visual Basic extensibility module, is also a widely accepted prototyping tool that is used by many non-technical business analysts. Although VB is a programming language it has many features that facilitate using it to create prototypes, including:

- An interactive/visual user interface design tool.
- Easy connection of user interface components to underlying functional behavior.
- Modifications to the resulting software are easy to perform.



## 5. Requirements Engineering Environment

It provides an integrated toolset for rapidly representing, building, and executing models of critical aspects of complex systems.

It is currently used by the Air Force to develop systems. It is: an integrated set of tools that allows systems analysts to rapidly build functional, user interface, and performance prototype models of system components. These modeling activities are performed to gain a greater understanding of complex systems and lessen the impact that inaccurate requirement specifications have on cost and scheduling during the system development process.

REE is composed of three parts. The first, called proto is a CASE tool specifically designed to support rapid prototyping. The second part is called the Rapid Interface Prototyping System or RIP, which is a collection of tools that facilitate the creation of user interfaces. The third part of REE is a user interface to RIP and proto that is graphical and intended to be easy to use.

Rome Laboratory, the developer of REE, intended that to support their internal requirements gathering methodology. Their method has three main parts:

- Elicitation from various sources which means u loose (users, interfaces to other systems), specification, and consistency checking
- Analysis that the needs of diverse users taken together do not conflict and are technically and economically feasible
- Validation that requirements so derived are an accurate reflection of user needs.

## 6. LYMB

LYMB is an object-oriented development environment aimed at developing applications that require combining graphics-based user interfaces, visualization, and rapid prototyping.

## 7. Non-relational environments

Non-relational definition of data (e.g. using Cache or associative models can help make end-user prototyping more productive by delaying or avoiding the need to normalize data at every iteration of a simulation. This may yield earlier/greater clarity of business requirements, though it does not specifically confirm that requirements are technically and economically feasible in the target production system.

## 8. PSDL

PSDL is a prototype description language to describe real-time software.

## Prototyping in the Software Process

### System prototyping

- Prototyping is the rapid development of a system
- In the past, the developed system was normally thought of as inferior in some way to the required system so further development was required
- Now, the boundary between prototyping and normal system development is blurred and many systems are developed using an evolutionary approach

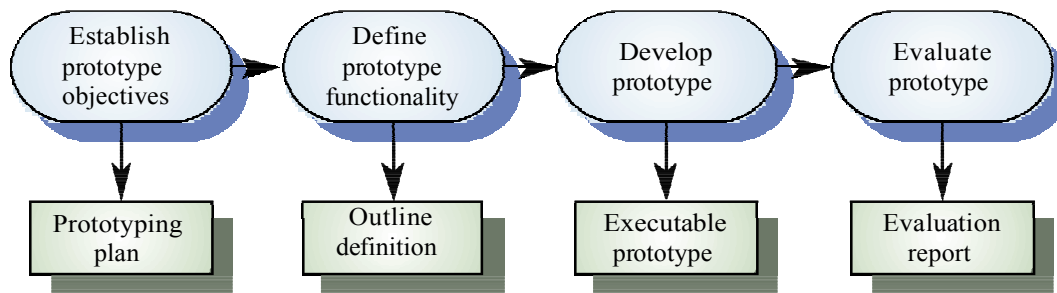
### Uses of system prototypes

- The principal use is to help customers and developers understand the requirements for the system
  - Requirements elicitation. Users can experiment with a prototype to see how the system supports their work
  - Requirements validation. The prototype can reveal errors and omissions in the requirements
- Prototyping can be considered as a risk reduction activity which reduces requirements risks

### Prototyping benefits

- Misunderstandings between software users and developers are exposed
- Missing services may be detected and confusing services may be identified
- A working system is available early in the process
- The prototype may serve as a basis for deriving a system specification
- The system can support user training and system testing

### Prototyping process

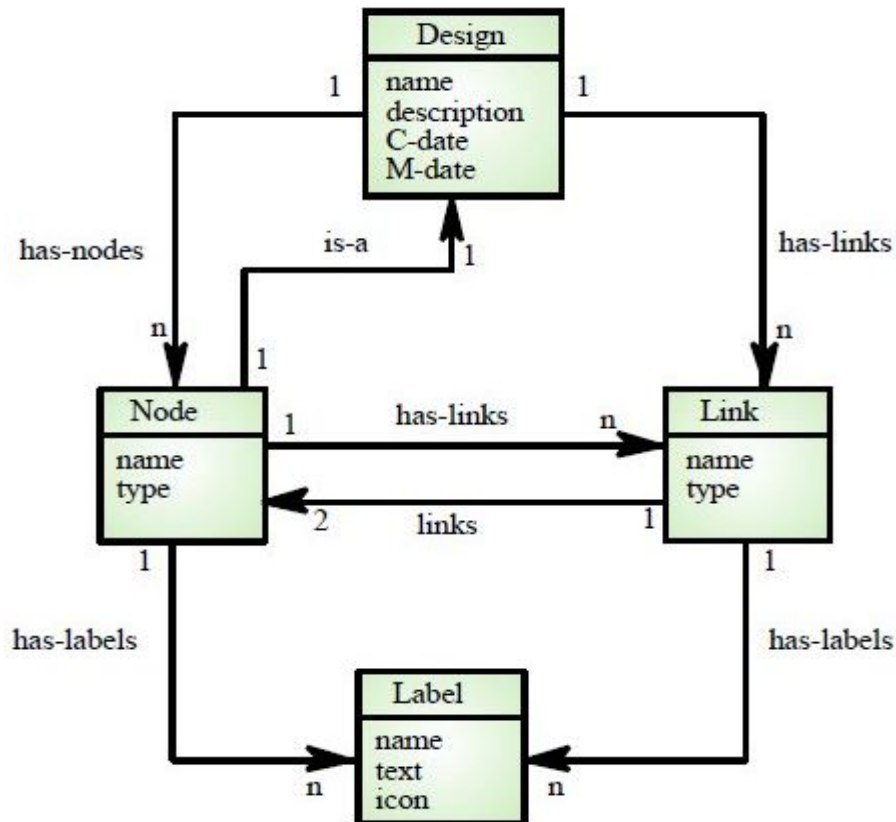


### Prototyping in the software process

- **Evolutionary prototyping**
  - An approach to system development where an initial prototype is produced and refined through a number of stages to the final system
- **Throw-away prototyping**
  - A prototype which is usually a practical implementation of the system is produced to help discover requirements problems and then discarded. The system is then developed using some other development process

### Data Model

- Used to describe the logical structure of data processed by the system
- Entity-relation-attribute model sets out the entities in the system, the relationships between these entities and the entity attributes
- Widely used in database design. Can readily be implemented using relational databases
- No specific notation provided in the UML but objects and associations can be used



### Behavioural Model

- Behavioural models are used to describe the overall behaviour of a system
- Two types of behavioural model are shown here
  - Data processing models that show how data is processed as it moves through the system
  - State machine models that show the systems response to events
- Both of these models are required for a description of the system's behaviour

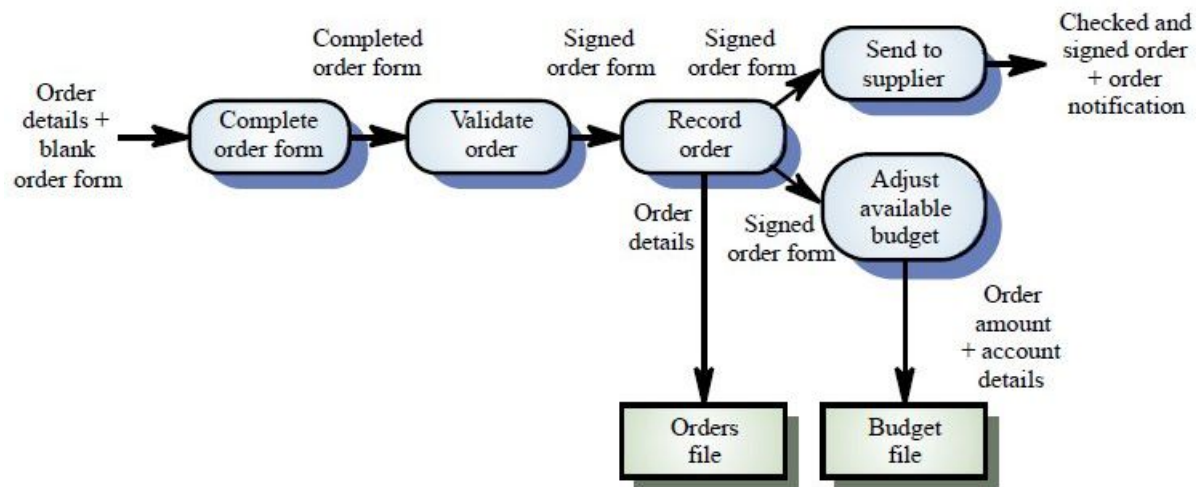
#### 1. Data-processing models

- Data flow diagrams are used to model the system's data processing
- These show the processing steps as data flows through a system
- Intrinsic part of many analysis methods
- Simple and intuitive notation that customers can understand
- Show end-to-end processing of data

#### Data flow diagrams

- DFDs model the system from a functional perspective
- Tracking and documenting how the data associated with a process is helpful to develop an overall understanding of the system
- Data flow diagrams may also be used in showing the data exchange between a system and other systems in its environment

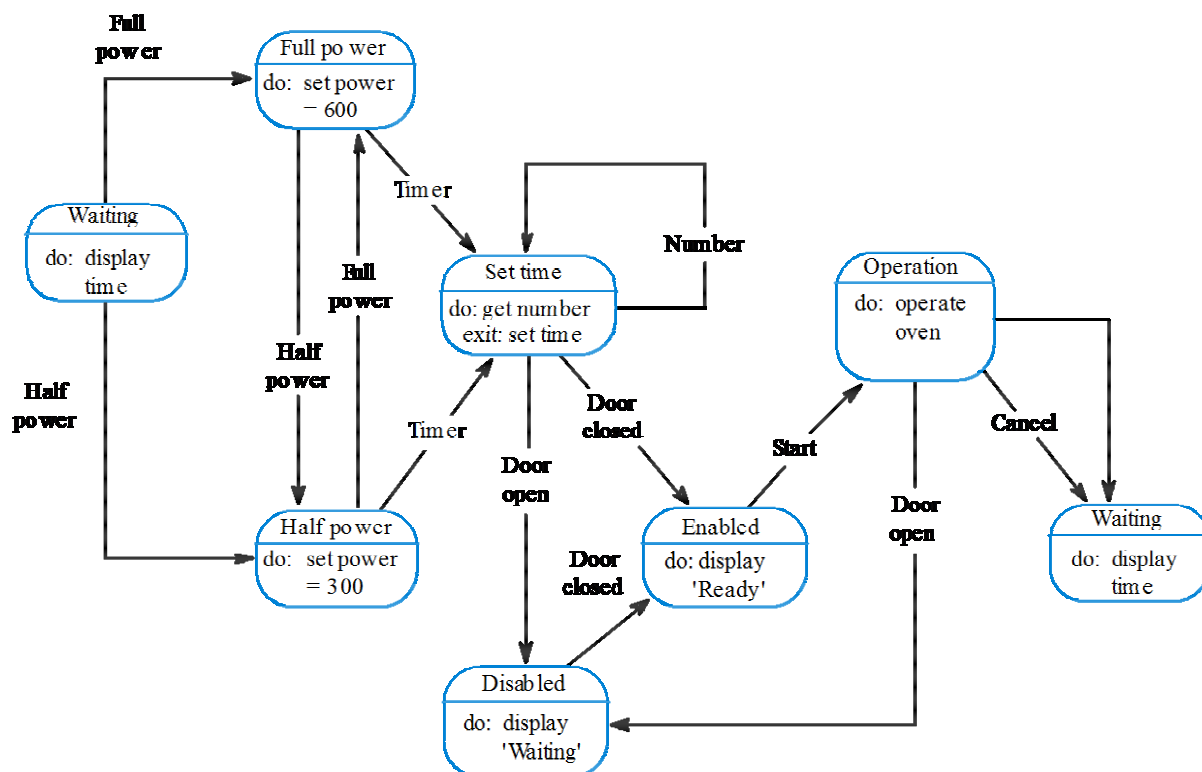
## Order processing DFD



## 2. State machine models

- These model the behaviour of the system in response to external and internal events
- They show the system's responses to stimuli so are often used for modelling real-time systems
- State machine models show system states as nodes and events as arcs between these nodes.
- When an event occurs, the system moves from one state to another
- Statecharts are an integral part of the UML

### Microwave oven model



**Statecharts**

- Allow the decomposition of a model into submodels
- A brief description of the actions is included following the `_do` in each state
- Can be complemented by tables describing the states and the stimuli

**Structured Analysis**

- The data-flow approach is typified by the Structured Analysis method (SA)
- Two major strategies dominate structured analysis
  - `_Old` method popularised by DeMarco
  - `_Modern` approach by Yourdon

**DeMarco**

- A top-down approach
  - The analyst maps the current physical system onto the current logical data-flow model
- The approach can be summarised in four steps:
  - Analysis of current physical system
  - Derivation of logical model
  - Derivation of proposed logical model
  - Implementation of new physical system

**Modern structured analysis**

- Distinguishes between user's real needs and those requirements that represent the external behaviour satisfying those needs
- Includes real-time extensions
- Other structured analysis approaches include:
  - Structured Analysis and Design Technique (SADT)
  - Structured Systems Analysis and Design Methodology (SSADM)

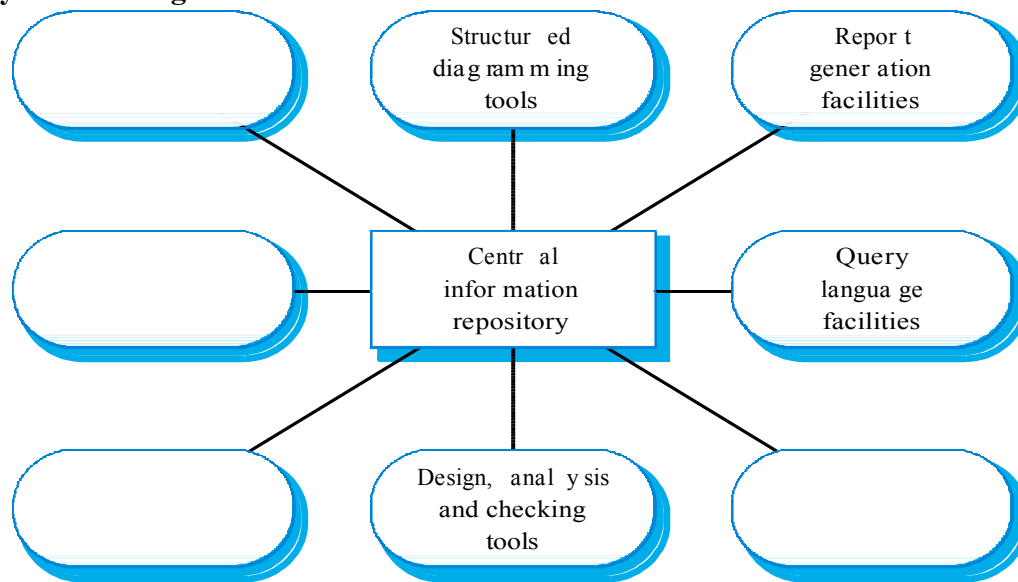
**Method weaknesses**

- They do not model non-functional system requirements.
- They do not usually include information about whether a method is appropriate for a given problem.
- They may produce too much documentation.
- The system models are sometimes too detailed and difficult for users to understand.

**CASE workbenches**

- A coherent set of tools that is designed to support related software process activities such as analysis, design or testing.
- Analysis and design workbenches support system modelling during both requirements engineering and system design.
- These workbenches may support a specific design method or may provide support for a creating several different types of system model.

### An analysis and design workbench



- Diagram editors
- Model analysis and checking tools
- Repository and associated query language
- Data dictionary
- Report definition and generation tools
- Forms definition tools
- Import/export translators
- Code generation tools

### Data Dictionary

- Data dictionaries are lists of all of the names used in the system models. Descriptions of the entities, relationships and attributes are also included
- Advantages
  - Support name management and avoid duplication
  - Store of organisational knowledge linking analysis, design and implementation
- Many CASE workbenches support data dictionaries

### Data dictionary entries

Name	Description	Type	Date
has-labels	1:N relation between entities of type Node or Link and entities of type Label.	Relation	5.10.1998
Label	Holds structured or unstructured information about nodes or links. Labels are represented by an icon (which can be a transparent box) and associated text.	Entity	8.12.1998
Link	A 1:1 relation between design entities represented as nodes. Links are typed and may be named.	Relation	8.12.1998
name (label)	Each label has a name which identifies the type of label. The name must be unique within the set of label types used in a design.	Attribute	8.12.1998
name (node)	Each node has a name which must be unique within a design. The name may be up to 64 characters long.	Attribute	15.11.1998

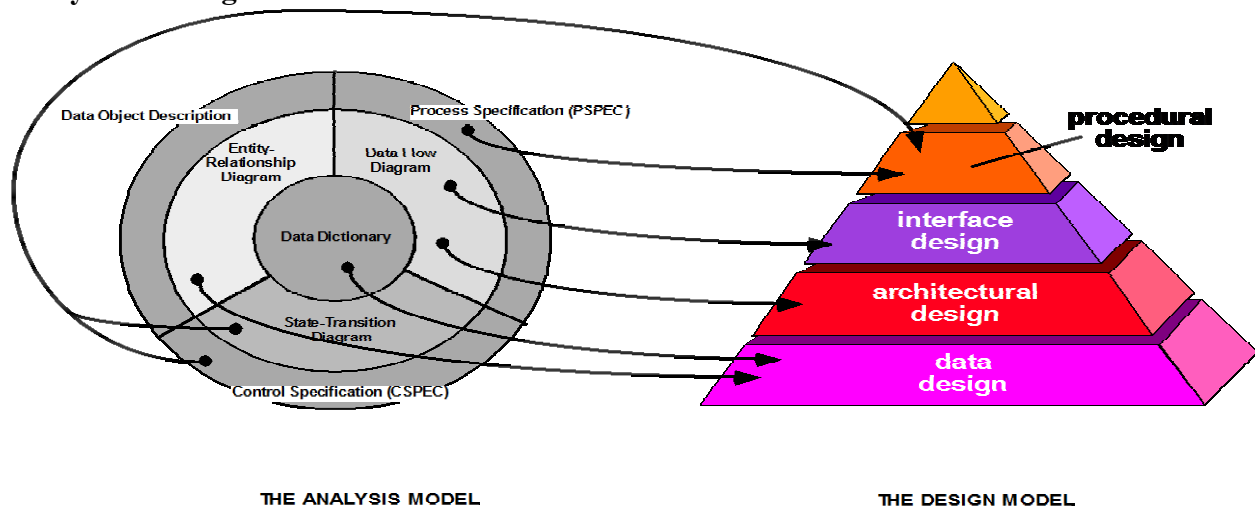
### UNIT III

#### ANALYSIS, DESIGN CONCEPTS AND PRINCIPLES

##### Design Concepts and Principles:

- Map the information from the analysis model to the design representations - data design, architectural design, interface design, procedural design

##### Analysis to Design:



##### Design Models – 1: