SOFTWARE REQUIREMENTS

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product.

Requirement Engineering

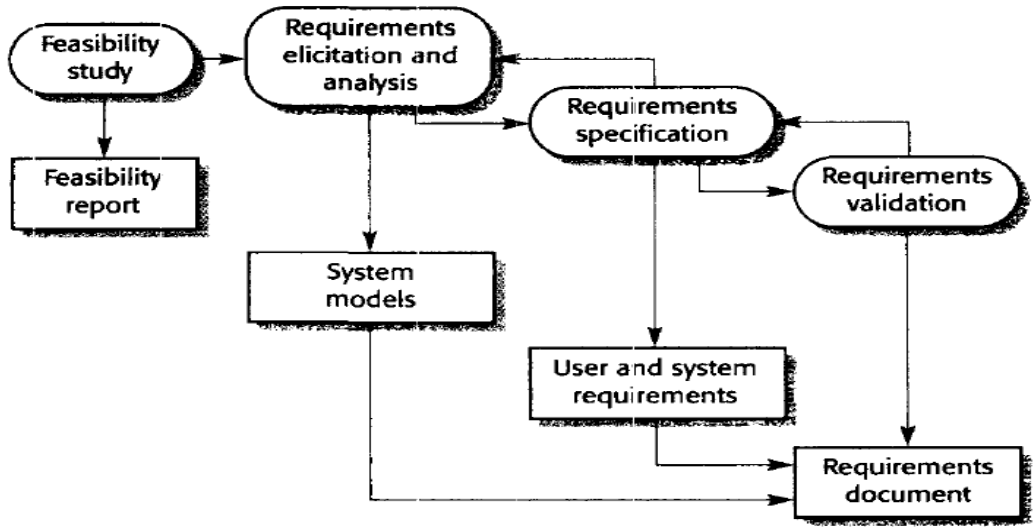
The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'Software Requirements Specification' document.

Requirement Engineering Process

It is a four step process, which includes –

- Feasibility Study
- Requirements elicitation and analysis
- Software Requirement Specification
- Software Requirement Validation



Feasibility studies

For all new systems, the requirements engineering process should start with a feasibility study. The input to the feasibility study is a set of preliminary business requirements, an outline description of the system and how the system is intended to support business processes. The results of the feasibility study should be a report that recommends whether or not it is worth carrying on with the requirements engineering and system development process.

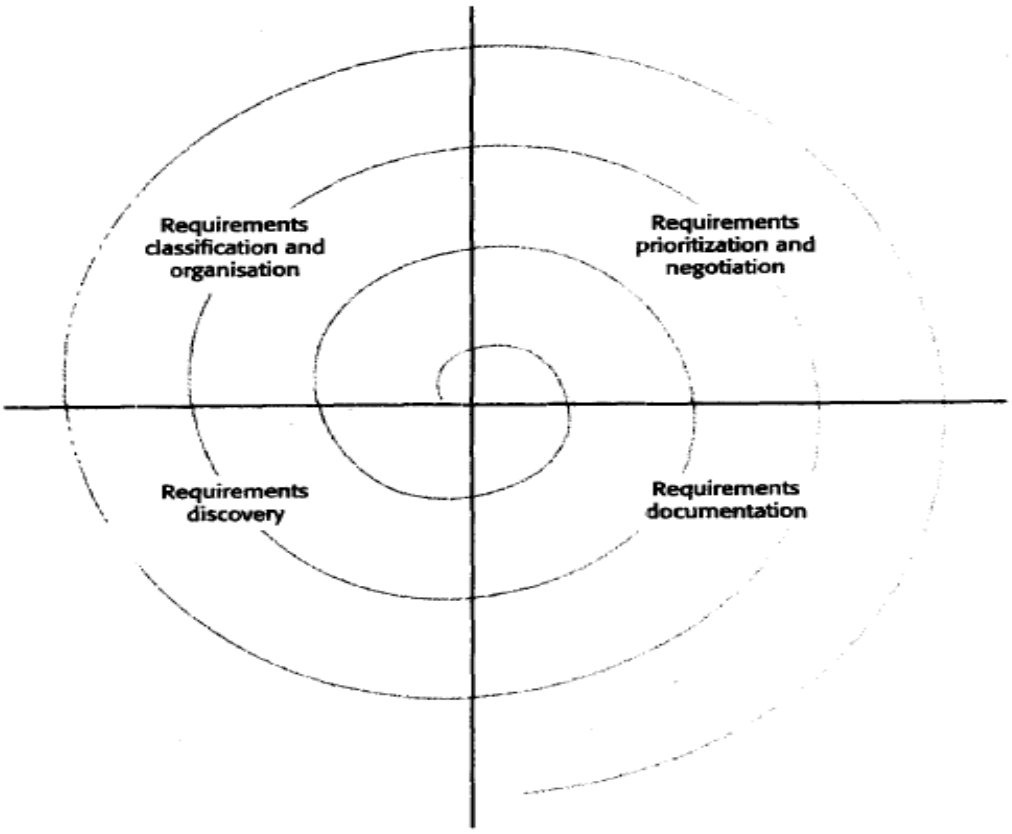
In a feasibility study, you may consult information sources such as the managers of the departments where the system will be used, software engineers who are familiar with the type of system that is proposed, technology experts and end-users of the system. Normally, you should try to complete a feasibility study in two or three weeks.

Requirements elicitation and analysis

The next stage of the requirements engineering process is requirements elicitation and analysis. In this activity, software engineers work with customers and system end-users to find out about the application domain, what services the system should provide, the required performance of the system, hardware constraints, and so on.

Requirements elicitation and analysis may involve a variety of people in an organization. The term stakeholder is used to refer to any person or group who will be affected by the system, directly or indirectly. Stakeholders include end-users who interact with the system and everyone else in an organisation that may be affected by its installation.

A very general process model of the elicitation and analysis process is shown in Figure:



The process activities are:

1. *Requirements discovery* This is the process of interacting with stakeholders in the system to collect their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity.
2. *Requirements classification and organisation* This activity takes the unstructured collection of requirements, groups related requirements and organises them into coherent clusters.
3. *Requirements prioritisation and negotiation* Inevitably, where multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritising requirements, and finding and resolving requirements conflicts through negotiation.
4. *Requirements documentation* The requirements are documented and input into the next round of the spiral. Formal or informal requirements documents may be produced.

Requirements validation

Requirements validation is concerned with showing that the requirements actually define the system that the customer wants. Requirements validation overlaps analysis in that it is concerned with finding problems with the requirements.
Requirements validation is important because errors in a requirements document can lead to extensive rework costs when they are discovered during development or after the system is in service. The cost of fixing a requirements problem by making a system change is much greater than repairing design or coding errors.

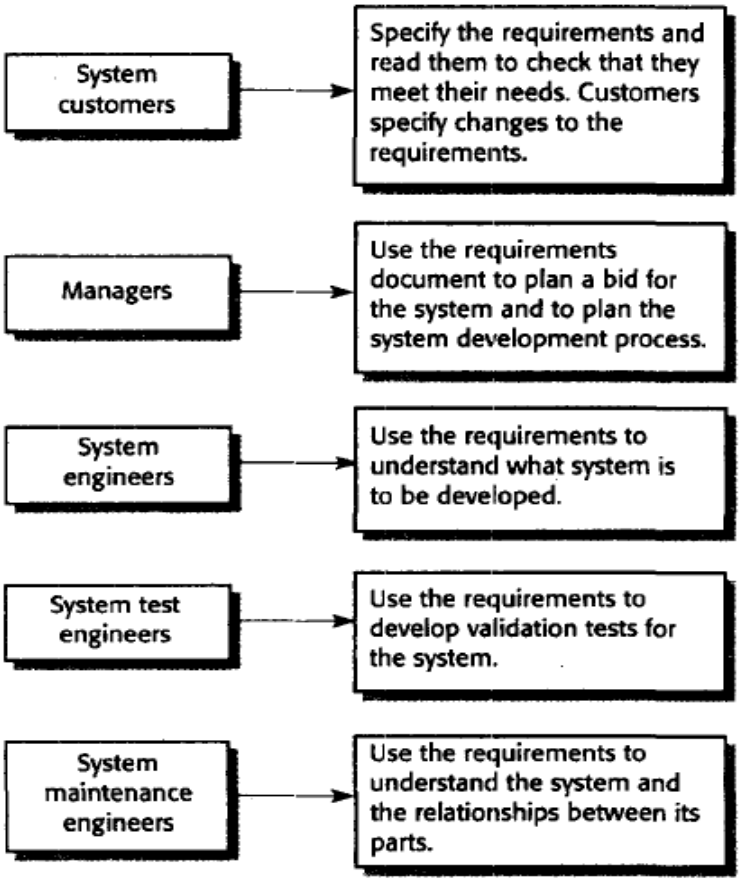
During the requirements validation process, checks should be carried out on the requirements in the requirements document. These checks include:

1. *Validity checks* A user may think that a system is needed to perform certain functions. However, further thought and analysis may identify additional or different functions that are required. Systems have diverse stakeholders with distinct needs, and any set of requirements is inevitably a compromise across the stakeholder community.
2. *Consistency checks* Requirements in the document should not conflict. That is, there should be no contradictory constraints or descriptions of the same system function.
3. *Completeness checks* The requirements document should include requirements, which define all functions, and constraints intended by the system user.
4. *Realism checks* Using knowledge of existing technology, the requirements should be checked to ensure that they could actually be implemented. These checks should also take account of the budget and schedule for the system development.
5. *Verifiability* To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable. This means that you should be able to write a set of tests that can demonstrate that the delivered system meets each specified requirement.

The software requirements document

The software requirements document (sometimes called the software requirements specification or SRS) is the official statement of what the system developers should implement. It should include both the user requirements for a system and a detailed specification of the system requirements.
The diversity of possible users means that the requirements document has to be a compromise between communicating the requirements to customers, defining the requirements in precise detail for developers and testers, and including information about possible system evolution. Information on anticipated changes can help system designers avoid restrictive design decisions and help

system maintenance engineers who have to adapt the system to new requirements.



The structure of a requirements document

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe its functions and explain how it will work with other systems. It should describe how the system fits into the overall business or strategic objectives of the organisation commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	The services provided for the user and the non-functional system requirements should be described in this section. This description may use natural language, diagrams or other notations that are understandable by customers. Product and process standards which must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.
System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements, e.g. interfaces to other systems may be defined.
System models	This should set out one or more system models showing the relationships between the system components and the system and its environment. These might be object models, data-flow models and semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based and anticipated changes due to hardware evolution, changing user needs, etc.
Appendices	These should provide detailed, specific information which is related to the application which is being developed. Examples of appendices that may be included are hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organisation of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, etc.

Requirements specification

- The process of writing donw the user and system requirements in a requirements document.
- User requirements have to be understandable by end-users and customers who do not have a technical background.
- System requirements are more detailed requirements and may include more technical information.
- The requirements may be part of a contract for the system development
  - It is therefore important that these are as complete as possible.

Ways of writing system requirements specification:

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

Functional requirements

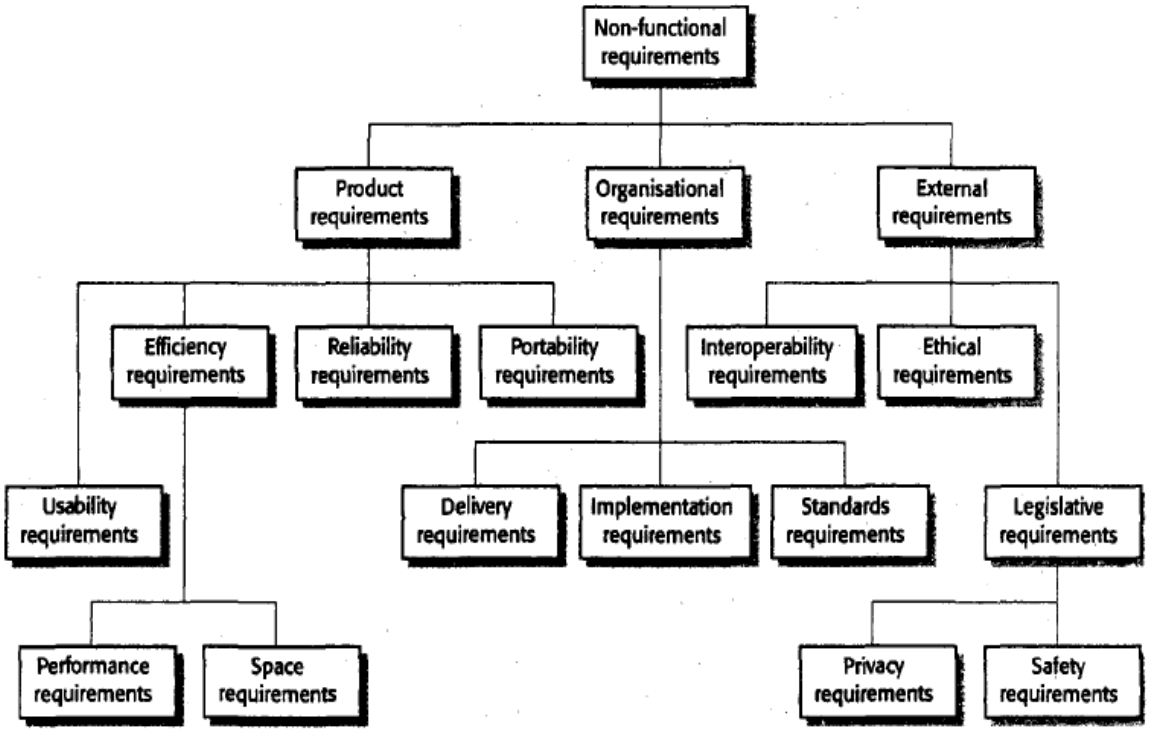
The functional requirements for a system describe what the system should do. These requirements depend on the type of software being developed, the expected users of the software and the general approach taken by the organisation when writing requirements. When expressed as user requirements, the requirements are usually described in a fairly abstract way. However, functional system requirements describe the system function in detail, its inputs and outputs, exceptions, and so on.

In principle, the functional requirements specification of a system should be both complete and consistent. *Completeness* means that all services required by the user should be defined. *Consistency* means that requirements should not have contradictory definitions. In practice, for large, complex systems, it is practically impossible to achieve requirements consistency and completeness.

Non-functional requirements

Non-functional requirements, as the name suggests, are requirements that are not directly concerned with the specific functions delivered by the system. They may relate to emergent system properties such as reliability, response time and store occupancy. Alternatively, they may define constraints on the system such as the capabilities of I/O devices and the data representations used in system interfaces.

Types of non-functional requirements



1. *Product requirements* These requirements specify product behaviour. Examples include performance requirements on how fast the system must execute and how much memory it requires; reliability requirements that set out the acceptable failure rate; portability requirements; and usability requirements.

2. *Organisational requirements* These requirements are derived from policies and procedures in the customer s and developer s organisation. Examples include process standards that must be used; implementation requirements such as the programming language or design method used; and delivery requirements that specify when the product and its documentation are to be delivered.

3. *External requirements* This broad heading covers all requirements that are derived from factors external to the system and its development process. These may include interoperability requirements that define how the system interacts with systems in other organisations; legislative requirements that must be followed to ensure that the system operates within the law; and ethical requirements. Ethical requirements are requirements placed on a system to ensure that it will be acceptable to its users and the general public.

Ethnography

Ethnography is an observational technique that can be used to understand social and organisational requirements. An analyst immerses him or herself in the working environment where the system will be used. He: or she observes the day-to-day work and notes made of the actual tasks in which participants are involved. The value of ethnography is that it helps analysts discover implicit system requirements that reflect the actual rather than the formal processes in which people are involved.

Scope of ethnography

- Requirements that are derived from the way that people actually work rather than the way I which process definitions suggest that they ought to work.
- Requirements that are derived from cooperation and awareness of other people's activities.
  - Awareness of what other people are doing leads to changes in the ways in which we do things.
- Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.

Requirements management

The requirements for large software systems are always changing. One reason for this is that these systems are usually developed to address 'wicked' problems Because the problem cannot be fully defined, the software requirements are bound to be incomplete.. During the software process, the stakeholders' understanding of the problem is constantly changing. These requirements must then evolve to reflect this changed problem view.

Requirements management is the process of understanding and controlling changes to system requirements. You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements. The process of requirements

management should start as soon as a draft version of the requirements document is available, but you should start planning how to manage changing requirements during the requirements elicitation process.

1. *Enduring requirements* These are relatively stable requirements that derive from the core activity of the organisation and which relate directly to the domain of the system. For example, in a hospital, there will always be requirements concerned with patients, doctors, nurses and treatments. These requirements may be derived from domain models that show the entities and relations that characterise an application domain (Easterbrook, 1993; Prieto-Díaz and Arango, 1991).
2. *Volatile requirements* These are requirements that are likely to change during the system development process or after the system has been become operational. An example would be requirements resulting from government healthcare policies.

Requirements management planning

Planning is an essential first stage in the requirements management process. Requirements management is very expensive. For each project, the planning stage establishes the level of requirements management detail that is required. During the requirements management stage, you have to decide on:

1. *Requirements identification* Each requirement must be uniquely identified so that it can be cross-referenced by other requirements and so that it may be used in traceability assessments.
2. *A change management process* This is the set of activities that assess the impact and cost of changes. I discuss this process in more detail in the following section.
3. *Traceability policies* These policies define the relationships between requirements, and between the requirements and the system design that should be recorded and how these records should be maintained.
4. *CASE tool support* Requirements management involves the processing of large amounts of information about the requirements. Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

Requirements change management

✧ Deciding if a requirements change should be accepted

- *Problem analysis and change specification*
  - During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.
- *Change analysis and costing*
  - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
- Change implementation
  - The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented.

System models

System modeling helps the analyst to understand the functionality of the system and models are used to communicate with customers

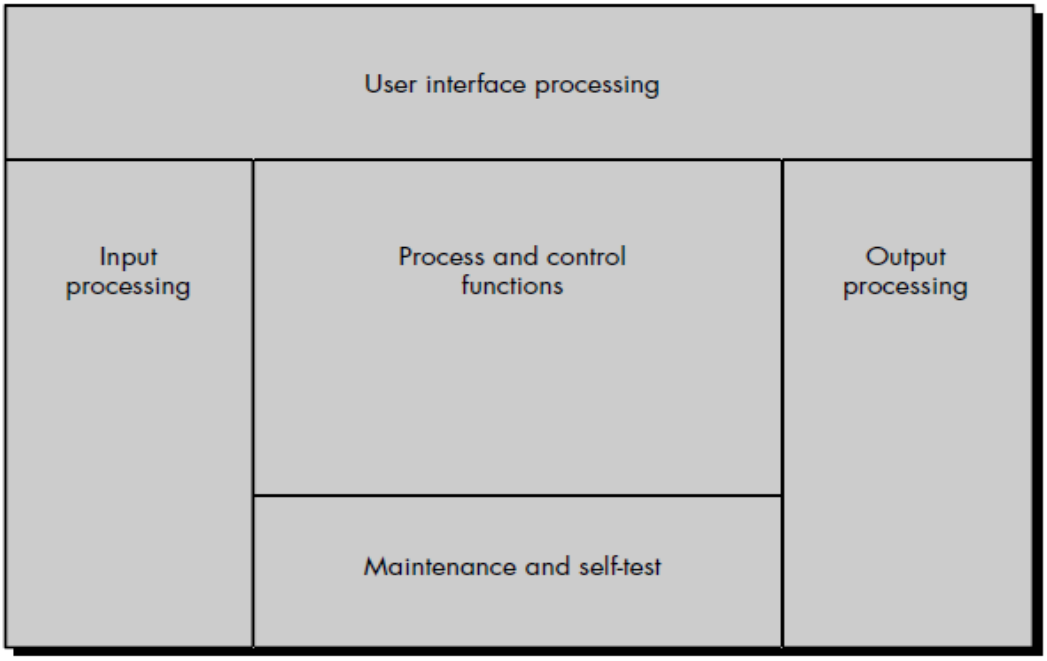
. • Different models present the system from different perspectives

– External perspective showing the system's context or environment;

– Behavioural perspective showing the behavior of the system;

– Structural perspective showing the system or data architecture.

Using a representation of input, processing, output, user interface processing, and self-test processing, a system engineer can create a model of system components that sets a foundation for later steps in each of the engineering disciplines.
To develop the system model, a *system model template* is used. The system engineer allocates system elements to each of five processing regions within the template: (1) user interface, (2) input, (3) system function and control, (4) output, and (5) maintenance and self-test.

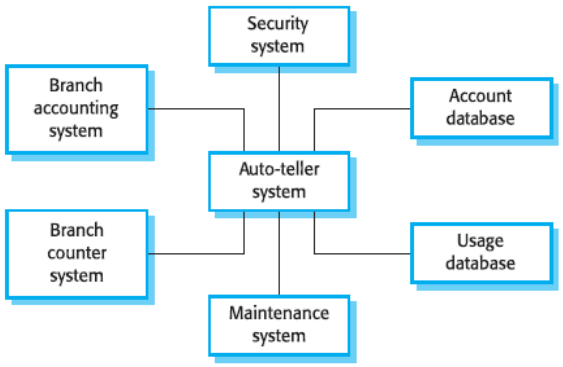


Model types

• Data processing model showing how the data is processed at different stages.
• Composition model showing how entities are composed of other entities.
• Architectural model showing principal sub-systems.
• Classification model showing how entities have common characteristics.
• Stimulus/response model showing the system's reaction to events.

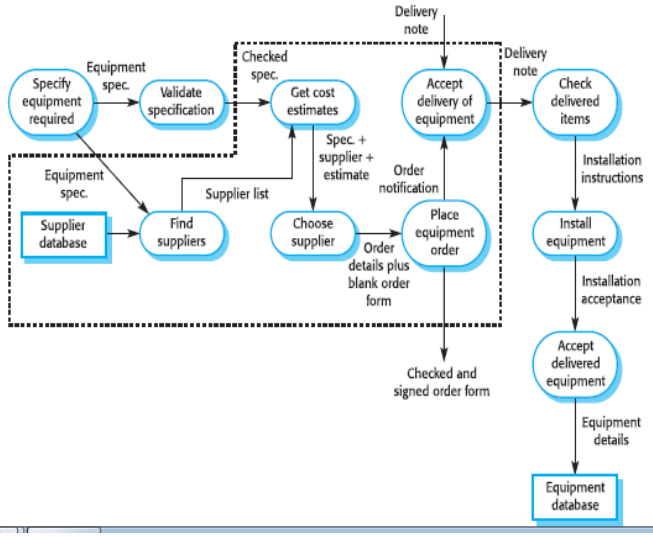
Context models

- Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- Social and organizational concerns may affect the decision on where to position system boundaries.
- Architectural models show the system and its relationship with other systems.



Process models

- Process models show the overall process and the processes that are supported by the system.
- Data flow models may be used to show the processes and the flow of information from one process to another.

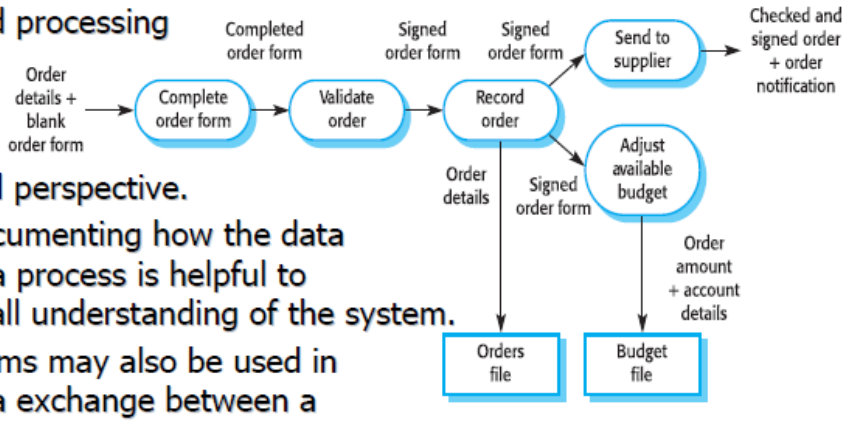


Behavioral models

- Behavioral models are used to describe the overall behavior of a system.
- Two types of behavioral model are:
  - 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.
- These models show different perspectives so both of them are required to describe the system's behavior.

Data-processing models

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

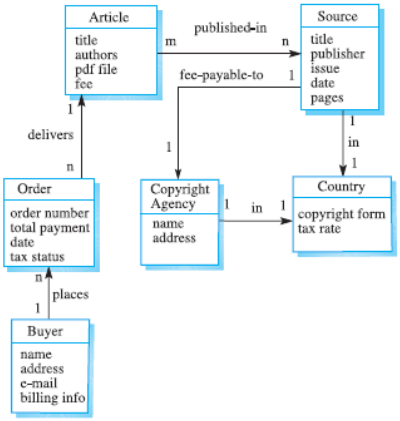


State machine models

- These model the behavior of the system in response to external and internal events.
- They show the system's responses to stimuli so are often used for modeling 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
  - An integral part of the UML and are used to represent state machine models.
  - Allow the decomposition of a model into sub-models (see following slide).
  - 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.

Semantic data models

- Used to describe the logical structure of data processed by the system.
- An 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.



Stan Kurkovsky

Object models

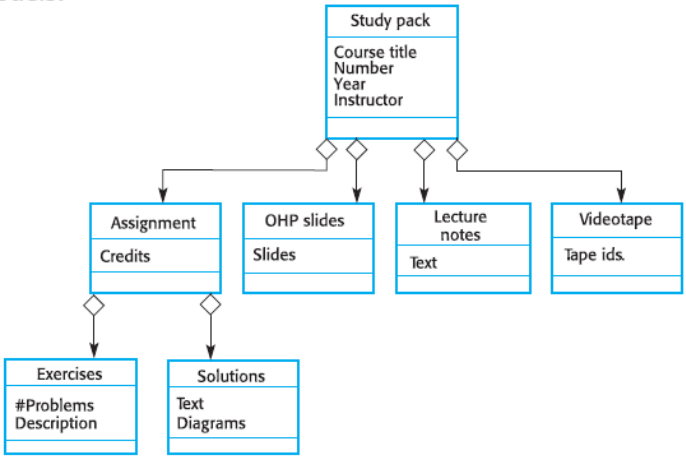
- Object models describe the system in terms of object classes and their associations.
- An object class is an abstraction over a set of objects with common attributes and the services (operations) provided by each object.
- Various object models may be produced
  - Inheritance models;
  - Aggregation models;
  - Interaction models.
- Natural ways of reflecting the real-world entities manipulated by the system
- More abstract entities are more difficult to model using this approach
- Object class identification is recognized as a difficult process requiring a deep understanding of the application domain
- Object classes reflecting domain entities are reusable across systems

Inheritance models

- Organize the domain object classes into a hierarchy.
- Classes at the top of the hierarchy reflect the common features of all classes.
- Object classes inherit their attributes and services from one or more super-classes. these may then be specialized as necessary.
- Class hierarchy design can be a difficult process if duplication in different branches is to be avoided.

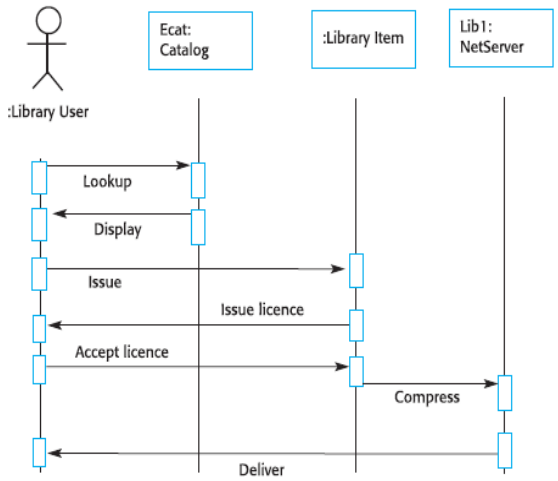
Object aggregation

- An aggregation model shows how classes that are collections are composed of other classes.
- Aggregation models are similar to the part-of relationship in semantic data models.



Object behavior modeling

- A behavioral model shows the interactions between objects to produce some particular system behavior that is specified as a use-case.
- Sequence diagrams (or collaboration diagrams) in the UML are used to model interaction between objects.



Structural models

- Structural models of software display the organization of a system in terms of the components that make up that system and their relationships.
- Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing.
- You create structural models of a system when you are discussing and designing the system architecture.

Interaction models

- Modeling user interaction is important as it helps to identify user requirements.
- Modeling system-to-system interaction highlights the communication problems that may arise.
- Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
- Use case diagrams and sequence diagrams may be used for interaction modeling.