Unit 2

Requirement Analysis

Requirement

1. The requirement for a system is the description of the services provided by the system and its operational constraints.
2. A requirement is simply a high level, abstract statement of a service that the system should provide or a constraint on the system.

Requirement Engineering

The process of finding out, analyzing, documenting and checking these services and constraints is called requirement engineering.

User Requirements

1. These are statements, in a natural language plus diagrams, of what services the system is expected to provide and the constraints under which it must operate.
2. The user requirements for a system should describe the functional and non-functional requirements so that they are understandable by system users without detailed technical knowledge.
3. They should only specify the external behavior of the system and should avoid, as for as possible, system design characteristics.
4. The readers of the user requirements are not usually concerned with how the system will be implemented and may be managers who are not interested in the detailed facilities of the system.

For example

"LIBSYS shall keep track of all data required by copyright licensing agencies in the UK and elsewhere."

However, various problems can arise when requirements are written in natural language sentences in a text document.

a) **Lack of clarity** – It is sometimes difficult to use language in a precise and unambiguous way without making the document wordy and difficult to read.
b) **Requirement Confusion** – Functional requirements, non-functional requirements, system goals and design information may be clearly distinguished.
c) **Requirements Amalgamation** – Several different requirements may be expressed together as a single requirement.

As an illustration of some of these problems, consider one of the requirements for the library

"LIBSYS" shall provide a financial accounting system that maintains records of all payments made by users of the system. System managers may configure this system so that regular users may receive discounted rates."

This requirement includes both conceptual and detailed information. It expresses the concept that there should be an accounting system as an inherent part of LIBSYS. However, it also includes the detail that the accounting system should support discounts for regular LIBSYS users. This detail would have been left to the system requirements specification.

To minimize misunderstandings when writing user requirements, you follow some simple guidelines:

1. Invent a standard format and ensure that all requirement definitions adhere to that format. Standardizing the format makes omissions less likely and requirements easier to check. You may also include information on who proposed the requirement (the requirement source) so that you know whom to consult if the requirement has to be changed.
2. Use language consistently. You should always distinguish between mandatory and desirable requirements. Mandatory requirements are requirements that the system must support and are usually written using 'shall'. Desirable requirements are not essential and are written using 'should'.
3. Use text highlighting (bold, italic or color) to pick out key parts of the requirement.
4. Avoid, as for as possible, the use of computer jargon. Inevitably, however, detailed technical terms will creep into the user requirements.

System Requirements

1. They set out the system's functions, services and operational constraints in detail.
2. System requirements are expanded versions of the user requirements that are used by software engineers as the starting point for the system design.
3. They add detail and explain how the user requirements should be provided by the system.
4. They may be used as part of the contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system.
5. The readers of the system requirements need to know more precisely what the system will do because they are concerned with how it will support the business processes or because they are involved in the system implementation.

For examples

- On making a request for a document from LIBSYS, the requester shall be presented with a form that records details of the user and the request made.
- LIBSYS request forms shall be stored on the system for five years form the date of the request.
- All LIBSYS request forms must be indexed by user, by the name of the material requested and by the supplier of the request.
- LIBSYS shall maintain a log of all requests that have been made to the system.
- For material where author's lending rights apply, loan details shall be sent monthly to copyright licensing agencies that have registered with LIBSYS.

Functional and Non-Functional Requirements

Software system requirements are often classified as functional, non-functional or domain requirements.

Functional Requirements

1. These are statement of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
2. In some cases, the functional requirements may also explicitly state what the system should not do.
3. These requirements depend on the type of software being developed, the expected users of the software and the general approach taken by the organization when writing requirements.

4. When expressed as user requirements, the requirements are usually described in a fairly abstract way.
5. Functional system requirements describe the system in detail, its inputs and outputs, exceptions, and so on.
6. 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.

For example, here are examples of functional requirements for a university library system called LIBSYS, used by the students and faculty to order books and documents from other libraries.

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

Non Functional Requirements

1. These are constraints on the services or functions offered by the system.
2. They are not directly concerned with the specific functions delivered by the system.
3. They include timing constraints, constraints on the development process and standards.
4. 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.
5. Non-functional requirements often apply to the system as a whole. They do not usually just apply to individual system features or services.
6. Failing to meet a non-functional requirement can mean that the whole system is unusable.
For e.g., if an aircraft system does not meet its reliability requirements, it will not be certified as safe for operation; if a real-time control system fails to meet its performance requirements, the control functions will not operate correctly.
7. Non-functional requirements arise through user needs, because of budget constraints, because of organizational policies, because of the need for interoperability with other software or hardware systems, or because of external factors such as safety regulations or privacy legislation.
8. The types of non-functional requirements

a) Product Requirements

- These requirements specify product behavior.
- 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.

For example

- The user interface for LIBSYS shall be implemented as simple HTML without frames or Java applets.

b) Organizational Requirements

- These requirements are derived from policies and procedures in the customer's and developer's organization.
- Examples include process standards that must be used.

- Implementation requirements such as the programming language or design method used.
- Delivery requirements that specify when the product and its documentation are to be delivered.

For example

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

c) External Requirements

- This broad heading covers all requirements that are derived from factors external to the system and its development process.
- They may include interoperability requirements that define how the system interacts with systems in other organizations.
- Legislative requirements that must be followed to ensure that the system operates within the law.
- Ethical requirements that the system which is developed will be acceptable to its users and the general public.

For example

- The system shall not disclose any personal information about system users apart from their name and library reference number to the library staff who use the system.

Domain Requirements

1. These requirements that come from the application domain of the system and that reflect characteristics and constraints of that domain.
2. They may be functional or non-functional requirements.
3. Domain requirements are derived from the application domain of the system rather than from the specific needs of the system users.
4. They usually include specialized domain terminology or reference to domain concepts.
5. They may be new functional requirements in their own right, constrain existing functional requirements or set out how particular computations must be carried out.
6. Because these requirements are specialized, software engineers often find it difficult to understand how they are related to other system requirements.
7. Domain requirements are important because they often reflect fundamental of the application domain.
8. If these requirements are not satisfied, it may be impossible to make the system work satisfactorily.

For example

The LIBSYS system includes a number of domain requirements:

a) There shall be a standard user interface to all databases that shall be based on the Z39.50 standard.

b) Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manual forwarding to the user or routed to a network printer.

The first requirement is a design constraint. It specifies that the user interface to the database must be implemented according to a specific library standard. The developers therefore have to find out about that standard before starting the interface design. The second requirement has been introduced because of copyright laws that apply to material used in libraries. It specifies that the system must include an automatic delete-on-print facility for some classes in document. This means that users of the library system cannot have their own electronic copy of the document.

Requirement Engineering Process

1. The goal of the requirement engineering process is to create and maintain a system requirement specification (SRS) document.
2. The overall process includes four high-level requirements engineering sub-processes.
3. These are concerned with
   a) Assessing whether the system is useful to the business i.e. **feasibility study**,
   b) Discovering requirements i.e. **requirement elicitation and analysis**,
   c) Converting these requirements into some standard form i.e. **requirements specification,** and
   d) Checking that the requirements actually define the system that the customer wants i.e. **requirement validation.**
4. Fig below illustrates the relationship between these activities. It also shows the documents produced at each stage of the requirements engineering process.

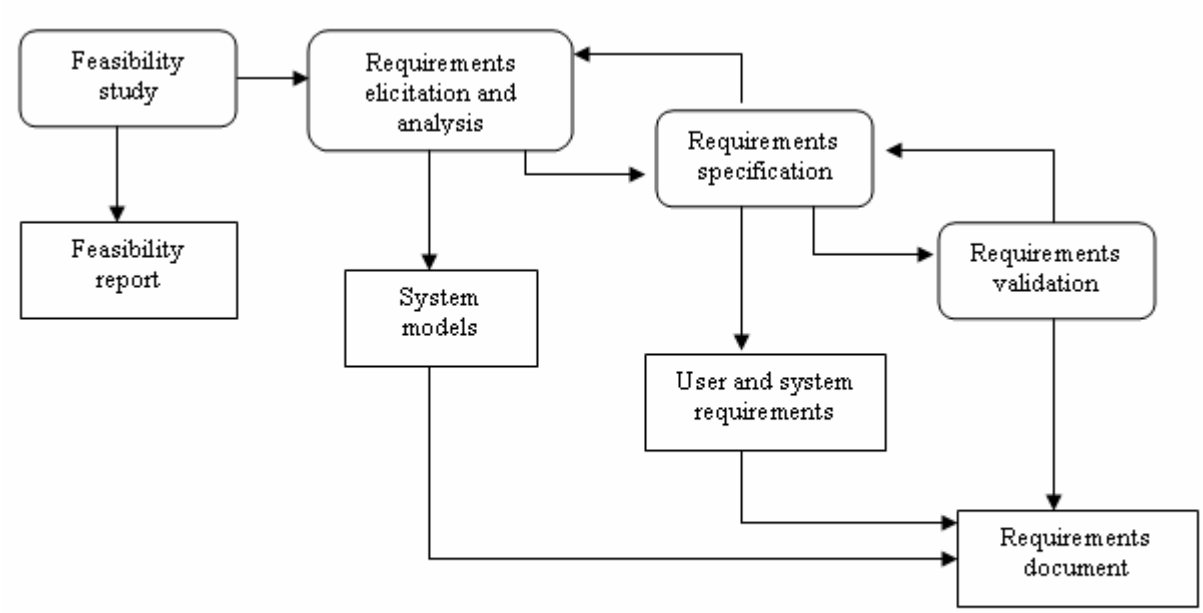


Fig Requirements engineering process

5. The activities shown in above fig are concerned with the discovery, documentation and checking of requirements. In virtually all systems, requirements change. The process involved develop a better understanding of what they want the software to do; the organization buying the system changes; modifications are made to the system's hardware, software and organizational environment. The process of managing these changing requirements is called **requirements management.**

Feasibility Study

1. For all new systems, the requirements engineering process should start with a feasibility study.
2. 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.
3. 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.
4. A feasibility study is a short, focused study that aims to answer a number of questions:
    a) Does the system contribute to the overall objectives of the organization?
    b) Can the system be implemented using current technology and within cost and schedule constraints?
    c) Can the system be integrated with other systems which are already in place?
5. The issue of whether or not the system contributes to business objectives is critical. If a system does not support these objectives, it has no real value to the business.
6. Carrying out a feasibility study involves information assessment, information collection and report writing.
7. The information assessment phase identifies the information that is required to answer these three questions. Once the information has been identified, you should talk with information sources to discover the answers to these questions. Some of the possible questions that may be put are:
    a) How would the organization cope if this system were not implemented?
    b) What are the problems with current processes and how would a new system help to solve these problems?
    c) What direct contribution will the system make to the business objectives and requirements?
    d) Can information be transferred to and from other organizational systems?
    e) Does the system require technology that has not previously been used in the organization?
    f) What must be supported by the system and what need not be supported?
8. 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 the system that is proposed, technology experts and end-users of the system.
9. Once you have the information, you write the feasibility study report. You should make a recommendation about whether or not the system development should continue. In the report, you may propose changes to the scope, budget and schedule of the system and suggest further high-level requirements for the system.

Requirements elicitation and analysis (Requirement Gathering)

1. 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.
2. 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 organization that may be affected by its installation. Other system stakeholders may be engineers who are developing or maintaining related systems, business managers, domain experts and trade union representatives.
3. Eliciting and understanding stakeholders requirements is difficult for several reasons:
    a) Stakeholders often don't know what they want from the computer system except in the most general terms. They may find it difficult to articulate what they want the system to do or make unrealistic demands because they are unaware of the cost of their requests.

b) Stakeholders naturally express requirements in their own terms and with implicit knowledge of their own work. Requirements engineers, without experience in the customer's domain, must understand these requirements.

c) Different stakeholders have different requirements, which they may express in different ways. Requirements engineers have to consider all potential sources of requirements and discover commonalities and conflict.

d) Political factors may influence the requirements of the system. For example, managers may demand specific system requirements that will increase their influence in the organization.

e) The economic and business environment in which the analysis takes place is dynamic. It inevitably changes during the analysis process. Hence the importance may emerge from new stakeholders who were not originally consulted.

4. A very general process model of the elicitation and analysis process is shown in fig below. Each organization will have its own version or instantiation of this general model, depending on local factors such as the expertise of the staff, the type of system being developed and the standards used.

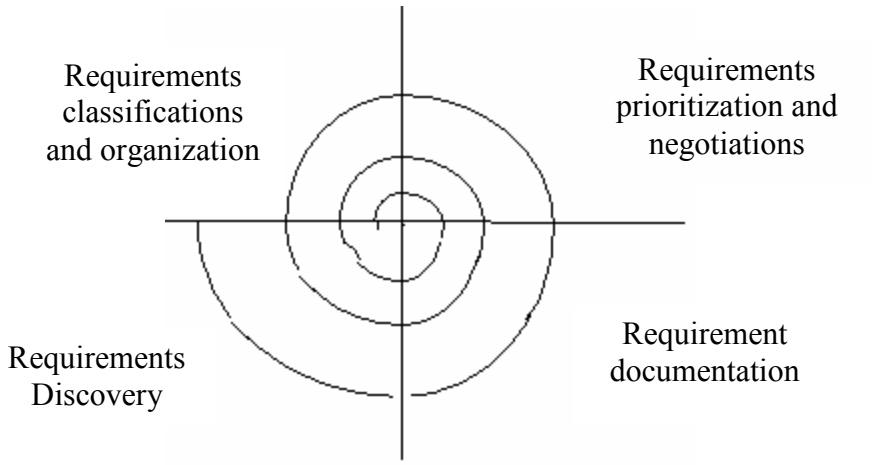
5. The process activities are:

a) **Requirement 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.

b) **Requirement classification and organization** – This activity takes the unstructured collection of requirements, group related requirements and organizes them into coherent clusters.

c) **Requirement prioritization and negotiation** – Inevitably, where multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirements, and finding and resolving requirements conflicts through negotiation.

d) **Requirements documentation** – The requirements are documented and input into the next round of the spiral. Formal and informal requirements documents may be produced.



6. The above fig shows that requirements elicitation and analysis is an iterative process with continual feedback from each activity to other activities. The process cycle starts with requirements discovery and ends with requirements documentation. The analyst's understanding of the requirements improves with each round of the cycle.

Requirements Validation

1. Requirements validation is concerned with showing that the requirements actually define the system that the customer wants.
2. Requirements validation overlaps analysis in that it is concerned with finding problems with the requirements.

3. 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.

4. The cost of fixing a requirements problem by making a system change is much greater than repairing design or coding errors. The reason for this is that a change to the requirements usually means that the system design and implementation must also be changed and then the system must be tested again.

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

   a) **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.
   b) **Consistency checks** – Requirements in the document should not conflict. That us, there should be no contradictory constraints or descriptions of the same system function.
   c) **Completeness checks** – The requirements document should include requirements, which define all functions, and constraints intended by the system user.
   d) **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.
   e) **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 are demonstrate that the delivered system meets each specified requirement.

6. A number of requirements validation techniques can be used in conjunction or individually:
   a) **Requirements reviews** – The requirements are concerned systematically by a team of reviewers.
   b) **Prototyping** – In this approach to validation, an executable model of the system is demonstrated to end-users and customers. They can experiment with this model to see if it meets their real needs.
   c) **Test-case generation** – Requirements should be testable. If the tests for the requirements are devised as part of the validation process, this often reveals requirements problems. If the test is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should be reconsidered. Developing tests from the user requirements before any code is written is an integral part of extreme programming.

Requirements Management

1. 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 stake holder's understanding of the problem is constantly changing. These requirements must then evolve to reflect this changed problem view.

2. Furthermore, once a system has been installed, new requirements inevitably emerge. It is hard for users and system customers to anticipate what effects the new system will have on the organization. Once end-users have experience of a system, they discover new needs and priorities:

   a) Large systems usually have a diverse user community where users have different requirements and priorities. These may be conflicting or contradictory. The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed.

   b) The people who pay for a system and the users of a system are rarely the same people. System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery; new features may have to be added for user support if the system is to meet its goals.

c) The business and technical environment of the system changes after installation, and these changes must be reflected in the system. New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change with consequent changes in the system support, and new legislation and regulations may be introduced which must be implemented by the system.

3. Requirements management is the process of understanding and controlling changes to system requirements.
4. You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes.
5. You need to establish a formal process for making change proposals and linking these to system requirements.
6. The process of requirements management should start as soon as a draft version of the requirements document is available, but you start planning how to manage changing requirements during the requirements elicitation process.

Software Prototyping

1. A prototype is an initial version of the software system which is used to demonstrate concepts, try out design options and, generally, to find out more about the problem and its possible solutions.
2. Rapid development of the prototype is essential so that costs are controlled and users can experiment with the prototype early in the software process.
3. A software prototype supports two requirements engineering process activities:

a) **Requirements Elicitation** – System prototypes allow users to experiment to see how the system supports their work. They get new ideas for requirements and can find areas of strength and weakness in the software. They may then propose new system requirements.

b) **Requirement validation** – The prototype may reveal errors and omissions in the requirements which have been proposed. A function described in a specification may seem useful and well defined. However, when that function is used with others, users often find that their initial view was incorrect or incomplete. The system specification may then be modified to reflect their changed understanding of the requirements.

4. Prototyping can be used as a risk analysis and reduction technique. A significant risk in software development is requirements errors and omissions. The costs of fixing requirements errors at later stages in the process can be very high. Experiments have shown that prototyping reduces the number of problems with the requirement specification. Furthermore, the overall development costs may be lower if a prototype is developed.
5. Prototyping is therefore part of the requirements engineering process.
6. Developing a prototype usually leads to improvements in the specification of the system. Once a prototype is available, it can also be used for other purposes:
   a) User training – A prototype system can be used for training users before the final system has been delivered.
   b) System testing – Prototypes can run ‘back-to-back’ tests. The same test cases are submitted to the prototype and to the system under test. If both systems give the same result, the test case has not detected a fault. If the results differ, it may mean that there is a system fault and the reasons for the difference should be investigated.

Benefits of Software Prototyping

1. Misunderstanding between software developers and users may be identified as the system functions are demonstrated.
2. Software development staff may find incomplete and/or inconsistent requirements as the prototype is developed.
3. A working, albeit limited, system is available quickly to demonstrate the feasibility and usefulness of the application to management.
4. The prototype may be used as a basis for writing the specification for a production-quality system.
5. Improved system usability.
6. A closer match of the system to the user needs.
7. Improved design quality.
8. Improved maintainability.
9. Reduced development effort.

Prototyping Development Process Model

A process model for prototype development is shown in fig below. The objectives of prototyping should be made explicit from the start of the process. These may be to develop a system to prototype the user interface, to develop a system to validate functional system requirements or to develop a system to demonstrate the feasibility of the application to management. The same prototype cannot meet all objectives. If objectives are left implicit, management or end-users may misunderstand the function of the prototype. Consequently, they may not get the benefits that they expected from the prototype development.

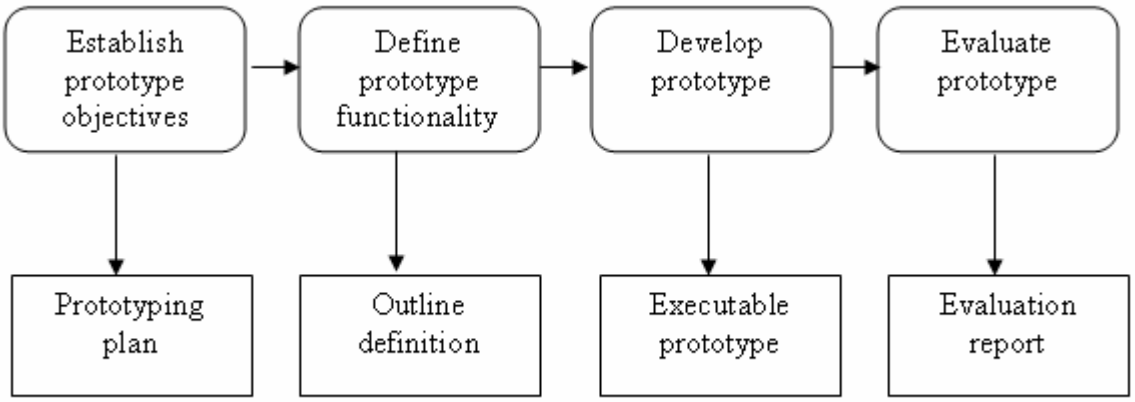


Fig The process of prototype development

The next stage in the process is to decide what to put into and, perhaps more importantly, what to leave out of the prototype system. To reduce prototyping costs and accelerate the delivery schedule, you may leave some functionality out of the prototype. You may decide to relax non-functional requirements such as response time and memory utilization. Error handling and management may be ignored or may be rudimentary unless the objective of the prototype is to establish a user interface. Standards of reliability and program quality may be reduced.

The final stage of the process is prototype evaluation. Provision must be made during this stage for user training and the prototype objectives should be used to derive a plan for evaluation. Users need time to become comfortable with a new system and to settle into a normal pattern of usage. Once they are using the system normally, they then discover requirement errors and omissions.

Prototyping in the software process

There are two approaches to the prototyping

a) Evolutionary prototyping

1. It is based on the idea of developing an initial implementation, exposing this to user comment and refining this through many stages until an adequate system has been developed.
2. It starts with a relatively simple system which implements the most important user requirements. This is augmented and changed as new requirements are discovered. Ultimately, it becomes the system which is required. There is no detailed system specification and, in many cases, there may not be a formal requirements document.
3. It is now the normal technique used for web-site development and e-commerce applications.
4. The **objective** of evolutionary prototyping is to deliver a working system to end-users. This means that you should normally start with the user requirements which are best understood and which have the highest priority. Lower priority and vaguer requirements are implemented when and if they are demanded by the users.
5. Evolutionary prototyping is part of or has much in common with techniques of Rapid application development (RAD) and Joint Application Development (JAD).

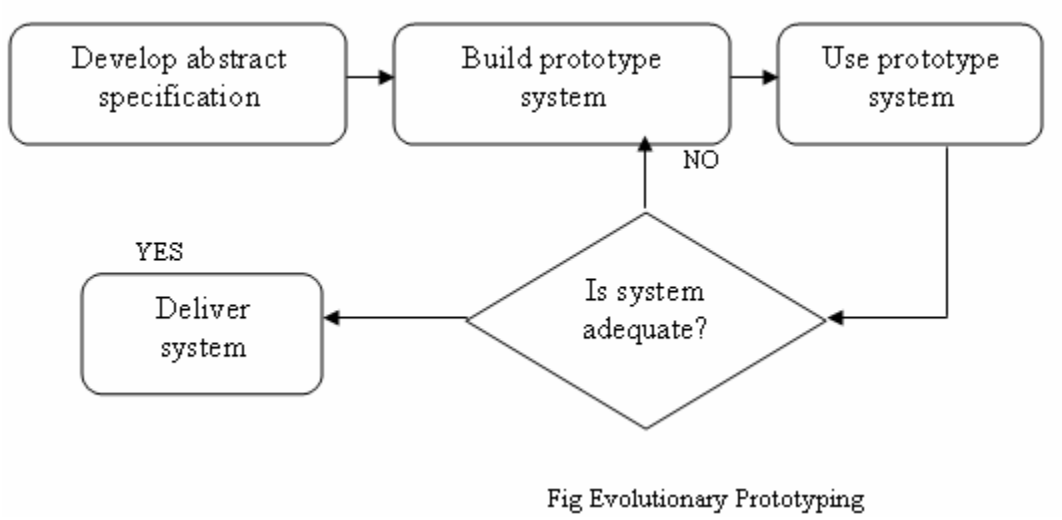


Fig Evolutionary Prototyping

Advantages of Evolutionary Prototyping

1. **Accelerated delivery of the system** – In evolutionary prototyping the delivery of the system is accelerated because the working system is made available in the early stages of the system.
2. **User engaged with the system –** The involvement of users with the development process does not just mean that the system is more likely to meet their requirements. It also means that the end-users of the system have made a commitment to it and are likely to want to make it work.

Problems with Evolutionary Prototyping

1. **Management problems –** Software management structures for large systems are set up to deal with a software process model that generates regular deliverables to assess progress. Prototypes evolve so quickly that it is not cost-effective to produce a great deal of system documentation. Furthermore, rapid prototype development may require unfamiliar technologies to be used. Managers may find it difficult to use existing staff because they lack these skills.

2. Maintenance problems – Continual change tends to corrupt the structure of the prototype system. This means that anyone apart from the original developers is likely to find it difficult to understand. Furthermore, if specialized technology is used to support rapid prototype development this may become obsolete. Therefore, finding people who have the required knowledge to maintain the system may be difficult.

3. Contractual problems – The normal contractual model between a customer and a software developer is based around a system specification. When there is no such specification, it may be difficult to design a contract for the system development. Customers may be unhappy with a contract which simply pays developers for the time spent on the project as this can lead to function creep and budget overruns; developers are unlikely to accept a fixed-price contract as they cannot control the changes requested by the end-users.

b) Throw-away prototyping

1. This approach extends the requirement analysis process with the intension of reducing overall life-cycle costs.
2. The principle function of the prototype is to clarify requirements and provide additional information for managers to assess process risks.
3. After evaluation, the prototype is thrown away. It is not used for further system development.
4. The **objective** of throw-away prototyping is to validate or derive the system requirements. You should start with those requirements that are not well understood because you need to find out more about them. Requirements that are straightforward may never need to be prototyped.
5. This approach to system prototyping is commonly used for hardware systems.
6. It is not normally used for design validation but to help develop the system requirements. The prototype design is often quite different from that of the final system.
7. The system must be developed as quickly as possible so that users can feed back their prototype experience to the development of the system specification.
8. Functionality may be stripped from the throw-away prototype where these functions are well understood, quality standards may be relaxed and performance criteria ignored.
9. The prototype development language will often be different from the final system implementation language.

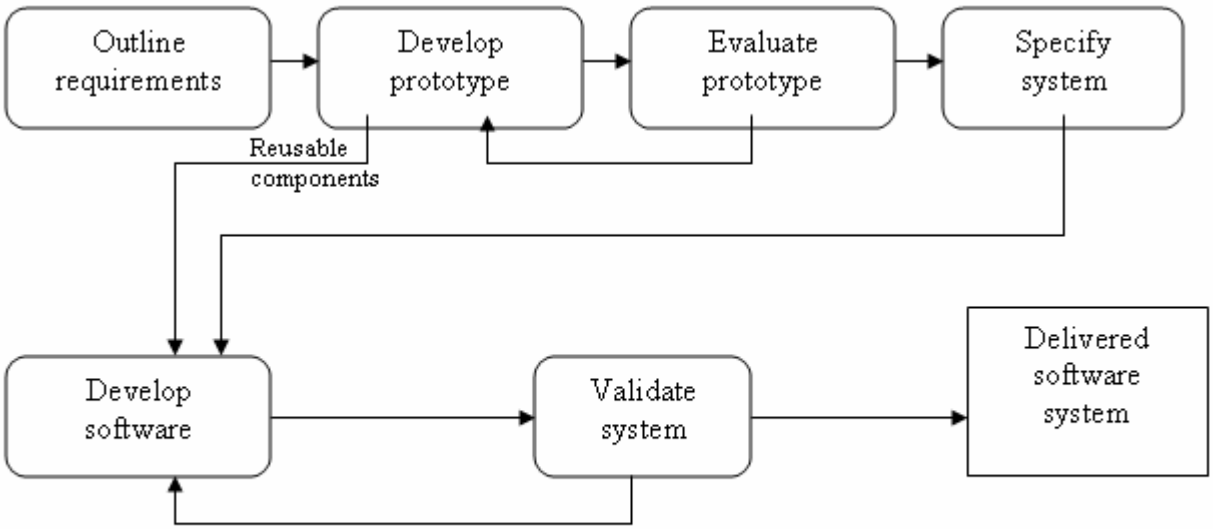


Fig. A software process with throw-away prototype

Difference between Evolutionary Prototype and Throw-away Prototype

No.	Throw-away Prototype	Evolutionary Prototype
1.	The prototype so developed is quick and dirty.	The prototype is not dirty.
2.	This type of prototype is thrown away.	This type of prototype is not thrown away but is evolved further.
3.	Here, we built only difficult parts.	Here, we build the understood parts first on sound foundations.
4.	It involves just one task.	It involves successive steps.
5.	Lesser probability of meeting user's needs.	Higher probability of meeting user needs as we have already used version *i* before using and implementing version $(i+1)^{th}$.

Rapid Prototyping Techniques

Rapid prototyping techniques are development techniques which emphasize speed of delivery rather than other system characteristics such as performance, maintainability or reliability. There are three rapid development techniques that are practical for developing industrial-strength prototypes:

a) Dynamic high-level language development

1. Dynamic high-level languages are programming languages which include powerful run-time data management facilities.
2. These simplify program development because they reduce many problems of storage allocation and management.
3. The language system includes facilities which normally have to be built from more primitive constructs in languages like Ada or C.
4. Examples of very high-level languages are Lisp (based on list structures), Prolog (based on logic) and Smalltalk (based on objects).
5. Until relatively recently, very high-level languages were not widely used for large system development because they need a large run-time support system. This run-time support increases the storage needs and reduces the execution speeds of programs written in the language. However, the increasing power and reducing cost of computer hardware have made these factors less important.
6. Java provides many of the advantages of very high-level languages with the rigor and the opportunities for performance optimization offered by conventional third-generation languages. Many reusable Java components are available so it is clearly a very suitable language for evolutionary prototyping.
7. Table below shows the dynamic languages that are most commonly used for prototype development.

Language	Type	Application domain
Smalltalk	Object-oriented	Interactive Systems
Java	Object-oriented	Interactive Systems
Prolog	Logic	Symbolic processing
Lisp	List-based	Symbolic processing

8. When choosing a prototyping language, you must ask a number of questions:
   - **What is the application domain of the problem?** – As shown in table, different languages are best suited to different application domains. If you want to prototype applications which involve natural language processing, a language such as Lisp or Prolog is more suitable than Java or Smalltalk.
   - **What user interaction is required?** – Different languages provide different levels of support for user interaction. Some languages, such as Smalltalk and Java, are well integrated with web browsers while others, such as Prolog, are best suited to text-based interfaces.
   - **What support environment is provided with the language?** – Mature support environments with many tools and easy access to reusable components simplify the prototype development process.

b) Database programming

1. Evolutionary development is now a standard technique for implementing small and medium-sized applications in the business systems domain. The majority of business applications involve manipulating data from a database and producing outputs which involve organizing and formatting that data.
2. To support the development of these applications, all commercial database management systems now support database programming.
3. Database programming is carried out using a specialized language which embeds knowledge of the database and which includes operations that are geared to database manipulation.
4. The language's supporting environment provides tools to support user interface definition, numeric computation and report generation.
5. The term fourth-generation language (4GL) is used to refer to both the database programming language and its supporting environment.
6. Fourth-generation languages are successful because there is a great deal of commonality across data processing applications. In essence, these applications are concerned with updating a database and producing reports from the information in the database. Standard forms are used for input and output.
7. 4GL are geared towards producing interactive applications which rely on abstracting information from an organizational database, presenting it to end-users on their terminal or workstation and then updating the database with changes made by users.
8. The user interface usually consists of a set of standard forms or a spreadsheet.

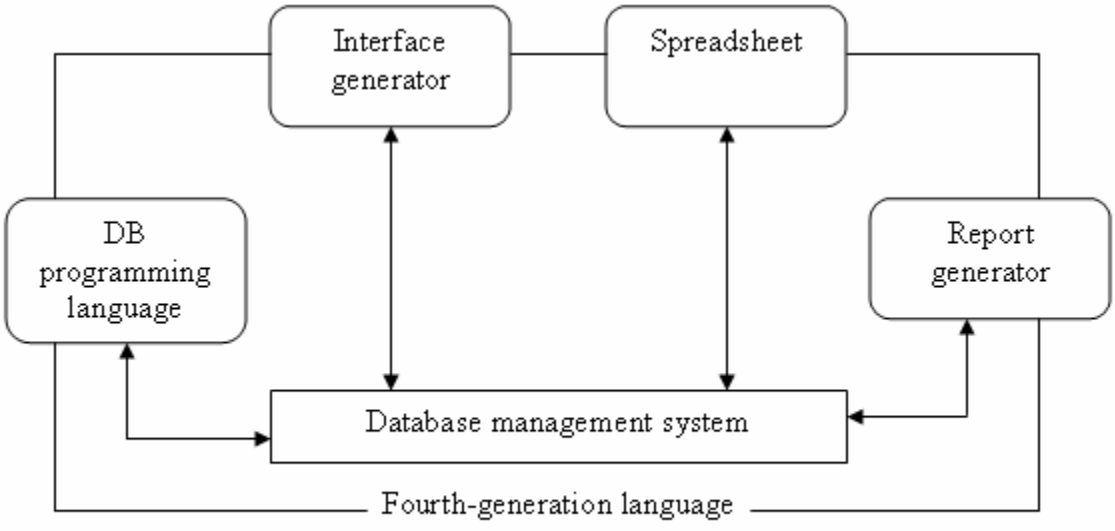


Fig Fourth-generation language components

9. The tools which are included in 4GL environment are:
   - A database query language which is now usually SQL. This may be input directly or generated automatically from forms filled in by an end-user.
   - An interface generator which is used to create forms for data input and display.
   - A spreadsheet for the analysis and manipulation of numeric information.
   - A report generator which is used to define and create reports from information in the database.

10. While 4GLs are very suitable for prototype development, there are some disadvantages in using them for production systems. Programs written in 4GL are usually slower than similar programs in conventional programming languages and usually require much more memory.

c) Component and application assembly

1. The time needed to develop a system can be reduced if many parts of that system can be reused rather than designed and implemented.
2. Prototypes can be constructed quickly if you have a set of reusable components and some mechanism to compose these components into systems.
3. The composing mechanism must include control facilities and a mechanism for component communications.
4. Prototyping with reusable components involves developing a system specification by taking account of what reusable components are available. This may mean that some requirements compromises may have to be made. The functionality of the available components may not be a precise fit for the user requirements. However, user requirements are often fairly flexible so, in most case, this approach can be used for prototype development.

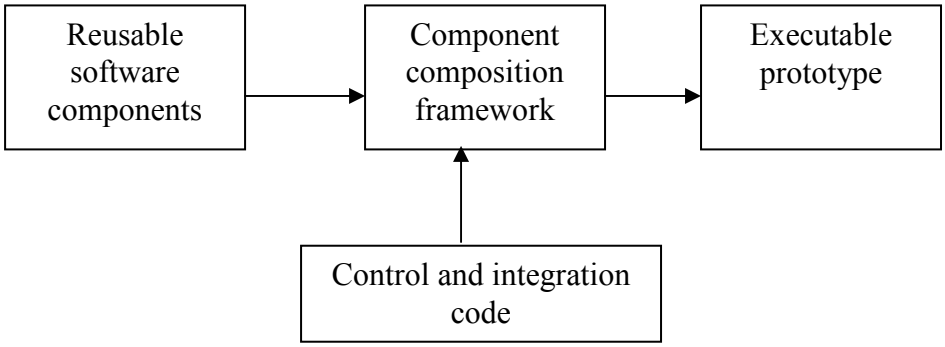


Fig Reusable component composition

5. Prototype development with reuse can be supported at two levels:
   a) The application level where entire application systems are integrated with the prototype so that their functionality can be shared. For example, if the prototype requires a text processing capability, this can be provided by integrating a standard word processing system. Application, such as Microsoft Office applications, support application linking.
   b) The component level where individual components are integrated within a standard framework to implement the system. The standard framework can be a scripting language which is designed for evolutionary development such as Visual Basic, TCL/TK, Python or Perl. Alternatively it can be more general component integration framework based on CORBA, DCOM or JavaBeans.

6. Visual development systems such as Visual Basic support this reuse-based approach to application development. Application programmers build the system interactively by defining the interface in terms of screens, fields, buttons and menus.

7. This approach to system development allows for the rapid development of relatively small and simple applications which can be built by one person or a small team of people. For large systems which must be developed by larger teams it is more difficult to organize.

User Interface Prototyping

1. Graphical user interfaces have now become the norm for interactive systems.
2. The effort involved in specifying, designing and implementing a user interface represents a significant part of application development costs.
3. The designers should not impose their view of an acceptable user interface on users. The users must take part in the interface design process. This realization led to an approach to design called user-centered design that depends on interface prototyping and user involvement throughout the interface design process.
4. From a software engineering point of view, prototyping is an essential part of the user interface design process. Because of the dynamic nature of user interfaces, textual descriptions and diagrams are not good enough for expressing the user interface requirements. Therefore, evolutionary prototyping with end-user involvement is the only sensible way to develop graphical user interfaces for software systems.
5. Interface generators are graphical screen design systems where interface components such as menus, fields, icons and buttons are selected from a menu and positioned on an interface.

Software Requirement Specification (SRS) or Software Requirement Plan (SRP): - It contains the following elements –

1. **Introduction**: - It states the goals and objectives of the software. It represents the scope of the software and the entire project.
2. **Information description**: - It provides a detailed description of the problem that the software has to solve. Here various diagrams are created for representing the relationships, data flow and control flow between various elements (example, DFD).
3. **Functional description**: - It contains the description related to problem solution attained through partitioning and the respective functions to implement the partitioning. It also contains a description of design constraints with justification, performance characteristics and a diagrammatic representation of the overall structure of the software and interrelations among various system elements.
4. **Behavioral description**: - It contains the description related to the internal operation of the software including its response to external events. It may contain various state transition diagrams.
5. **Bibliography**: - The bibliography contains references to various documents that are related to the software, engineering documents, technical references, vendor literature and standards.

Structure of a SRS

The general structure of an SRS proposed by IEEE standard is given below:

1. Introduction
    - 1.1 Purpose
    - 1.2 Scope
    - 1.3 Definitions, Acronyms, and Abbreviations
    - 1.4 References
    - 1.5 Overview
2. Overall Description
    - 2.1 Product Perspective
    - 2.2 Product Functions
    - 2.3 User Characteristics
    - 2.4 General Constraints
    - 2.5 Assumptions and Dependencies
3. Specific Requirements
    - 3.1 External Interface Requirements
        - 3.1.1 User Interfaces
        - 3.1.2 Hardware Interfaces
        - 3.1.3 Software Interfaces
        - 3.1.4 Communication Interfaces
    - 3.2 Functional Requirements
        - 3.2.1 Mode 1
            - 3.2.1.1 Functional Requirement 1.1
            - 3.2.1.2 Functional Requirement 1.2
        - 3.2.2 Mode 2
            - 3.2.2.1 Functional Requirement 2.1
            - 3.2.2.2 Functional Requirement 2.2
    - 3.3 Performance Requirements
    - 3.4 Design Constraints
    - 3.5 Attributes
    - 3.6 Other Requirements

The description of various sections of SRS is:

1. **Introduction** – This section contains the purpose, scope, overview, etc. of the requirements document. It also contains the references cited in the document and any definitions that are used.
2. **Overall Description** – It describes the general factors that affect the product and its requirements. Specific requirements are not mentioned, but a general overview is presented to make the understanding of the specific requirements easier. Product perspective is essentially the relationship of the product to other products; defining if the product is independent or is a part of a larger product, and what the principle interfaces of the product are. A general abstract description of the functions to be performed by the product is given.

3. **Specific Requirements** – It describes all the details that the software developer needs to know for designing and developing the system. This is typically the largest and most important part of the document. The external interface requirements section specifies all the interfaces of the software: to people, other software, hardware, and other systems. In the functional requirements section, the functional capabilities of the system are described. For each functional requirement, the required inputs, desired outputs, and processing requirements will have to be specified.

Benefits of a Great SRS (Need for SRS)

The IEEE 830 standard defines the benefits of a good SRS:

1. ***Establish the basis for agreement between the customers and the suppliers on what the software product is to do.*** The complete description of the functions to be performed by the software specified in the SRS will assist the potential users to determine if the software specified meets their needs or how the software must be modified to meet their needs. [NOTE: We use it as the basis of our contract with our clients all the time].
2. ***Reduce the development effort.*** The preparation of the SRS forces the various concerned groups in the customer's organization to consider rigorously all of the requirements before design begins and reduces later redesign, recoding, and retesting. Careful review of the requirements in the SRS can reveal omissions, misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct.
3. ***Provide a basis for estimating costs and schedules.*** The description of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates. [NOTE: Again, we use the SRS as the basis for our fixed price estimates]
4. ***Provide a baseline for validation and verification.*** Organizations can develop their validation and Verification plans much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured. [NOTE: We use the SRS to create the Test Plan].
5. ***Facilitate transfer.*** The SRS makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organization, and suppliers find it easier to transfer it to new customers.
6. ***Serve as a basis for enhancement.*** Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued production evaluation. [NOTE: This is often a major pitfall – when the SRS is not continually updated with changes]

Characteristics of a great SRS

An SRS should be

a) **Correct** – An SRS is correct if every requirements included in the SRS represents something required in the final system.

b) **Unambiguous** – An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. Requirements are often written in natural language, which are inherently ambiguous. If the requirements are specified in a natural language, the SRS write has to be especially careful to ensure that there are no ambiguities.

c) **Complete** – An SRS is complete if everything the software is supposed to do and the responses of the software to all classes of input data are specified in the SRS.

d) **Consistent** – An SRS is consistent if there is no requirement that conflicts with another. The SRS should be consistent within itself and consistent to its reference documents. If you call an input "Start and Stop" in one place, don't call it "Start/Stop" in another.

e) **Ranked for Importance** – Generally all the requirements are not of equal importance. Some are critical, others are important but not critical and there are some which are desirable but not very important. Some requirements are core requirements which are not likely to change as time passes, while others are more dependent on time. An SRS should be ranked for importance and / or stability. Stability of a requirement reflects the chances of it changing in future.

f) **Verifiable** – An SRS is verifiable if and only if every stated requirement is verifiable. A requirement is verifiable if there exists some cost-effective process that can check whether the final software meets that requirement. This implies that the requirement should have as little subjectivity as possible because subjective requirements are difficult to verify. Don't put in requirements like - "It should provide the user a fast response." Another of my favorites is - "The system should never crash." Instead, provide a quantitative requirement like: "Every key stroke should provide a user response within 100 milliseconds."

g) **Modifiable** – An SRS is modifiable if its structure and style should be such that any necessary changes can be made easily while preserving completeness and consistency. Presence of redundancy is a major obstacle to modifiability, as it can easily lead to errors. For example, assume that a requirement is stated in two places and that the requirement later needs to be changed. If only one occurrence of the requirements is modified, the resulting SRS will be inconsistent.

h) **Traceable** – An SRS is traceable if the origin of each or its requirements is clear and if it facilitates the referencing of each requirement in future development. Forward traceability means that each requirement should be traceable to some design and code elements. Backward traceability requires that it be possible to trace design and code elements to the requirements they support. Traceability aids verification and validation.

Analysis and Modeling

Models used in the Analysis phase of the development process

The modeling use during analysis phase should achieve three primary objectives –

1) To describe what the customer requires,

2) To establish a basis of software design and

3) To define a set of requirements that can be validated once the software is built.

To attain the above mentioned objectives the following structure for the analysis model is suitable:-

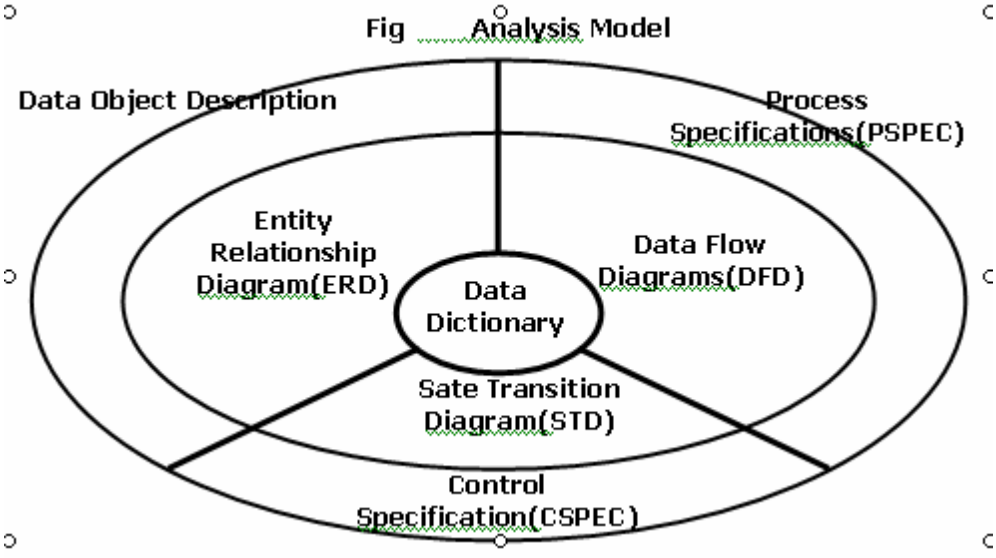


Fig Analysis Model

Entity Relationship Diagram (ERD)

The ERD is used to depict the relationship between the data objects. This notation is used in Data Modeling activity. The attributes of each object can be described using Data Object Description.

Data Modeling: - It identifies the following points related to data objects –

1) The primary data objects to be processed by the system,

2) The composition and attributes of each data object,

3) The current position and visibility of the data objects in the system,

4) The relationship between various data objects and

5) The processes that transform those data objects.

A basic data model consists of three interrelated elements –

1) Data objects,

2) Attributes and

3) Relationships.

1) **Data Objects**: - It's a representation of any composite information that must be understood by the software. Composite information means a collection of different properties or attributes. A data object can be an event (e.g. alarm), a role (e.g. salesperson), an organizational unit (e.g. EDP dept.), a place or a structure (e.g. file) etc. A data object encapsulated data only and there is no reference to the operations that can be performed on it.

2) **Attributes** :- It defines the properties of a data object and can be used for the following purposes – a) Naming an instance of the data object, b) Describe the instance, or c) Make reference to another instance in a different table. The problem context determines the attributes for a data object.

3) **Relationships**: - It's the relevant connection between the any numbers of data objects. While establishing a relationship it must be observed that the relationship is bi-directional.

Understanding Cardinality and Modality during Data modeling

Cardinality: - It's the specification of the number of occurrences of an object that can be related to a number of occurrences of another object. Cardinality is expressed in the terms "one" or "many". Following are the cardinality types:-

a) **One to One (1:1)**:- Here an occurrence of any object 'A' can relate to one and only one occurrence of object 'B' and vice-versa.

b) **One to many (1:N)**:- Here an occurrence of an object 'A' can relate to one or many occurrences of another object 'B', but an occurrence of 'B' can relate to only one occurrence of 'A'.

c) **Many to many (M:N)**:- Here an occurrence of an object 'A' can relate of one or more occurrences of another object 'B' and also an occurrence of 'B' can relate to one or more occurrences of 'A'.

The symbols on the relationship connection closest to the data object rectangles indicate cardinality. The vertical bar represents "One", and the three-pronged fork indicates "Many".

Modality: - Modality can be either expressed as 1 or 0. The modality is 1 if an occurrence of relationship is mandatory. The modality is 0 if there is no explicit need for the relationship to occur or the relationship is optional. Diagrammatically the symbols used further away from the object rectangles next to the cardinality symbols. Here the vertical bar represents the occurrence of mandatory relationship and a circle represents no explicit need for the relationship.

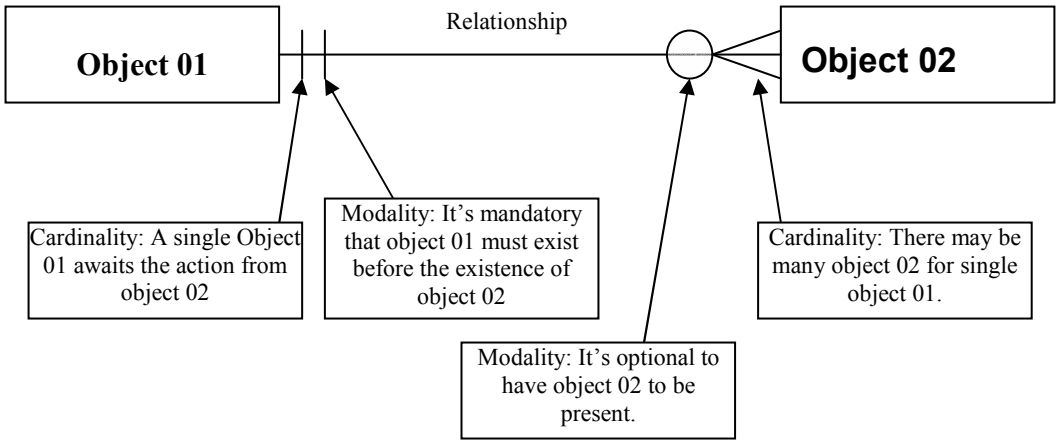


Fig. Entity Relationship Diagram

Guidelines for creating Entity Relationship Diagrams (ERD)

Following are some basic components of an ERD: - a) Data objects, b) Attributes, c) Relationships and d) Various types of indicators. Data objects are represented using rectangles. Relationships are represented using labeled lines, some ERD have a diamond shaped box for representing the relationship. Cardinality and Modality are represented using the above mentioned symbols. The entity relationship diagrams enables a software engineer to fully specify the data objects that go as input into the system and come as out put from a system. It also helps in specifying the properties of the input/output objects and their inter relationships. ERD follow an iterative process of construction and include the following steps:-

1) A list of input and output data objects as well as the entities that consume or produce information is gathered from the customer and also from analyzing what the customer means (knowingly or unknowingly).
2) Taking one object at a time, the customer and the analyst decide whether there exists a connection between the object under scrutiny and other objects.
3) Whenever a connection exists, the analyst and customer create one or more object-relationship pairs.
4) For each object-relationship pair, the respective cardinality and modality are explored.
5) Steps 2 through 4 are continued iteratively until all object-relationship pairs have been defined. During iterations there may be additions and omissions of the object-relationship pairs.
6) After the desired numbers of iterations are completed, the attributes of each entity are defined.
7) The resulting entity-relationship diagram is formalized and reviewed.
8) Steps 1 through 7 are repeated until data modeling is complete.

Data Flow Diagrams (DFD) or Data Flow Graph or Bubble Chart

A DFD is graphical technique that depicts –

a) The information flow and

b) The transformation activities that are applied on the data, as it moves from input to output.

It's very useful during software requirements analysis. The DFDs have basic two levels of development, they are as follows –

1) A Level 0 DFD, also known as Fundamental System Model or Context Model, represents the entire system as a single bubble with incoming arrowed lines as the input data and outgoing arrowed lines as output.
2) A Level 1 DFD might contain 5 or 6 bubbles with interconnecting arrows. Each process represented here is the detailed view of the functions shown in the level 0 DFD.
    - Here the rectangular boxes are used to represent the external entities that may act as input or output outside the system.
    - Round Circles are used to represent any kind of transformation process inside the system.
    - Arrow headed lines are used to represent the data object and its direction of flow.

A Process Specification (PSPEC) can be used to specify the processing details implied by a bubble within a DFD. It also indicates the limitations and restrictions imposed on the process.

Following are some guidelines for developing a DFD:-

1) Level 0 DFD should depict the entire software system as a single bubble.
2) Primary input and output should be carefully noted.
3) For the next level of DFD the candidate processes, data objects and stores should be recognized distinctively.
4) All the arrows and bubbles should be labeled with meaningful names.
5) Information flow continuity should be maintained in all the levels.
6) One bubble should be refined at a time.

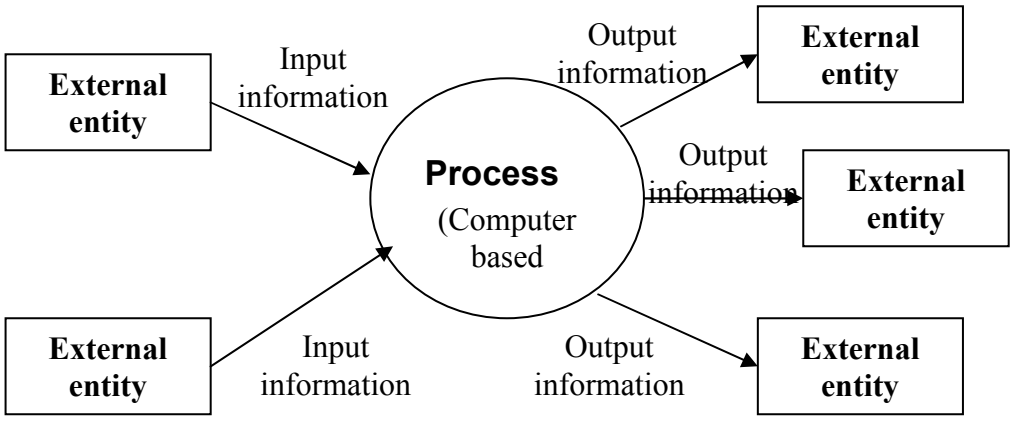


Fig Data Flow Diagram

The Control Specification (CSPEC):- The CSPEC represents the behavior of the system in two different ways –

1) State Transition Diagram (STD), which is a sequential specification of the behavior of the system in terms of state transitions between various processes in the system as bubbles and arrowed lines,

2) Process Activation Table (PAT), which is a table containing the explanation of the states in the STD, in terms of processes.

State Transition Diagrams

Transition tables: - They are used to specify changes in the state of a system as a function of the driving forces. The state of a system summarizes the status of all the entries in the system at a particular time. Given the current state and current conditions the next state is evaluated.

A simple transition table can be evaluated as f(Si, Cj) = Sk, where Si – is the present state, Cj – is the present condition , Sk – is the resultant state acquired by the system due to the present Si & Cj.

Current State	Current Input	
	A	b
State 0	Attained State 0	Attained State 1
State 1	Attained State 1	Attained State 0
	Next State	

From the above table we can derive a conclusion that when the system is in the present state, for ex. 'State 0' and the input given to the system is 'a' then the system would attain a state 'Attained State 0' and remain in it until there are some changes in the input.

1) **Finite State Mechanisms**: - Data flow diagrams, regular expressions and transition tables can be combined to provide a powerful finite state mechanism for functional specification of the software systems.

For example we assume the following transition table and create a transition diagram

Current State	Current Input	
	A	b
S0	S0	S1
S1	S1	S0
	Next State	

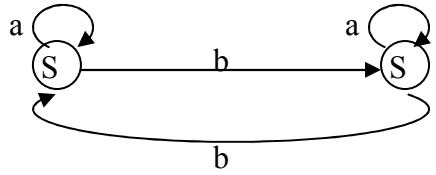


Fig State Transition Diagram

Data Dictionary: - The data dictionary is an organized collection of all data elements related to a system. It contains precise and complete definitions such that the user and the system analyst will have a common understanding of inputs, outputs, storage components and intermediate calculations. Some common information found in data dictionaries are as follows:-

1) **Name**: - It's the information by which an element in the system can be distinctively recognized.
2) **Alias**: - It's a duplicate name for any element.
3) **Where-used / how-used**: - It's the listing of the processes that use a particular element and the manner or mode in which the element is used.
4) **Content description**: - It contains some standard notations for representing the content of any element.
5) **Supplementary notations**: - It contains other information about data types, present value, restrictions, and limitations etc, related to elements of the system.

Advantages of Data Dictionary

1. **It is a mechanism for name management** – Many people may have to invent names for entities and relationships when developing a large system model. These names should be used consistently and should not clash. The data dictionary software can check for name uniqueness where necessary and warn requirements analysts of name duplications.
2. **It serves as a store of organizational information** – As the system is developed information that can link analysis, design, implementation and evolution is added to the data dictionary, so that all information about an entity is in one place.

Difference between DFD and ERD

No	DFD	ERD
1	It is a graphical technique that represents information flow and the transforms that applied as data move form input to output.	It is a graphical technique that represents data objects their relations and associations.
2	It provides a mechanism for functional modeling as well as information flow modeling.	It provides a mechanism for data modeling.
3	It does not contain the relationships among data.	It contains the relationships among various entities.
4	Inputs and outputs are represented separately.	Inputs and outputs are not represented separately.
5	Classes/objects cannot be clear from DFD i.e. Which data table should be kept together.	Classes/objects are clear from ERD i.e. which data table should be kept together.
6	It can give detailed functional overview of a system.	It cannot give detailed functional overview of a system.

Difference between DRD and Flow Chart

No	DFD	Flow Chart
1.	It is a graphical representation of data flow in the system.	It is a graphical representation for procedural design.
2.	It depicts the functions and sub-functions that transform data flow.	It cannot depict the sub-functions.
3.	It cannot depict conditions and loops.	It depicts conditions and loops.
4.	It provides a mechanism for functional modeling as well as information modeling.	It provides a mechanism for functional modeling.
5.	It may be used to represent a system or software at any level of abstraction.	It may be used to represent a system at only one level of abstraction.