Paper – II (Software Engineering)

**Paper - II (Software Engineering)**

**Unit – II**

**2 Requirements engineering processes**

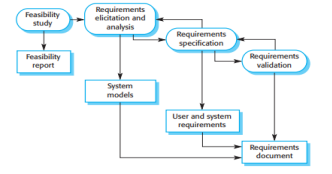
The goal of the requirements engineering process is to create and maintain a system requirements document. The overall process includes four high-level requirements engineering sub-processes. These are concerned with assessing whether the system is useful to the business (feasibility study); discovering requirements (elicitation and analysis); converting these requirements into some standard form (specification); and checking that the requirements actually define the system that the customer wants (validation). Figure 2.1 illustrates the relationship between these activities. It also shows the documents produced at each stage of the requirements engineering process. Specification and documentation are covered. The activities shown in Figure 2.1 are concerned with the discovery, documentation and checking of requirements. In virtually all systems, however, requirements change. The people 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, present an alternative perspective on the requirements engineering process in Figure 2.2. This presents the process as a three-stage activity where the activities are organized as an iterative process around a spiral. The amount of time and effort devoted to each activity in iteration depends on the stage of the overall process and the type of system being developed. Early in the process, most effort will be spent on understanding high-level business and non

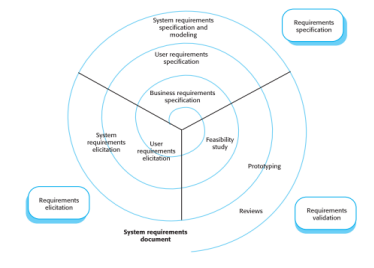
functional requirements and the user requirements. Later in the process, in the outer rings of the spiral, more effort will be devoted to system requirements engineering and system modeling. This spiral model accommodates approaches to development in which the requirements are developed to different levels of detail. The number of iterations around the spiral can vary, so the spiral can be exited after some or all of the user requirements have been elicited. If the prototyping activity shown under requirements validation is extended to include iterative development.

M.Sc. Semester – III (Computer Science) 1

Paper – II (Software Engineering)



**Fig. 2.1 The requirement Engineering Process**

**Fig. 2.2 Spiral model of requirements engineering processes**

This model allows the requirements and the system implementation to be developed together. Some people consider requirements engineering to be the process of applying a structured analysis method such as object-oriented analysis. This involves analysing the system and developing a set of graphical system models, such as use-case models, that then serve as a system specification. The set of models describes the behaviour of the system and are annotated with additional information describing, for example, its required performance or reliability. Although structured methods have a role to play in the requirements engineering process, there is

M.Sc. Semester – III (Computer Science) 2

Paper – II (Software Engineering)

much more to requirements engineering than is covered by these methods. Requirements elicitation, in particular, is a human-centred activity and people dislike the constraints imposed by rigid system models. Focus on general approaches to requirements engineering here and cover structured methods and system models.

**2.1 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. A feasibility study is a short, focused study that aims to answer a number of questions:

1. Does the system contribute to the overall objectives of the organisation? 2. Can the system be implemented using current technology and within given cost and schedule constraints?

3. Can the system be integrated with other systems which are already in place? 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. While this may seem obvious, many organisations develop systems which do not contribute to their objectives because they don’t have a clear statement of these objectives, because they fail to define the business requirements for the system or because other political or organisation factors influence the system procurement. Although this is not discussed explicitly, a feasibility study should be part of the Inception phase in the Rational Unified Process. Carrying out a feasibility study involves information assessment, information collection and report writing. The information assessment phase identifies the information that is required to answer the three questions set out above. Once the information has been identified, should talk with information sources to discover the answers to these questions. Some examples of possible questions that may be put are:

1. How would the organisation cope if this system were not implemented? 2. What are the problems with current processes and how would a new system help alleviate these problems?

3. What direct contribution will the system make to the business objectives and requirements?

M.Sc. Semester – III (Computer Science) 3

Paper – II (Software Engineering)

4. Can information be transferred to and from other organisational systems? 5. Does the system require technology that has not previously been used in the organisation? 6. What must be supported by the system and what need not be supported?

In a feasibility study, 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, try to complete a feasibility study in two or three weeks. Once have the information, write the feasibility study report. You should make a recommendation about whether or not the system development should continue. In the report, may propose changes to the scope, budget and schedule of the system and suggest further high-level requirements for the system.

**2.2 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 organisation. 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. Other system stakeholders may be engineers who are developing or maintaining related systems, business managers, domain experts and trade union representatives. Eliciting and understanding stakeholder requirements is difficult for several reasons:

1. 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. 2. 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.

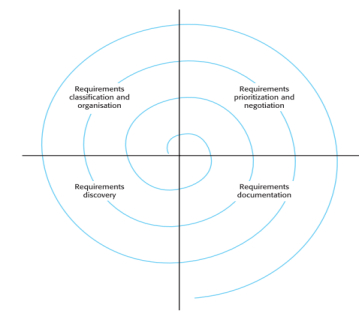
3. Different stakeholders have different requirements, which they may express in different ways. Requirements engineers have to consider all potential sources of requirements and discover commonalities and conflict.

M.Sc. Semester – III (Computer Science) 4

Paper – II (Software Engineering)

4. Political factors may influence the requirements of the system. For example, managers may demand specific system requirements that will increase their influence in the organisation. 5. The economic and business environment in which the analysis takes place is dynamic.

It inevitably changes during the analysis process. Hence the importance of particular requirements may change. New requirements may emerge from new stakeholders who were not originally consulted. A very general process model of the elicitation and analysis process is shown in Figure 2.3. Each organisation will have its own version or instantiation of this

**Fig. 2.3 The requirements elicitation and analysis process**

general model, depending on local factors such as the expertise of the staff, the type of system being developed and the standards used. Again, you can think of these activities within a spiral so that the activities are interleaved as the process proceeds from the inner to the outer rings of the spiral. 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.

M.Sc. Semester – III (Computer Science) 5

Paper – II (Software Engineering)

2. Requirements classification and organization - 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. Figure 2.3 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. Focus primarily on requirements discovery and the various techniques that have been developed to support this. Requirements classification and organization is primarily concerned with identifying overlapping requirements from different stakeholders and grouping related requirements. The most common way of grouping requirements is to use a model of the system architecture to identify subsystems and to associate requirements with each sub-system.

This emphasises that requirements engineering and architectural design cannot always be separated. Inevitably, stakeholders have different views on the importance and priority of requirements, and sometimes these views conflict. During the process, you should organise regular stakeholder negotiations so that compromises can be reached. It is impossible to completely satisfy every stakeholder, but if some stakeholders feel that their views have not been properly considered, they may deliberately attempt to undermine the RE process. In the requirements documentation stage, the requirements that have been elicited are documented in such a way that they can be used to help with further requirements discovery. At this stage, an early version of the system requirements document may be produced, but it will have missing sections and incomplete requirements. Alternatively, the requirements may be documented as tables in a document or on cards. Writing requirements on cards (the approach used in extreme programming) can be very effective, as these are easy for stakeholders to handle, change and organise.

M.Sc. Semester – III (Computer Science) 6

Paper – II (Software Engineering)

**2.2.1 Requirements discovery**

Requirements discovery is the process of gathering information about the proposed and existing systems and distilling the user and system requirements from this information. Sources of information during the requirements discovery phase include documentation, system stakeholders and specifications of similar systems. You interact with stakeholders through interviews and observation, and may use scenarios and prototypes to help with the requirements discovery, discuss an approach that helps ensure you get broad stakeholder coverage when discovering requirements, and describe techniques of requirements discovery including interviewing, scenarios and ethnography. Other requirements discovery techniques that may be used include structured analysis methods, and system prototyping, covered. Stakeholders range from system end-users through managers and external stakeholders such as regulators who certify the acceptability of the system. For example, system stakeholders for a bank ATM include:

1. Current bank customers who receive services from the system

2. Representatives from other banks who have reciprocal agreements that allow each other’s ATMs to be used

3. Managers of bank branches who obtain management information from the system 4. Counter staff at bank branches who are involved in the day-to-day running of the system 5. Database administrators who are responsible for integrating the system with the bank’s customer database

6. Bank security managers who must ensure that the system will not pose a security hazard 7. The bank’s marketing department who are likely be interested in using the system as a means of marketing the bank

8. Hardware and software maintenance engineers who are responsible for maintaining and upgrading the hardware and software

9. National banking regulators who are responsible for ensuring that the system conforms to banking regulations In addition to system stakeholders, we have already seen that requirements may come from the application domain and from other systems that interact with the system being specified.

All of these must be considered during the requirements elicitation process. These requirements sources (stakeholders, domain, systems) can all be represented as system

M.Sc. Semester – III (Computer Science) 7

Paper – II (Software Engineering)

viewpoints, where each viewpoint presents a sub-set of the requirements for the system. Each viewpoint provides a fresh perspective on the system, but these perspectives are not completely independent—they usually overlap so that they have common requirements. **Viewpoints**

Viewpoint-oriented approaches to requirements engineering; organise both the elicitation process and the requirements themselves using viewpoints. A key strength of viewpoint-oriented analysis is that it recognizes multiple perspectives and provides a framework for discovering conflicts in the requirements proposed by different stakeholders.

Viewpoints can be used as a way of classifying stakeholders and other sources of requirements. There are three generic types of viewpoint:

1. Interactor viewpoints represent people or other systems that interact directly with the system. In the bank ATM system, examples of interactor viewpoints are the bank’s customers and the bank’s account database.

2. Indirect viewpoints represent stakeholders who do not use the system themselves but who influence the requirements in some way. In the bank ATM system, examples of indirect viewpoints are the management of the bank and the bank security staff.

3. Domain viewpoints represent domain characteristics and constraints that influence the system requirements. In the bank ATM system, an example of a domain viewpoint would be the standards that have been developed for interbank communications.

Typically, these viewpoints provide different types of requirements. Interactor viewpoints provide detailed system requirements covering the system features and interfaces. Indirect viewpoints are more likely to provide higher-level organisational requirements and constraints. Domain viewpoints normally provide domain constraints that apply to the system. The initial identification of viewpoints that are relevant to a system can sometimes be difficult. To help with this process, you should try to identify more specific viewpoint types:

1. Providers of services to the system and receivers of system services

2. Systems that should interface directly with the system being specified

3. Regulations and standards that apply to the system

4. The sources of system business and non-functional requirements

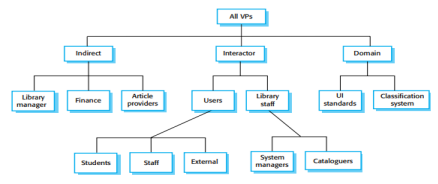
5. Engineering viewpoints reflecting the requirements of people who have to develop, manage and maintain the system

M.Sc. Semester – III (Computer Science) 8

Paper – II (Software Engineering)

6. Marketing and other viewpoints that generate requirements on the product features expected by customers and how the system should reflect the external image of the organisation Almost all organisational systems must interoperate with other systems in the organisation.

When a new system is planned, the interactions with other systems must be planned. The interfaces offered by these other systems have already been designed. These may place requirements and constraints on the new system. Furthermore, new systems may have to conform to existing regulations and standards, and these constrain the system requirements.

**Fig. 2.4 Viewpoints in LIBSYS**

Identify high-level business and non-functional requirements early in the RE process. The sources of these requirements may be useful viewpoints in a more detailed process. They may be able to expand and develop the high-level requirements into more specific system requirements. Engineering viewpoints may be important for two reasons.

Firstly, the engineers developing the system may have experience with similar systems and may be able to suggest requirements from that experience.

Secondly, technical staff who have to manage and maintain the system may have requirements that will help simplify system support. System management requirements are increasingly important because system management costs are an increasing proportion of the total lifetime costs for a system.

Finally, viewpoints that provide requirements may come from the marketing and external affairs departments in an organisation. This is especially true for web-based systems, particularly e commerce systems and shrink-wrapped software products.

M.Sc. Semester – III (Computer Science) 9

Paper – II (Software Engineering)

Web-based systems must present a favorable image of the organisation as well as deliver functionality to the user. For software products, the marketing department should know what system features will make the system more marketable to potential buyers. For any non-trivial system, there are a huge number of possible viewpoints, and it is practically impossible to elicit requirements from all of them. Therefore, it is important that you organise and structure the viewpoints into a hierarchy. Viewpoints in the same branch are likely to share common requirements. As an illustration, consider the viewpoint hierarchy shown in Figure 2.4. This is a relatively simple diagram of the viewpoints that may be consulted in deriving the requirements for the LIBSYS system. You can see that the classification of interactor, indirect and domain viewpoints helps identify sources of requirements apart from the immediate users of the system. Once viewpoints have been identified and structured, you should try to identify the most important viewpoints and start with them when discovering system requirements.

**Interviewing**

Formal or informal interviews with system stakeholders are part of most requirements engineering processes. In these interviews, the requirements engineering team puts questions to stakeholders about the system that they use and the system to be developed. Requirements are derived from the answers to these questions. Interviews may be of two types:

1. Closed interviews where the stakeholder answers a predefined set of questions. 2. Open interviews where there is no predefined agenda.

The requirements engineering team explores a range of issues with system stakeholders and hence develops a better understanding of their needs. In practice, interviews with stakeholders are normally a mix of these. The answers to some questions may lead to other issues that are discussed in a less structured way.

Completely open-ended discussions rarely work well; most interviews require some questions to get started and to keep the interview focused on the system to be developed. Interviews are good for getting an overall understanding of what stakeholders do, how they might interact with the system and the difficulties that they face with current systems. People like talking about their work and are usually happy to get involved in interviews. However, interviews are not so good for understanding the requirements from the application domain. It is hard to elicit domain knowledge during interviews for two reasons:

M.Sc. Semester – III (Computer Science) 10

Paper – II (Software Engineering)

1. All application specialists use terminology and jargon that is specific to a domain. It is impossible for them to discuss domain requirements without using this terminology. They normally use terminology in a precise and subtle way that is easy for requirements engineers to misunderstand.

2. Some domain knowledge is so familiar to stakeholders that they either find it difficult to explain or they think it is so fundamental that it isn’t worth mentioning.

For example, for a librarian, it goes without saying that all acquisitions are catalogued before they are added to the library. However, this may not be obvious to the interviewer so it isn’t taken into account in the requirements. Interviews are not an effective technique for eliciting knowledge about organisational requirements and constraints because there are subtle power and influence relationships between the stakeholders in the organisation. Published organisational structures rarely match the reality of decision making in an organisation, but interviewees may not wish to reveal the actual rather than the theoretical structure to a stranger. In general, most people are reluctant to discuss political and organisational issues that may affect the requirements. Effective interviewers have two characteristics:

1. They are open-minded, avoid preconceived ideas about the requirements and are willing to listen to stakeholders. If the stakeholder comes up with surprising requirements, they are willing to change their mind about the system.

2. They prompt the interviewee to start discussions with a question, a requirements proposal or by suggesting working together on a prototype system. Saying to people ‘tell me what you want’ is unlikely to result in useful information.

Most people find it much easier to talk in a defined context rather than in general terms. Information from interviews supplements other information about the system from documents, user observations, and so on. Sometimes, apart from information from documents, interviews may be the only source of information about the system requirements. However, interviewing on its own is liable to miss essential information, so it should be used alongside other requirements elicitation techniques.

**Scenarios**

People usually find it easier to relate to real-life examples than to abstract descriptions. They can understand and critique a scenario of how they might interact with a software system. Requirements engineers can use the information gained from this discussion to formulate the

M.Sc. Semester – III (Computer Science) 11

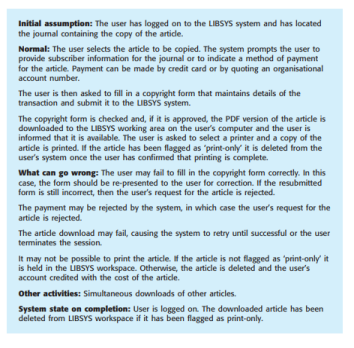
Paper – II (Software Engineering)

actual system requirements. Scenarios can be particularly useful for adding detail to an outline requirements description. They are descriptions of example interaction sessions. Each scenario covers one or more possible interactions. Several forms of scenarios have been developed, each of which provides different types of information at different levels of detail about the system. Using scenarios to describe requirements is an integral part of agile methods, such as extreme programming. The scenario starts with an outline of the interaction, and, during elicitation, details are added to create a complete description of that interaction. At its most general, a scenario may include:

1. A description of what the system and users expect when the scenario starts 2. A description of the normal flow of events in the scenario

3. A description of what can go wrong and how this is handled

4. Information about other activities that might be going on at the same time 5. A description of the system state when the scenario finishes

**Fig. 2.5 Scenario for article downloading in LIBSYS**

M.Sc. Semester – III (Computer Science) 12

Paper – II (Software Engineering)

Scenario-based elicitation can be carried out informally, where the requirements engineer works with stakeholders to identify scenarios and to capture details of these scenarios. Scenarios may be written as text, supplemented by diagrams, screen shots, and so on. Alternatively, a more structured approach such as event scenarios or use cases may be adopted. As an example of a simple text scenario, consider how a user of the LIBSYS library system may use the system. This scenario is shown in Figure 2.5. The user wishes to print a personal copy of an article in a medical journal. This journal makes copies of articles available free to subscribers, but nonsubscribers have to pay a fee per article. The user knows the article, title and date of publication.

**Use-cases**

Use-cases are a scenario-based technique for requirements elicitation which were first introduced in the Objectory method. They have now become a fundamental feature of the UML notation for describing object-oriented system models.



**Fig. 2.6 A simple use-case for article printing**

In their simplest form, a use-case identifies the type of interaction and the actors involved . For example, Figure 2.6 shows the high-level use-case of the article printing facility in LIBSYS described in Figure 2.5. Figure 2.6 illustrates the essentials of the use-case notation. Actors in the process are represented as stick figures, and each class of interaction is represented as a named ellipse. The set of use-cases represents all of the possible interactions to be represented in the system requirements. Figure 2.7 develops the LIBSYS example and shows other use-cases in that environment. There is sometimes confusion about whether a use-case is a scenario on its own or, as suggested by Fowler, a use-case encapsulates a set of scenarios, and each scenario is a single thread through the use-case. If a scenario includes multiple threads, there would be a scenario for the normal interaction plus scenarios for each possible exception. Use-cases identify the individual interactions with the system. They can be documented with text or linked to UML models that develop the scenario in more detail. Sequence diagrams are often used to add information to a use-case. These sequence diagrams show the actors involved in the interaction,

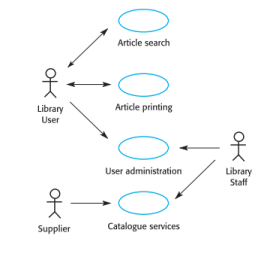
M.Sc. Semester – III (Computer Science) 13

Paper – II (Software Engineering)

the objects they interact with and the operations associated with these objects. As an illustration of this, Figure 2.8 shows the interactions involved in using LIBSYS for downloading and printing an article.

In Figure 2.8, there are four objects of classes—Article, Form, Workspace and Printer— involved in this interaction. The sequence of actions is from top to bottom, and the labels on the arrows between the actors and objects indicate the names of operations. Essentially, a user request for an article triggers a request for a copyright form. Once the user has completed the form, the article is downloaded and sent to the printer. Once printing is complete, the article is deleted from the LIBSYS workspace. The UML is a de facto standard for object-oriented modelling, so use-cases and use-case–based elicitation is increasingly used for requirements elicitation.

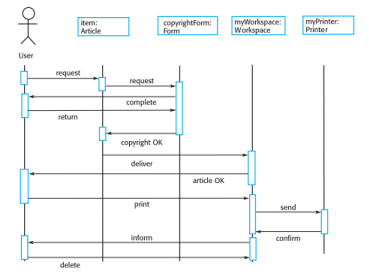
Scenarios and use-cases are effective techniques for eliciting requirements for interactor viewpoints, where each type of interaction can be represented as a use case. They can also be used in conjunction with some indirect viewpoints where these viewpoints receive some results (such as a management report) from the system. However, because they focus on interactions, they are not as effective for eliciting constraints or high-level business and non-functional requirements from indirect viewpoints or for discovering domain requirements.



**Fig. 2.7 Use cases for the library system**

M.Sc. Semester – III (Computer Science) 14

Paper – II (Software Engineering)



**Fig. 2.8 System interactions for article printing**

**2.2.2 Ethnography**

Software systems do not exist in isolation—they are used in a social and organisational context, and software system requirements may be derived or constrained by that context. Satisfying these social and organisational requirements is often critical for the success of the system. One reason why many software systems are delivered but never used is that they do not take proper account of the importance of these requirements. 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. People often find it very difficult to articulate details of their work because it is second nature to them. They understand their own work but may not understand its relationship with other work in the organisation. Social and organisational factors that affect the work but that are not obvious to individuals may only become clear when noticed by an unbiased observer. Suchman used

M.Sc. Semester – III (Computer Science) 15

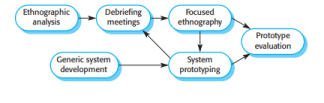
Paper – II (Software Engineering)

ethnography to study office work and found that the actual work practices were far richer, more complex and more dynamic than the simple models assumed by office automation systems. The difference between the assumed and the actual work was the most important reason why these office systems have had no significant effect on productivity. Other ethnographic studies for system requirements understanding have included work on air traffic control, underground railway control rooms, financial systems and various design activities. Methods of integrating ethnography into the software engineering process by linking it to requirements engineering methods and by documenting patterns of interaction in cooperative systems. Ethnography is particularly effective at discovering two types of requirements:

1. Requirements that are derived from the way in which people actually work rather than the way in which process definitions say they ought to work. For example, air traffic controllers may switch off an aircraft conflict alert system that detects aircraft with intersecting flight paths even though normal control procedures specify that it should be used. Their control strategy is designed to ensure that these aircraft are moved apart before problems occur and they find that

the conflict alert alarm distracts them from their work.

2. Requirements that are derived from cooperation and awareness of other people’s activities. For example, air traffic controllers may use an awareness of other controllers’ work to predict the number of aircraft that will be entering their control sector.



**Fig. 2.9 Ethnography and prototyping for requirements analysis**

They then modify their control strategies depending on that predicted workload. Therefore, an automated ATC system should allow controllers in a sector to have some visibility of the work in adjacent sectors. Ethnography may be combined with prototyping (Figure 2.9). The ethnography informs the development of the prototype so that fewer prototype refinement cycles are required.

Furthermore, the prototyping focuses the ethnography by identifying problems and questions that can then be discussed with the ethnographer. He or she should then look for the

M.Sc. Semester – III (Computer Science) 16

Paper – II (Software Engineering)

answers to these questions during the next phase of the system study. Ethnographic studies can reveal critical process details that are often missed by other requirements elicitation techniques. However, because of its focus on the end user, this approach is not appropriate for discovering organisational or domain requirements. Ethnographic studies cannot always identify new features that should be added to a system. Ethnography is not, therefore, a complete approach to elicitation on its own, and it should be used to complement other approaches, such as use-case analysis.

**2.3 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. 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.

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.

M.Sc. Semester – III (Computer Science) 17

Paper – II (Software Engineering)

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. A number of requirements validation techniques can be used in conjunction or individually:

1. Requirements reviews - The requirements are analysed systematically by a team of reviewers. This process is discussed in the following.

2. 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.

3. 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 a 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. You should not underestimate the problems of requirements validation. It is difficult to show that a set of requirements meets a user’s needs. Users must picture the system in operation and imagine how that system would fit into their work. It is hard for skilled computer professionals to perform this type of abstract analysis and even harder for system users. As a result, you rarely find all requirements problems during the requirements validation process. It is inevitable that there will be further requirements changes to correct omissions and misunderstandings after the requirements document has been agreed upon.

**2.3.1 Requirements reviews**

A requirements review is a manual process that involves people from both client and contractor organisations. They check the requirements document for anomalies and omissions. The review process may be managed in the same way as program inspections. Alternatively, it may be organised as a broader activity with different people checking different parts of the document. Requirements reviews can be informal or formal. Informal reviews simply involve contractors discussing requirements with as many system stakeholders as possible. It is surprising how often communication between system developers and stakeholders ends after elicitation and there is no confirmation that the documented requirements are what the

M.Sc. Semester – III (Computer Science) 18

Paper – II (Software Engineering)

stakeholders really said they wanted. Many problems can be detected simply by talking about the system to stakeholders before making a commitment to a formal review. In a formal requirements review, the development team should ‘walk’ the client through the system requirements, explaining the implications of each requirement. The review team should check each requirement for consistency as well as check the requirements as a whole for completeness. Reviewers may also check for:

1. Verifiability - Is the requirement as stated realistically testable?

2. Comprehensibility - Do the procurers or end-users of the system properly understand the requirement?

3. Traceability - Is the origin of the requirement clearly stated?

You may have to go back to the source of the requirement to assess the impact of a change. Traceability is important as it allows the impact of change on the rest of the system to be assessed. Adaptability Is the requirement adaptable? That is, can the requirement be changed without large-scale effects on other system requirements? Conflicts, contradictions, errors and omissions in the requirements should be pointed out by reviewers and formally recorded in the review report. It is then up to the users, the system procurer and the system developer to negotiate a solution to these identified problems.

**2.4 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. 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 organisation. Once end-users have experience of a system, they discover new needs and priorities:

1. 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.

M.Sc. Semester – III (Computer Science) 19

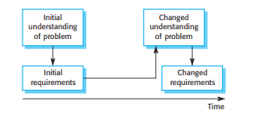
Paper – II (Software Engineering)

2. The people who pay for a system and the users of a system are rarely the same people. System customers impose requirements because of organisational 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.

3. 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. 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.

**2.4.1 Enduring and volatile requirements**

Requirements evolution during the RE process and after a system has gone into service is inevitable. Developing software requirements focuses attention on software 

**Fig. 2.10 Requirements of evaluation**

capabilities, business objectives and other business systems. As the requirements definition is developed, you normally develop a better understanding of users’ needs. This feeds information back to the user, who may then propose a change to the requirements (Figure 2.10). Furthermore, it may take several years to specify and develop a large system. Over that time, the system’s

M.Sc. Semester – III (Computer Science) 20

Paper – II (Software Engineering)

environment and the business objectives change, and the requirements evolve to reflect this. From an evolution perspective, requirements fall into two classes:

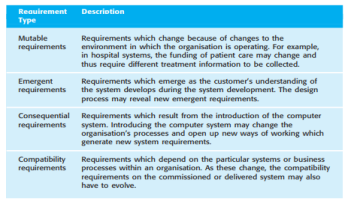
1. Enduring requirements - These are relatively stable requirements that derive from the core activity of the organisation and which relate directly to the domain of the system. For example, in a hospital, there will always be requirements concerned with patients, doctors, nurses and treatments. These requirements may be derived from domain models that show the entities and relations that characterise an application domain.

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. Have suggested that volatile requirements fall into five classes. Using these as a base, have developed the classification shown in Figure 2.11.

**2.4.2 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. **Fig. 2.11 Classification of volatile requirements**

M.Sc. Semester – III (Computer Science) 21

Paper – II (Software Engineering)

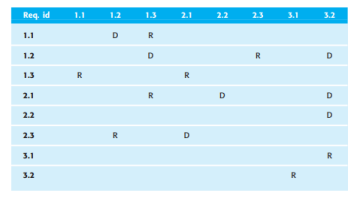
2. A change management process - This is the set of activities that assess the impact and cost of changes. I discuss this process in more detail in the following.

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. There are many relationships among requirements and between the requirements and the system design. There are also links between requirements and the underlying reasons why these requirements were proposed. When changes are proposed, you have to trace the impact of these changes on other requirements and the system design. Traceability is the property of a requirements specification that reflects the ease of finding related requirements. There are three types of traceability information that may be maintained:

1. Source traceability information links the requirements to the stakeholders who proposed the requirements and to the rationale for these requirements. When a change is proposed, you use this information to find and consult the stakeholders about the change.

**Fig. 2.12 A traceability matrix**

2. Requirements traceability information links dependent requirements within the requirements document. You use this information to assess how many requirements are likely to be affected by a proposed change and the extent of consequential requirements changes that may be necessary.

M.Sc. Semester – III (Computer Science) 22

Paper – II (Software Engineering)

3. Design traceability information links the requirements to the design modules where these requirements are implemented.

You use this information to assess the impact of proposed requirements changes on the system design and implementation. Traceability information is often represented using traceability matrices, which relate requirements to stakeholders, each other or design modules. In a requirements traceability matrix, each requirement is entered in a row and in a column in the matrix. Where dependencies between different requirements exist, these are recorded in the cell at the row/column intersection. Figure 2.12 shows a simple traceability matrix that records the dependencies between requirements. A ‘D’ in the row/column intersection illustrates that the requirement in the row depends on the requirement named in the column; an ‘R’ means that there is some other, weaker relationship between the requirements. For example, they may both define the requirements for parts of the same subsystem. Traceability matrices may be used when a small number of requirements have to be managed, but they become unwieldy and expensive to maintain for large systems with many requirements.

For these systems, you should capture traceability information in a requirements database where each requirement is explicitly linked to related requirements. You can then assess the impact of changes by using the database browsing facilities. Traceability matrices can be generated automatically from the database.

**Fig. 2.13 Requirements change management**

Requirements management needs automated support; the CASE tools for this should be chosen during the planning phase. You need tool support for:

1. Requirements storage - The requirements should be maintained in a secure, managed data store that is accessible to everyone involved in the requirements engineering process. 2. Change management - The process of change management (Figure 2.13) is simplified if active tool support is available.

3. Traceability management - As discussed above, tool support for traceability allows related requirements to be discovered.

M.Sc. Semester – III (Computer Science) 23

Paper – II (Software Engineering)

Some tools use natural language processing techniques to help you discover possible relationships between the requirements.

For small systems, it may not be necessary to use specialised requirements management tools. The requirements management process may be supported using the facilities available in word processors, spreadsheets and PC databases. However, for larger systems, more specialised tool support is required. I have included links to information about requirements management tools such as DOORS and RequisitePro in the book’s web pages.

**3.4.3 Requirements change management**

Requirements change management (Figure 2.13) should be applied to all proposed changes to the requirements. The advantage of using a formal process for change management is that all change proposals are treated consistently and that changes to the requirements document are made in a controlled way. There are three principal stages to a change management process: 1. Problem analysis and change specification - The process starts with an identified requirements problem or, sometimes, with a specific change proposal. During this stage, the problem or the change proposal is analysed to check that it is valid. The results of the analysis are fed back to the change requestor, and sometimes a more specific requirements change proposal is then made. 2. Change analysis and costing - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. The cost of making the change is estimated in terms of modifications to the requirements document and, if appropriate, to the system design and implementation. Once this analysis is completed, a decision is made whether to proceed with the requirements change.

3. Change implementation - The requirements document and, where necessary, the system design and implementation are modified.

You should organise the requirements document so that you can make changes to it without extensive rewriting or reorganisation. As with programs, changeability in documents is achieved by minimising external references and making the document sections as modular as possible. Thus, individual sections can be changed and replaced without affecting other parts of the document. If a requirements change to a system is urgently required, there is always a temptation to make that change to the system and then retrospectively modify the requirements document. This almost inevitably leads to the requirements specification and the system implementation getting out of step. Once system changes have been made, requirements

M.Sc. Semester – III (Computer Science) 24

Paper – II (Software Engineering)

document changes may be forgotten or made in a way that is not consistent with the system changes. Iterative development processes, such as extreme programming, have been designed to cope with requirements that change during the development process. In these processes, when a user proposes a requirements change, this does not go through a formal change management process. Rather, the user has to prioritise that change and, if it is high priority, decide what system features that were planned for the next iteration should be dropped.

**3. System models**

User requirements should be written in natural language because they have to be understood by people who are not technical experts. However, more detailed system requirements may be expressed in a more technical way. One widely used technique is to document the system specification as a set of system models. These models are graphical representations that describe business processes, the problem to be solved and the system that is to be developed. Because of the graphical representations used, models are often more understandable than detailed natural language descriptions of the system requirements. They are also an important bridge between the analysis and design processes. You can use models in the analysis process to develop an understanding of the existing system that is to be replaced or improved or to specify the new system that is required. You may develop different models to represent the system from different perspectives.

For example:

1. An external perspective, where the context or environment of the system is modeled. 2. A behavioural perspective, where the behaviour of the system is modeled. 3. A structural perspective, where the architecture of the system or the structure of the data processed by the system is modelled cover these three perspectives in this chapter and also discuss object modelling, which combines, to some extent, behavioural and structural modelling.

The most important aspect of a system model is that it leaves out detail. A system model is an abstraction of the system being studied rather than an alternative representation of that system. Ideally, a representation of a system should maintain all the information about the entity being represented. An abstraction deliberately simplifies and picks out the most salient characteristics. For example, in the very unlikely event of this book being serialised in a newspaper, the presentation there would be an abstraction of the book’s key points. If it were translated from English into Italian, this would be an alternative representation. The translator’s

M.Sc. Semester – III (Computer Science) 25

Paper – II (Software Engineering)

intention would be to maintain all the information as it is presented in English. Different types of system models are based on different approaches to abstraction. A data-flow model (for example) concentrates on the flow of data and the functional transformations on that data. It leaves out details of the data structures. By contrast, a model of data entities and their relationships documents the system data structures rather than its functionality. Examples of the types of system models that you might create during the analysis process are:

1. A data- flow model Data-flow models show how data is processed at different stages in the system.

2. A composition model - A composition or aggregation model shows how entities in the system are composed of other entities.

3. An architectural model Architectural models show the principal sub-systems that make up a system.

4. A classification model Object class/inheritance diagrams show how entities have common characteristics.

5. A stimulus-response model A stimulus-response model, or state transition diagram, shows how the system reacts to internal and external events.

All these types of models are covered in this chapter. Wherever possible, I use notations from the Unified Modeling Language (UML), which has become a standard modelling language for object-oriented modelling. Where UML does not include appropriate notations, I use simple intuitive notations for model description. A new version of UML (UML 2.0) is under development but was not available when wrote this chapter. However, understand that the UML notation that have used here is likely to be compatible with UML 2.0.

**3.1 Context models**

At an early stage in the requirements elicitation and analysis process you should decide on the boundaries of the system. This involves working with system stakeholders to distinguish what is the system and what is the system’s environment. You should make these decisions early in the process to limit the system costs and the time needed for analysis. In some cases, the boundary between a system and its environment is relatively clear.

For example, where an automated system is replacing an existing manual or computerized system, the environment of the new system is usually the same as the existing system’s environment. In other cases, there is more flexibility, and you decide what constitutes

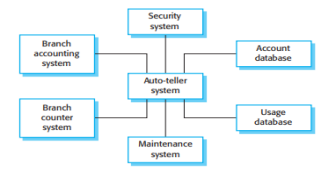
M.Sc. Semester – III (Computer Science) 26

Paper – II (Software Engineering)

the boundary between the system and its environment during the requirements engineering process. For example, say you are developing the specification for the library system LIBSYS. Recall that this system is intended to deliver electronic versions of copyrighted material to users’ computers.

The users may then print personal copies of the material. In developing the specification for this system, you have to decide whether other library database systems such as library catalogues are within the system boundary. If they are, then you may have to allow access to the system through the catalogue user interface; if they are not, then users may be inconvenienced by having to move from one system to another. The definition of a system boundary is not a value

free judgement. Social and organisational concerns may mean that the position of a system boundary may be determined by non-technical factors. For example, a system boundary may be positioned.



**Fig. 2.14 The context of an ATM system**

So, that the analysis process can all be carried out on one site; it may be chosen so that a particularly difficult manager need not be consulted; it may be positioned so that the system cost is increased, and the system development division must therefore expand to design and implement the system. Once some decisions on the boundaries of the system have been made, part of the analysis activity is the definition of that context and the dependencies that a system has on its environment. Normally, producing a simple architectural model is the first step in this activity.

Figure 2.14 is an architectural model that illustrates the structure of the information system that includes a bank auto-teller network. High-level architectural models are usually

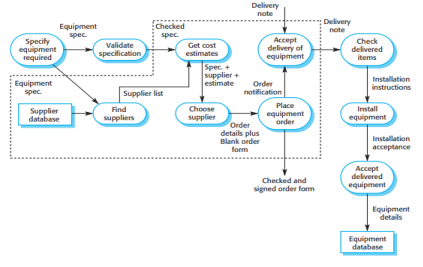
M.Sc. Semester – III (Computer Science) 27

Paper – II (Software Engineering)

expressed as simple block diagrams where each sub-system is represented by a named rectangle, and lines indicate associations between sub-systems. From Figure 2.14, we see that each ATM is connected to an account database, a local branch accounting system, a security system and a system to support machine maintenance. The system is also connected to a usage database that monitors how the network of ATMs is used and to a local branch counter system. This counter system provides services such as backup and printing.

These, therefore, need not be included in the ATM system itself. Architectural models describe the environment of a system. However, they do not show the relationships between the other systems in the environment and the system that is being specified. External systems might produce data for or consume data from the system. They might share data with the system, or they might be connected directly, through a network or not at all. They might be physically co

located or located in separate buildings. All of these relations might affect the requirements of the system being defined and must be taken into account. Therefore, simple architectural models are normally supplemented by other models, such as process models, that show the process activities supported by the system. Data-flow models may also be used to show the data that is transferred between the system and other systems in its environment.

**Fig. 2.15 Process model of equipment procurement**

M.Sc. Semester – III (Computer Science) 28

Paper – II (Software Engineering)

Figure 2.15 illustrates a process model for the process of procuring equipment in an organisation. This involves specifying the equipment required, finding and choosing suppliers, ordering the equipment, taking delivery of the equipment and testing it after delivery. When specifying computer support for this process, you have to decide which of these activities will actually be supported. The other activities are outside the boundary of the system. In Figure 2.15, the dotted line encloses the activities that are within the system boundary.

**3.2 Behavioural models**

Behavioural models are used to describe the overall behaviour of the system. discuss two types of behavioural model here: data-flow models, which model the data processing in the system, and state machine models, which model how the system reacts to events. These models may be used separately or together, depending on the type of system that is being developed.

**Fig. 2.15 Data-flow diagram of order processing**

Most business systems are primarily driven by data. They are controlled by the data inputs to the system with relatively little external event processing. A dataflow model may be all that is needed to represent the behaviour of these systems. By contrast, real-time systems are often event-driven with minimal data processing. A state machine model is the most effective way to represent their behaviour. Other classes of system may be both data and event driven. In these cases, you may develop both types of model.

**3.2.1 Data-flow models**

Data-flow models are an intuitive way of showing how data is processed by a system. At the analysis level, they should be used to model the way in which data is processed in the existing system. The use of data-flow models for analysis became widespread after the publication of DeMarco’s book on structured systems analysis. They are an intrinsic part of

M.Sc. Semester – III (Computer Science) 29

Paper – II (Software Engineering)

structured methods that have been developed from this work. The notation used in these models represents functional processing (rounded rectangles), data stores (rectangles) and data movements between functions (labelled arrows). Data-flow models are used to show how data flows through a sequence of processing steps.

For example, a processing step could be to filter duplicate records in a customer database. The data is transformed at each step before moving on to the next stage. These processing steps or transformations represent software processes or functions when data-flow diagrams are used to document a software design. However, in an analysis model, people or computers may carry out the processing. A data-flow model, which shows the steps involved in processing an order for goods (such as computer equipment) in an organisation, is illustrated in Figure 2.15. This particular model describes the data processing in the Place equipment order

**Fig. 2.16 Data-flow diagram of an insulin pump**

activity in the overall process model shown in Figure 2.14. The model shows how the order for the goods moves from process to process. It also shows the data stores (Orders file and Budget file) that are involved in this process. Data-flow models are valuable because tracking and documenting how the data associated with a particular process moves through the system helps analysts understand what is going on. Data-flow diagrams have the advantage that, unlike some other modelling notations, they are simple and intuitive. It is usually possible to explain them to potential system users who can then participate in validating the analysis. In principle, the development of models such as data-flow models should be a ‘top-down’ process.

In this example, this would imply that you should start by analysing the overall procurement process. You then move on to the analysis of sub-processes such as ordering. In practice, analysis is never like that. You learn about several levels at the same time. Lower-level

M.Sc. Semester – III (Computer Science) 30

Paper – II (Software Engineering)

models may be developed first and then abstracted to create a more general model. Data-flow models show a functional perspective where each transformation represents a single function or process. They are particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system. That is, they show the entire sequence of actions that take place from an input being processed to the corresponding output that is the system’s response. Figure 2.14 illustrates this use of data flow diagrams. It is a diagram of the processing that takes place in the insulin pump system.

**3.2.2 State machine models**

A state machine model describes how a system responds to internal or external events. The state machine model shows system states and events that cause transitions from one state to another. It does not show the flow of data within the system. This type of model is often used for modelling real-time systems because these systems are often driven by stimuli from the system’s environment. For example, the real-time

**Fig. 2.17 State machine model of a simple microwave oven**

alarm system discussed responds to stimuli from movement sensors, door opening sensors, and so on. State machine models are an integral part of real-time design methods such as that proposed by Ward and Mellor and Harel. Harel’s method uses a notation called Statecharts and

M.Sc. Semester – III (Computer Science) 31

Paper – II (Software Engineering)

these were the basis for the state machine-modelling notation in the UML. A state machine model of a system assumes that, at any time, the system is in one of a number of possible states. When a stimulus is received, this may trigger a transition to a different state.

For example, a system controlling a valve may move from a state ‘Valve open’ to a state ‘Valve closed’ when an operator command (the stimulus) is received. This approach to system modelling is illustrated in Figure 2.16. This diagram shows a state machine model of a simple microwave oven equipped with buttons to set the power and the timer and to start the system. Real microwave ovens are actually much more complex than the system described here. However, this model includes the essential features of the system. To simplify the model, I have assumed that the sequence of actions in using the microwave is:

1. Select the power level (either half-power or full-power).

2. Input the cooking time.

3. Press Start, and the food is cooked for the given time.

For safety reasons, the oven should not operate when the door is open and, on completion of cooking, a buzzer is sounded. The oven has a very simple alphanumeric display that is used to display various alerts and warning messages. The UML notation that I use to describe state machine models is designed for modelling the behaviour of objects. However, it is a general

purpose notation that can be used for any type of state machine modelling. The rounded rectangles in a model represent system states. They include a brief description (following ‘do’) of the actions taken in that state. The labelled arrows represent stimuli that force a transition from one state to another. Therefore, from Figure 2.16, we can see that the system responds initially to either the full-power or the half-power button. Users can change their mind after selecting one of these and press the other button. The time is set and, if the door is closed, the Start button is enabled. Pushing this button starts the oven operation and cooking takes place for the specified time.

The UML notation lets you indicate the activity that takes place in a state. In a detailed system specification, you have to provide more detail about both the stimuli and the system states (Figure 2.17). This information may be maintained in a data dictionary or encyclopaedia (covered in Section 2.15) that is maintained by the CASE tools used to create the system model. The problem with the state machine approach is that the number of possible states increases rapidly. For large system models, therefore, some structuring of these state models is necessary.

M.Sc. Semester – III (Computer Science) 32

Paper – II (Software Engineering)

One way to do this is by using the notion of a superstate that encapsulates a number of separate states.

This superstate looks like a single state on a high-level model but is then expanded in more detail on a separate diagram. To illustrate this concept, consider the Operation state in Figure 2.15. This is a superstate that can be expanded, as illustrated in Figure 2.17. The Operation state includes a number of sub-states. It shows that operation starts with a status check, and that if any problems are discovered, an alarm is indicated and operation is disabled. Cooking involves running the microwave generator for the specified time; on completion, a buzzer is sounded. If the door is opened during operation, the system moves to the disabled state, as shown in Figure 2.15.

**3.3 Data models**

Most large software systems make use of a large database of information. In some cases, this database is independent of the software system. In others, it is created for the system being developed. An important part of systems modelling is defining the logical form of the data processed by the system. These are sometimes called semantic data models. The most widely used data modelling technique is Entity-Relation-Attribute modelling (ERA modelling), which shows the data entities, their associated attributes and the relations between these entities. This approach to modelling was first proposed in the mid-1970s by Chen; several variants have been developed since then, all with the same basic form.

M.Sc. Semester – III (Computer Science) 33

Paper – II (Software Engineering)



**Fig. 2.18 State and stimulus description for the microwave oven Fig. 2.19 Microwave oven operation**

M.Sc. Semester – III (Computer Science) 34

Paper – II (Software Engineering)

Entity-relationship models have been widely used in database design. The relational database schemas derived from these models are naturally in third normal form, which is a desirable characteristic. Because of the explicit typing and the recognition of sub- and super types, it is also straightforward to implement these models using object-oriented databases. The UML does not include a specific notation for this database modelling, as it assumes an object oriented development process and models data using objects and their relationships.

However, you can use the UML to represent a semantic data model. You can think of entities in an ERA model as simplified object classes (they have no operations), attributes as class attributes and named associations between the classes as relations. Figure 2.18 is an example of a data model that is part of the library system LIBSYS introduced.

Recall that LIBSYS is designed to deliver copies of copyrighted articles that have been published in magazines and journals and to collect payments for these articles. Therefore, the data model must include information about the article, the copyright holder and the buyer of the article. Assumed that payments for articles are not made directly but through national copyright agencies. Figure 2.18 shows that an Article has attributes representing the title, the authors, the name of the PDF file of the article and the fee payable. This is linked to the Source, where the article was published, and to the Copyright Agency for the country of publication. Both Copyright Agency and Source are linked to Country. The country of publication is important because copyright laws vary by country. The diagram also shows that Buyers place Orders for Articles. Like all graphical models, data models lack detail, and you should maintain more detailed descriptions of the entities, relationships and attributes that are included in the model.

M.Sc. Semester – III (Computer Science) 35

Paper – II (Software Engineering)



**Fig. 2.19 Semantic data model for the LIBSYS system**

You may collect these more detailed descriptions in a repository or data dictionary. Data dictionaries are generally useful when developing system models and may be used to manage all information from all types of system models. A data dictionary is, simplistically, an alphabetic list of the names included in the system models. As well as the name, the dictionary should include an associated description of the named entity and, if the name represents a composite object, a description of the composition. Other information such as the date of creation, the creator and the representation of the entity may also be included depending on the type of model being developed. The advantages of using a data dictionary are:

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 organisational 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.

M.Sc. Semester – III (Computer Science) 36

Paper – II (Software Engineering)



**Fig. 2.20 Examples of data dictionary entries**

The data dictionary entries shown in Figure 2.19 define the names in the semantic data model for LIBSYS (Figure 2.18). Have simplified the presentation of this example by leaving out some names and by shortening the associated information. All system names, whether they are names of entities, relations, attributes or services, should be entered in the dictionary. Software is normally used to create, maintain and interrogate the dictionary. This software might be integrated with other tools so that dictionary creation is partially automated.

For example, CASE tools that support system modelling generally include support for data dictionaries and enter the names in the dictionary when they are first used in the model. **3.4 Object models**

An object-oriented approach to the whole software development process is now commonly used, particularly for interactive systems development. This means expressing the systems requirements using an object model, designing using objects and developing the system in an object-oriented programming language such as Java or C++. Object models that you develop during requirements analysis may be used to represent both system data and its processing. In this respect, they combine some of the uses of data-flow and semantic data models. They are also useful for showing how entities in the system may be classified and composed of other entities. For some classes of system, object models are natural ways of reflecting the real world entities that are manipulated by the system. This is particularly true

M.Sc. Semester – III (Computer Science) 37

Paper – II (Software Engineering)

when the system processes information about tangible entities, such as cars, aircraft or books, which have clearly identifiable attributes. More abstract, higher-level entities, such as the concept of a library, a medical record system or a word processor, are harder to model as object classes. They do not necessarily have a simple interface consisting of independent attributes and operations. Developing object models during requirements analysis usually simplifies the transition to object-oriented design and programming.

However, found that end users of a system often find object models unnatural and difficult to understand. They may prefer to adopt a more functional, data-processing view. Therefore, it is sometimes helpful to supplement object models with data-flow models that show the end-to-end data processing in the system. An object class is an abstraction over a set of objects that identifies common attributes (as in a semantic data model) and the services or operations that are provided by each object. Objects are executable entities with the attributes and services of the object class. Objects are instantiations of the object class, and many objects may be created from a class.

Generally, the models developed using analysis focus on object classes and their relationships. In object-oriented requirements analysis, you should model real-world entities using object classes. You should not include details of the individual objects (instantiations of the class) in the system. You may create different types of object models, showing how object classes are related to each other, how objects, are aggregated to form other objects, how objects interact with other objects and so on. These each present unique information about the system that is being specified. The analysis process for identifying objects and object classes is recognised as one of the most difficult areas of object-oriented development. Object identification is basically the same for analysis and design. The methods of object identification covered, which discusses object-oriented design, may be used, concentrate here on some of the object models that might be generated during the analysis process.

Various methods of object-oriented analysis were proposed in the 1990s. These methods had a great deal in common, and three of the key developers (Booch, Rumbaugh, and Jacobsen) decided to integrate their approaches to produce a unified method (Rumbaugh et al., 1999b). The Unified Modeling Language (UML) used in this unified method has become a standard for object modelling. The UML includes notations for different types of system models. We have already seen use-case models and sequence diagrams and state machine models. An object class

M.Sc. Semester – III (Computer Science) 38

Paper – II (Software Engineering)

in UML, as illustrated in the examples in Figure 2.20, is represented as a vertically oriented rectangle with three sections:

1. The name of the object class is in the top section.

2. The class attributes are in the middle section.

3. The operations associated with the object class are in the lower section of the rectangle. I don’t have space to cover all of the UML, so I focus here on object models that show how objects can be classified and can inherit attributes and operations from other objects, aggregation models that show how objects are composed, and simple behavioural models, which show object interactions.

**3.4.1 Inheritance models**

Object-oriented modelling involves identifying the classes of object that are important in the domain being studied. These are then organised into a taxonomy. A taxonomy is a classification scheme that shows how an object class is related to other classes through common attributes and services. To display this taxonomy, the classes are organised into an inheritance hierarchy with the most general object classes at the top of the hierarchy. More specialised objects inherit their attributes and services. These specialised objects may have their own attributes and services. Figure 2.20 illustrates part of a simplified class hierarchy for a model of a library. This hierarchy gives information about the items held in the library. The library holds various items, such as books, music, recordings of films, magazines and newspapers. In Figure 2.20, the most general item is at the top of the tree and has a set of attributes and services that are common to all library items.

These are inherited by the classes Published item and Recorded item, which add their own attributes that are then inherited by lower-level items. Figure 2.21 is an example of another inheritance hierarchy that might be part of the library model. In this case, the users of a library are shown. There are two classes of user: those who are allowed to borrow books, and those who may only read books in the library without taking them away. In the UML notation, inheritance is shown ‘upwards’ rather than ‘downwards’ as it is in some other object-oriented notations or in languages such as Java, where sub-classes inherit from super-classes. That is, the arrowhead (shown as a triangle) points from the classes that inherit attributes and operations to the super class. Rather than use the term inheritance, UML refers to the generalisation relationship.

M.Sc. Semester – III (Computer Science) 39

Paper – II (Software Engineering)



**Fig. 2.21 Part of a class hierarchy for a library**

****

**Fig. 2.22 User class hierarchy**

The design of class hierarchies is not easy, so the analyst needs to understand, in detail, the domain in which the system is to be installed. As an example of the subtlety of the problems that arise in practice, consider the library item hierarchy. It would seem that the attribute Title could be held in the most general item, then inherited by lower-level items. However, while everything in a library must have some kind of identifier or registration number, it does not

M.Sc. Semester – III (Computer Science) 40

Paper – II (Software Engineering)

follow that everything must have a title. For example, a library may hold the personal papers of a retired politician. Many of these items, such as letters, may not be explicitly titled. These will be classified using some other class (not shown here) that has a different set of attributes.

Figure 2.20 and Figure 2.21 show class inheritance hierarchies where every object class inherits its attributes and operations from a single parent class. Multiple inheritance models may also be constructed where a class has several parents. Its inherited attributes and services are a conjunction of those inherited from each super-class. Figure 2.22 shows an example of a multiple inheritance model that may also be part of the library model. The main problem with multiple inheritance is designing an inheritance graph where objects do not inherit unnecessary attributes. Other problems include the difficulty of reorganising the inheritance graph when changes are required and resolving name clashes where attributes of two or more super-classes have the same name but different meanings. At the system modelling level, such clashes are relatively easy to resolve by manually altering the object model. They cause more problems in object-oriented programming.

**3.4.2 Object aggregation**

As well as acquiring attributes and services through an inheritance relationship with other objects, some objects are groupings of other objects. That is, an object is an aggregate of a set of other objects.



**Fig. 2.23 Multiple inheritance**

M.Sc. Semester – III (Computer Science) 41

Paper – II (Software Engineering)



**Fig. 2.24 Aggregate object representing a course**

The classes representing these objects may be modelled using an object aggregation model, as shown in Figure 2.23. In this example, have modelled a library item, which is a study pack for a university course. This study pack includes lecture notes, exercises, sample solutions, copies of transparencies used in lectures, and videotapes. The UML notation for aggregation is to represent the composition by including a diamond shape on the source of the link. Therefore, Figure 2.23 can be read as ‘A study pack is composed of one of more assignments, OHP slide packages, lecture notes and videotapes.’

**3.4.3 Object behaviour modeling**

To model the behaviour of objects, you have to show how the operations provided by the objects are used. In the UML, you model behaviours using scenarios that are represented as UML use-cases. One way to model behaviour is to use UML sequence diagrams that show the sequence of actions involved in a use-case. As well as sequence diagrams, the UML also includes collaboration diagrams that show the sequence of messages exchanged by objects. These are similar to sequence diagrams so I do not cover them here. You can see how sequence diagrams can be used for behaviour modelling in Figure 2.24 that expands a use-case from the LIBSYS system where users with draw items from the library in electronic form.

For example, imagine a situation where the study packs shown in Figure 2.23 could be maintained electronically and downloaded to the student’s computer. In a sequence diagram, objects and actors are aligned along the top of the diagram. Labelled arrows indicate operations; the sequence of operations is from top to bottom. In this scenario, the library user accesses the

M.Sc. Semester – III (Computer Science) 42

Paper – II (Software Engineering)

catalogue to see whether the item required is available electronically; if it is, the user requests the electronic issue of that item. For copyright reasons, this must be licensed so there is a transaction between the item and the user where the license is agreed. The item to be issued is then sent to a network server object for compression before being sent to the library user. You can find another example of a sequence diagram that expands a LIBSYS use-case in Figure 2.8, which shows the sequence of actions involved in printing an article.



**Fig. 2.25 The issue of electronic items**

**3.5 Structured methods**

A structured method is a systematic way of producing models of an existing system or of a system that is to be built. They were first developed in the 1970s to support software analysis and design and evolved in the 1980s and 1990s to support object oriented development. These object-oriented methods coalesced, with the UML proposed as a standard modelling language and the Unified Process, and later with the Rational Unified Process, as an associated structured method. Budgen summaries and compares several of these structured methods. Structured methods provide a framework for detailed system modelling as part of requirements elicitation and analysis. Most structured methods have their own preferred set of system models. They usually define a process that may be used to derive these models and a set of rules and guidelines that apply to the models. Standard documentation is produced for the system.

CASE tools are usually available for method support. These tools support model editing and code and report generation, and provide some model-checking capabilities. Structured methods have been applied successfully in many large projects. They can deliver significant cost

M.Sc. Semester – III (Computer Science) 43

Paper – II (Software Engineering)

reductions because they use standard notations and ensure that standard design documentation is produced. However, structured methods suffer from a number of weaknesses: 1. They do not provide effective support for understanding or modelling nonfunctional system requirements.

2. They are indiscriminate in that they do not usually include guidelines to help users decide whether a method is appropriate for a particular problem. Nor do they normally include advice on how they may be adapted for use in a particular environment.

3. They often produce too much documentation. The essence of the system requirements may be hidden by the mass of detail that is included.

4. The models that are produced are very detailed, and users often find them difficult to understand.

These users therefore cannot check the realism of these models. In practice, however, requirements engineers and designers don’t restrict themselves to the models proposed in any particular method. For example, object-oriented methods do not usually suggest that data-flow models should be developed. However, in my experience, such models are often useful as part of a requirements analysis process because can present an overall picture of the end-to-end processing in the system. They may also contribute directly to object identification (the data which flows) and the identification of operations on these objects (the transformations). Analysis and design CASE tools support the creation, editing and analysis of the graphical notations used in structured methods. Figure 2.25 shows the components that may be included method support environment.



**Fig. 2.26 The components of a CASE tool for structured method support**

M.Sc. Semester – III (Computer Science) 44

Paper – II (Software Engineering)

Comprehensive method support tools, as illustrated in Figure 2.25, normally include: 1. Diagram editors used to create object models, data models, behavioural models, and so on. These editors are not just drawing tools but are aware of the types of entities in the diagram. They capture information about these entities and save this information in the central repository. 2. Design analysis and checking tools that process the design and report on errors and anomalies. These may be integrated with the editing system so that user errors are trapped at an early stage in the process.

3. Repository query languages that allow the designer to find designs and associated design information in the repository.

4. A data dictionary that maintains information about the entities used in a system design. 5. Report definition and generation tools that take information from the central store and automatically generate system documentation.

6. Forms definition tools that allow screen and document formats to be specified. 7. Import/export facilities that allow the interchange of information from the central repository with other development tools.

8. Code generators that generate code or code skeletons automatically from the design captured in the central store.

Most comprehensive CASE toolsets allow the user to generate a program or a program fragment from the information provided in the system model. CASE tools often support different languages so the user can generate a program in C, C++ or Java from the same design model. Because models exclude low-level details, the code generator in a design workbench cannot usually generate the complete system. Some hand coding is usually necessary to add detail to the generated code.

---------------------------------------------------------------------------------------------------------------------

M.Sc. Semester – III (Computer Science) 45