UNIT 1 WEB SOFTWARE ENGINEERING

Structure **Page Nos.**

1.0 INTRODUCTION

Today's Internet age has brought about a large demand for services that can be delivered over the network. These services rely in large measure on web software applications that can be accessed concurrently by many users. Increasing competition has meant that vendors want shorter and shorter cycle times for producing new products in any sphere, but in the arena of web based services, the situation is extreme. Instead of years or even months, new applications have to be readied and deployed in weeks, before someone else does so. Such pressures are compounded by the need to have high quality software as no one will visit a buggy website for the second time.

Besides technical challenges of architecture and design, web software brings in difficulties caused by the peculiarly distributed nature of the whole endeavour. Not only is the architecture such that the different tiers could be working out of potentially different machines located widely apart, the users are likely to be geographically separated by vast distances and could come from different cultural backgrounds. And, as likely as not, the development team could be spread out in space and these greatly separated team members being responsible for the software produced.

Web software could be –

- Meant for the Internet, with users coming from the public in a geographical region that could perhaps be the whole world.
- Meant for a closed group such as a corporation, in which case, it would be running over an Intranet.
- Meant for an extended, but still closed groups such as corporations, its customers and suppliers, where upon it is an Extranet.

Irrespective of the type of usage intended, the users could be separated by large distances and the application needs to behave correctly for all of them. In this unit, we will concentrate on web applications meant for the Internet.

We will also explore the difficulties associated with developing and managing a web based software project. Also examined will be some approaches and general rules

that could help us to do such a project well. Please be aware that the subject is vast, and in this short unit we can only touch upon the complexities of the topic.

1.1 OBJECTIVES

After going through this unit, you should be able to:

- identify the characteristics of a web based software application;
- describe the architecture of such an application;
- identify the challenges of managing a web based software project;
- think up relevant metrics to gauge the health of such a project;
- ensure proper understanding of the requirements in such a project so that development can be pursued, and
- facilitate quality control of a web based project.

1.2 DIFFERENT CHARACTERISTICS

While web based software applications can vary a lot in size and complexity, all of them share common characteristics, some of which are peculiar to them and are not prominent in other applications. We can look at these characteristics in terms of–

- The way the applications are developed, tested and deployed
- The way the applications are used
- The way they are maintained
- The design of these applications.

1.2.1 Characteristics of a Web Application

In this section, we will look at the overall architecture and deployment context of web based applications and see how this leads to their peculiar management challenges. Unlike conventional applications that can be monolithic, web applications are by their very nature amenable to layering. One cannot have a monolithic web application as the client and the rest of the application have to be of necessity separated. In principle, one could have a thick client application with a lot of intelligence embedded at the client end, but that would defeat the very purpose of delivering an application over the web. The idea is to have a client that is common to all users so that anybody can access the application without having to do anything special. This client is typically a browser of some sort that can render a HTML page on the user's screen.

Most of the processing is done at the other end of the application, that is, at the server. Here again there can be separation between the application and the data storage in the database. These two layers can be on separate machines that are themselves separated over the network.

The layering approach can bring about much greater flexibility and simplicity in design, maintenance and usage. In a monolithic application, everything is coupled together and in a layered application, we can change one layer without affecting the behaviour of others. For example, the business logic at the application layer can be changed without affecting the user interface or the database design. We can change the database management system from one vendor's offering to another without changing the application code or the user interface.

There can be many kinds of web applications such as –

- Those with static text
- Content that is changed frequently
- Interactive websites that act on user input

- Portals that are merely gateways to different kinds of websites
- Commercial sites that allow transactions
- Those that allow searches.

1.2.2 Development, Testing and Deployment

Although many basic processes and methodologies do not change, there are some features of the development cycle of web applications that are different. While some of these will be looked at in more detail later in the unit, we need to understand these features to be able to effectively develop a web application project.

Development

The development of such an application has to take into account the fact that there is now one more element in the whole behaviour of the application – the network. The element of communication could be ignored in conventional software, but it plays a pivotal role in web based projects. Different users can be using networks that are slow or fast and everything in between. Networks are not fully reliable and communication lines do drop or fluctuate in their speed. While network protocols will take care of the issues of data reliability, this comes at the cost of uniform speed. Designers of web applications need to take care of situations like race conditions, lost connections, timeouts and all the other tiresome realities of wide area networks.

Web applications, if used over the Internet, can potentially have a very large number of users. So we need to be careful about matters like machine configurations and sizing. It is also frequently necessary to have a scalable architecture that can be upgraded as application loads rise with the growing popularity of the site. And if the site becomes really popular, this need can be very frequent indeed!

Testing

The testing of web applications has its own difficulties, caused mainly by the variable speed with which inputs reach the server and responses reach the user over the World Wide Web. It is hard to simulate real Internet like conditions in test environments that are typically based on a local area network.

Testing tools do exist that allow inputs to be sent from machines located in different parts of the world to the application. While comparatively expensive, they do allow for realistic testing of the applications. There are other ways of testing the applications by trying to simulate delays and other conditions of the real Internet. To be able to obtain best, average and worst case response times, it is necessary to run tests many times. Automated testing is important for such load, performance and stress tests. They cannot be realistically performed manually.

Although interoperability and consistency are assumed to be the good features of standardized browsers and databases, in practice there are several problems that can arise because of the differing behaviour of browsers caused by–

- Different operating systems such as Linux, Windows versions or Unix systems.
- Different versions of the operating systems.
- Differences in other components such as the Java Virtual Machine.
- Differences in behaviour of the browsers themselves because of extensions or incomplete adherence to standards.
- Bugs in certain versions of the browsers on certain operating systems.

It is often impractical to test the application on all possible combinations of browsers, operating systems, databases and their various versions. In such cases the combinations for which tests have been performed are sometimes indicated and the

appropriate fixes have to be made to the application as bugs are discovered and reported during field use.

Deployment

In most conventional application rollouts, there is often a fair degree of control available to the developers because of the possibility of limiting access. Limited releases to a select test group can be made so that the most obvious and common errors and annoyances are taken care of before the regular release. But in a web application that works over the Internet, it can be difficult to do such tests. True, access can be limited by not publicising the URL at which the application is available, or by having access control in some form that is known only to the select band of users in the test group. Still, since the application is to be available publicly, there is a conflict between the small group which gets early access and the possibility of realistic testing. The very characteristic of a web application is having a large body of concurrent users.

We have already mentioned the need for scalability in the hardware. This is particularly important in terms of storage. We need to be careful that we do not quickly run out of space, for example, in an e-mail or storage site. Besides, it might be important to have the capability to beef up the bandwidth available to the server that is hosting the application, lest it get swamped by user requests.

1.2.3 Usage of Web Applications

Applications hosted on the web are meant to be used by lay users who may have little knowledge of computers. Conventional applications also have to deal with usage by non- technical people, but there we can often organise training or help desks to take care of any problems that may arise. That possibility is not available for a web based application. Because, we cannot reach and train all of our target audience, even if it were possible to identify them individually. As any developer with a little experience knows, users can "exercise" software in ways that were never anticipated.

Normally, if a user comes across difficulties while using the application, she could be expected to report the problem to the appropriate group in the organisation or to the vendor responsible for maintaining and deploying the software. But in the case of an application on the web, the user may simply give up and just not come back to the site if she is not able to work the page, and that too in the first few seconds. Or, her network connection might break while trying to use the site, and that could well be her last visit. So, the tolerance level of users is exceptionally low in such applications.

Users can also be expected to want a fast response and if some operation is likely to take more than a few seconds, it will be best to keep them informed. Feedback on the state of a request is also necessary, as also instructions on what to do if they are disconnected in the middle of a long operation.

An application on the web is much more likely to be internationalised as it might have to cater to the needs of users from different parts of the world. You may be shutting out a large number of potential visitors to your application if you restrict yourself to any particular language. You should also expect to see little data entry and as much as possible of features such as auto completion, drop down menus, look up tables and the like.

1.2.4 Maintaining Web Applications

The maintenance of web applications has its own set of challenges. Frequent changes to the presentation, or look and feel are expected. Many of the static text pages may need updating ever so often and some of the content could be changing daily or hourly. So the task of maintenance has to expand to take care of changing content and this can be a major exercise in some sites. Business rules could also be volatile, with new schemes and packages being frequently invented to entice the users. It means that the maintainers have to take care of frequently changing the business logic with little change to the other layers.

People are sensitive about sites that are perceived as not trendy enough and expectations are high as regards features, good looks and easily usable interfaces. One might have to work hard at changing and upgrading these to keep up with the latest advances. If a site is not changed for more than a few months, it will not be looked upon as live or up to date. So, taking care of a web application is often much more than just fixing bugs or updating the software to implement changes to business rules. There is a fair amount of effort that has to be put in to improve usability.

1.2.5 Designing Web Applications

Since web applications are layered, we get a simplified design that can be easily maintained and altered. However, that also produces the corresponding need to design the different components to be generalised enough so that they can be useful to a larger audience. Applications could be running under the control of an application server for which the designers have to create and implement the requisite beans. For some purposes, beans can be purchased from other software vendors or they could have to be written from scratch for the application.

These days, some functionality in the application could be coming out of web services, components on the web that can run with your application and that provide defined behavior. An application could look for an available web service that can do some needed action. Web services advertise their available functionality that needy applications can tap. This distributed nature of the application greatly increases the amount of reusability, but brings about problems of security and reliability.

The user interface in a web application is very important indeed and can often be the tipping point in the success of a website. Ease of use and ergonomic considerations are important. The application has to be able to cater to users of disparate skill levels and in some countries may be required to provide access to people with disabilities.

All the above characteristics need to be kept in mind when planning and otherwise managing a web based application project.

Check Your Progress 1

1) The _____ can bring about much greater flexibility and simplicity in design, maintenance and usage of web applications.

2) One of the significant characteristics of a Web Application is to have a large number of _____ users.

1.3 ISSUES OF MANAGEMENT OF WEB BASED PROJECTS

Like any other software application project, we need to use good software development practices when faced with working on a web application. Otherwise the project would not remain in control and we would face problems with timeliness, budgets and quality. Before going on to the special issues, it would be useful to review the general good practices that we need to follow.

1.3.1 Review of Good Software Development Practices

There is by now a wealth of literature and experience on how to manage software projects. While two decades ago the software crisis was a big stumbling block and failed or overdue projects were commonplace, the efforts of quality gurus have resulted in much better techniques of management that address these problems. Many projects still have trouble, but that is not because the techniques were not known but because they were not followed.

There are several quality models available to organisations that can be followed, such as the ISO 9001:2000 quality system model or the Capability Maturity Model Integration (CMMI) of the Software Engineering Institute of the Carnegie Mellon

University, Pittsburgh, USA. Let us look now at some of the basic, common principles of software management practice.

Managing Requirements

It is very important in any project to have a clear understanding of the requirements shared with the customer. Very often the developers have no notion of how the software will be used or what would be nice for the user to have. While a requirements document might be prepared and approved by the customer, there are many issues that are hard to capture in any document, despite having detailed discussion with customer. To ensure that the viewpoints of the developers and customer do not diverge, regular communication is a must. Finer points that could not be captured in the documentation have to be understood by the project team through proper coordination with the customer. Here the customer could be an external party or could be another group within the same organisation, in which case it is an internal customer.

Even if we achieve a good understanding of the requirements, software projects are notorious for changes. This means that we must have a good system of tracking requirements as they change. The rest of the artifacts in the project must reflect the changes that have occurred in the requirements.

Managing the Project

The essence of managing the project is proper planning for the project and then executing the project according to the plan. If the project deviates from the plan, we need to take corrective action or change the plan. We must always be proactive and keep looking for the risks in the project, rather than react to problems after they have arisen. We should try to anticipate what can go wrong and take action accordingly. Here a plan needs to keep something apart from the schedule of the project.

The project plan is the basic document used to execute the project and has all the required information about it. It should be the guiding document for the project and hence must be kept up to date. Some of the items of information in the plan could be–

- Background information and context
- Customer and other stakeholder information
- Estimates of cost, effort and duration
- Dependencies on outside groups
- Resource requirements – equipment and people
- Methodology to be used to do the project
- How the project will be monitored.
- Schedule.

Besides the project management plan, there are various subordinate plans that need to be made for different aspects of the project, such as,

- Configuration Management Plan
- Quality Assurance Plan that covers audits and process compliance
- Quality Control Plan that covers technical reviews and testing
- Risk Management Plan
- Training Plan
- Metrics Plan
- Defect Prevention Plan

There should be regular communication between the team members themselves to ensure that there is a shared vision and sense of purpose, and that perceptions remain aligned to the project goals.

As the project is being executed, there has to be regular monitoring of the progress and of the different parameters of the project, such as effort, cost and quality. If any of these parameters is not in line with what was planned, appropriate investigation, analysis and corrective action needs to be taken. Sometimes circumstances change such that it is no longer possible to bring back the project on track with the original plan, whereupon the plan needs to be reviewed and revised.

Configuration Management

This is vital to retaining control of the artifacts produced by the project. Given that changes will occur, we need to be able to track and know at all times which version of an artifact is current and which ones are obsolete. We do not discard obsolete versions because it may sometimes be necessary to backtrack or to branch off in another direction from a common base.

Whenever an artifact is produced it should be reviewed so that we can have confidence in the veracity and appropriateness of its content. After this it should be placed in a baseline area and any further changes to it should be done only according to formal change procedures. We have to plan for which artifacts will be so controlled.

Access to baselined artifacts should be limited to a designated configuration controller. Any member of the team who needs to alter a baselined artifact needs to approach the configuration controller who will then check out the artifact and allow the team member to make the change. Simultaneous checking out by more than one team member is discouraged, but if done, any conflicting changes that may be made by them need to be taken care of.

Measurements

It is important to measure our software work so that we can keep control of the proceedings. This is the key to being able to manage the project. Without measurements we will not be able to objectively assess the state of the progress and other parameters of the project, and we would be reduced to relying on our intuition and feel for where the project stands.

Software engineers have been notorious for not bothering to measure their work, or for insisting that their work cannot be measured. It is an engineering discipline unique in that measurements have not been given. However, in the 1990s, great progress was made in spreading awareness and appreciation of the need for software measurements. Still, even today, many organisations do not have a good metrics program going.

Risk Management

An important constituent of project management is managing the risks of the project. A risk is an event that can have a deleterious impact on the project, but which may or may not occur. Thinking of the possible risks is the first step in trying to take care of them. While we cannot get rid of risks, we need to keep thinking of ways to reduce their effects. Better still, we should try to take steps that will prevent the risk from becoming a reality.

Not all risks need to be looked at. When we analyse risks, there are some that are more probable and also have a significant adverse effect on the project. Others may have a severe effect if they occur, but are not too likely to happen. Still others may not be too harmful, even if they do come to pass. There can be a large number of risks in a project, but thinking about all of them is not practical. We need to concentrate on the most important risks and direct our energies to managing them.

Over time, the risk parameters can change. We, therefore need to keep re-looking at our risks and alter our risk plan to take cognizance of the changed situation as the project progresses. We should keep carrying out the actions that make the risk less likely. Whenever a risk does occur, we should put into action our plan for reducing its harmful effects.

Thus risk management allows us to look at a project actively, rather than passively reacting to events after they have occurred.

1.3.2 Organisation of Web Application Teams

Web applications tend to need much more ongoing support, maintenance and enhancement than others. After the initial application has been rolled out, comes the stage of maintenance. This can amount to weekly or even daily releases in the first instance while the software stabilises. This endeavour requires a team structure that is a bit different from others. We start from the maintenance end to emphasise its importance, and the fact that design and development are done comparatively quickly. Sometimes the need for project management during development itself is questioned but, as in any other project, good management is critical to success in a web application project as well.

Webmaster

This role is not unique to web applications, but is usually otherwise referred to as the administrator. It entails taking care of the site on a day to day basis and keeping it in good health. It involves close interaction with the support team to ensure that problems are resolved and the appropriate changes made. Some of her activities include –

- Gathering user feedback, both on bugs (such as broken links) and also on suggestions and questions from the public. These need to be passed on to the support team who are engaged in adapting the site in tune with this information.
- Ensuring proper access control and security for the site. This could include authentication, taking care of the machines that house the server and so on.
- Obtaining statistics and other usage metrics on the site. This could include, among many others –
    - Number of hits, distributed by IP address
    - Number of registered visitors
    - Frequency distribution of the different pages visited
    - Bandwidth utilised for upload and download
    - Number and type of errors, particularly of service denials.
- Helping to ensure that change control procedures are followed.
- Archiving old content while keeping newer content in a more prominent and easy to find location.

Application Support Team

In conventional software projects, while maintenance is important, it may or may not be done by the organisation that developed the project. In web applications, the organisation that developed the site is quite likely to be given the responsibility of its maintenance. This is because web applications tend to keep evolving and what corresponds to the development phase in a conventional application is here quite brief and frenetic. A large part of the evolution of the software tends to happen subsequently, based on user feedback and continuing refinement of the concept by those who conceived of the application. The activities here can consist of–

- Removing bugs and taking care of cosmetic irritants.
- Changing the user interface from time to time for novelty, to accommodate user feedback and to make it more contemporary.
- Altering the business rules of the application as required.
- Introducing new features and schemes, while disabling or removing older ones.

The support team could consist of designers including graphic artists, programmers, database specialists and testers.

Content Development Team

In conventional business application software, there are usually changes and modifications required to master tables in databases, such as altering price lists, available items and so forth. Web applications frequently require much more ongoing, sustained effort to retain user interest because of the need to change the content in the site. In the case of a news site, this updation can be every few minutes. The actual news could be coming from a wire feed from some news agencies, or from the organisation's own sources.

Though not all sites need to be so current, it is often required to keep adding articles and papers of interest to the user community. These stories or articles are written by a team that has expertise in the domain concerned. A site could be serving out a wide variety of information and other stories from different interest areas. Content development teams could be researchers who seek out useful or interesting facts, authors, experts in different areas, and so on. The content may be edited before being served out.

The members of this team are usually not knowledgeable from the software point of view, though they might be well respected in their own domains. They may be full time members of the organisation or casual, freelance contributors. So though called a team, some of them might really be individual providers. The content they produce is typically in the form of a word processor file that might have images, graphs, tables and other non-text ingredients.

Other forms of content are now quite commonplace, such as audio files, slide presentations or video clippings of events, places or speakers. With increasing availability of high speed network connections for even individual users, such content is now becoming increasingly popular because of the impact it can produce.

Web Publisher

This is an important role that connects the content development team to the actual website. The raw material created by the writers has to be converted into a form that is suitable for serving out of a webserver. It means formatting the content according to the requirements of a markup language such as HTML. Though tools can help perform much of this, there still might be human effort that is needed.

In the case of automated news feeds, such publishing needs to be as tool driven as possible. The web publisher must have a good understanding of the technical aspects of web servers and the markup language.

1.3.3 Development and Maintenance Issues

Let us now look at some of the management issues that we will face while developing and maintaining a web application. We have already seen that web applications tend to evolve, and there is little distinction between what in a more conventional project we would refer to as development or maintenance. It is hard to say when the development is complete and maintenance starts in a web project. In practice it is a continual cycle of delivering something and then of fixing and correcting and enhancing it.

Configuration Management

During maintenance of a web application, because of the frequency of changes, configuration management assumes considerable importance. The change control process must be up to the demands placed on it of keeping control over the software configuration inspite of the many changes that will happen. So it must not be overly elaborate or hard to follow. That would mean it might not get followed and the consequences will be disturbing. Once one loses track of what is happening in the software it can become very difficult to get back. It will then become impossible to

predict the effect of changes to the software and one would have to grapple with mysterious bugs whose cause will be hard to find.

Besides the software itself, content also needs to be subjected to configuration management. This could include items such as information on schemes or other service or product offerings. Lack of discipline here could mean horrors like –

- Postings on past offerings that are no longer available,
- Incorrect prices or other terms being visible to users,
- Postings of content that is still work in progress and was not ready for publication apart from lesser embarrassments such as archival content reappearing on the site.

A very important aspect is that of inadequately tested code finding its way into the active configuration. It could mean that visitors to the site have to put up with features that do not work as expected or simply crash on them. Worse, it can result in security holes that compromise the security and integrity of the site. Sometimes, the privacy of data of registered users could be in question. In other cases it could expose other systems of the organisation itself to unauthorised intruders.

Since contributions to an organisation's website can come from diverse sources and departments, it can become difficult to establish accountability and responsibility for the web application. In conventional business software applications it is clearly a particular division or group in the company that is served and that would have an interest in the software working properly. This interest is often diffused and spread out in a web application. It cannot be the technical team or the software wing, as this is often an outsourced vendor. In any case the ownership of the application needs to be assumed by the business and not a technical group. While there can be many claimants for ownership and authority, the same may not hold when it comes to owning responsibility.

To determine who is responsible for the website, some of the relevant questions can be –

- Who pays for the website (this can be a charge to an internal budget)?
- Who is concerned with the accuracy of the content that is published?
- Who gives the requirements to the development or maintenance team?
- Who ensures quality control checks and does the acceptance check?

It is this owner of the website who should be ensuring that proper configuration management practices are followed. Even if this question is resolved, we have to grapple with how exactly to do configuration management. The identification of configuration items can be difficult. This is because there are many components to a web application, and some items such as news content can be very short-lived. Other content can last longer. To what extent does one look at configurations for transient content? Perhaps just recording when it was published and when it was removed is sufficient. For other content that may appear for days or weeks, we might have to deal with normal, conventional changes and updates. What constitutes a configuration unit is also not easy to define, as hyperlinks from within content can complicate the matter.

Conventional configuration management tools are also not really suited to deal with this kind of constant evolution. The usage of these tools is also made difficult because many in the maintenance team, such as content developers, may not be familiar with software, let alone the tools. Also, web application support teams may not be geographically located in the same place and people could be working in different places. So the tools used need to be such that they can be operated over the network.

Project Management

If configuration management has its challenges as described above, they pale into insignificance compared to the difficulties we may have while managing the web

application project itself. When we look at the elements of project management the reasons will become apparent.

1. We have already seen that evolution and hazy, fluid requirement specifications are the norm here. How then does one perform estimation? That presupposes a clear knowledge of the scope and specifications.
2. More than in any other kind of software, schedules are dictated by the users. Quite often, the software is the means to seize upon a business opportunity that may be available for a small window in time. So the developers or maintainers may not have much influence on the time they can get. Whatever the estimate, the software has to be delivered by the date the user needs it.
3. Coordination and human issues can be more daunting than ever. The team may be geographically dispersed, may come from disparate backgrounds and may have conflicting interests or vision.
4. The question of ownership we have already seen in the section on configuration management. Even though stakeholders may not be hard to identify, overall accountability and responsibility could be.
5. The metrics to use to gauge success have to be identified carefully, keeping in mind the unique characteristics of a web application.
6. The actual engineering aspects, particularly analysis, design and testing have to be tuned to the life cycle of such a project. Agile programming methodologies may be of help here.
7. Quality assurance is made more difficult by the frenzied pace of activity that may induce the harried project team to jettison process driven working.

While none of these challenges permit of simple solutions, we can try to keep some things in mind and work towards keeping the project under control by –

- Recognising that the software will follow an iterative life cycle model for development, with each iteration being quite short. We may not be able to work out a grand plan for the whole project as it will be at the end, simply because we cannot predict what shape it will take eventually. So we could concentrate on the current iteration and be clear about what we will deliver in that one. Other items on the user wish list should be taken care of in the next delivery. Save in exceptional circumstances, do not alter the objectives or scope of the current iteration.
- The question of schedule is a difficult one. Managing expectations of the users is imperative here. If what the user's desire is clearly unattainable, we need to negotiate and reduce the scope for the current iteration. Also with more experience in getting a web application developed, the users will hopefully have a better appreciation of what is possible.
- With current day groupware and other tools, we can make it simpler to work with distributed teams. But there should be adequate attention paid to communication among team members so that the objectives of the project are clear. As in a conventional project, we should have team meetings regularly, even if those meetings are done over the telephone or using network meeting tools.
- As always, the processes to be followed in the project should be appropriate to the situation. They should not be cumbersome or difficult to do, so that the project team will be able to follow them. We need to remember that if the processes are complex and are not followed, the impact on the project is much worse, as we potentially could lose control of what is happening.

In subsequent sections we will be reviewing the strategies to follow to get a grip on the other issues such as life cycle activities and measurements to be made. The other, usual elements of project management such as managing risks, analysing defects and

trying to prevent them or monitoring project parameters such as cost, effort, schedule and quality, remain as valid as in any other software project.

Check Your Progress 2

1) The essence of managing the project is proper _____ for the project and then executing the project according to the plan.

2) The _____ process must be up to the demands placed on it of keeping control over the software configuration in spite of the many changes that will happen.

1.4 METRICS

Management of any kind is difficult, if not impractical, without measurements being made. Those values tell us where we stand in our endeavour, whatever it may be. Software engineers have been especially tardy in understanding the need for measurement, preferring to rely on intuition and subjective assessment. However, realisation does seem to have set in and many software projects do measure at least the basic parameters like schedule, effort and defects. When it comes to cost, practices seem to vary across organisations. Some are secretive about the costs of projects and only senior management is privy to such information. Others place the responsibility for the budget for the project on the shoulders of the project manager.

In web applications, these core metrics need to be measured and analysed just as in any other software application. Schedules could be made fine grained so that we are tracking things on small, quick milestones rather than only at the "end" of the project, which is difficult to define here. That could mean that it may not be very meaningful to measure them in terms of slippages (a one-day delay in a task lasting a week is 14%). One possibility could be to look at the percentage of schedules that could not be kept.

Effort metrics can be gathered through a time sheet and we can measure the total effort in different kinds of activities to build up our metrics database for future guidance. However, here we need to be sure to record the actual effort and not try to confine it to a conventional workday of 8 hours. Metrics data on effort is valuable in building up a body of knowledge for future estimation. This data should be recorded under time spent on analysis, design, construction, project management and so forth. Past data is also helpful in negotiating with users for a reasonable amount of time to perform a task

Another core attribute of a software project is its size. This should preferably be measured in an implementation independent way, such as function points. The International Function Point Users Group (IFPUG) has come up with guidelines for measuring the size of a web application from the user point of view. If the organisation has good reason to feel that this measure is not suitable, then the creation of one's own size measure could be considered. If a good size measure can be built, then it can be used to estimate the effort required for future projects, provided the productivity factor that converts the size to effort is available from data on past projects.

The quality of the application can be measured in terms of defects that are found during internal testing or reviews as well as external defects that are reported by visitors who come to the site. But all defects are not equal, and we need to analyse defect data carefully. Some attributes that can be used to classify defects for further analysis are:

- Severity on a scale of 3 levels, such as high, medium and low. More levels up to 5 can also be considered. The severity is in terms of impact on the usage of the software by the user.

- Phase in the life cycle where they were introduced, such as analysis, design or coding.
- Whether they are internal defects, detected by the development or testing team, or whether they were found by the end users.
- Whether they were detected during reviews or during testing.
- Effort required to fix the defect, as this might not have any correlation to the perceived severity of the defect.

Such an analysis would then form the basis for the formulation of a defect prevention plan for the software project. We would strive to prevent the kind of defects that are of high severity or those that take a lot of effort to remedy.

We can also look at other measures like –

- Percentage human resource utilisation in the project. Roles that are idle for significant periods could be allocated to another project as well.
- Percentage of code that is reused from the organisation's library of such components.
- Quantum of changes to the user requirements as the project progresses. An overly high value could indicate that more effort needs to be directed towards scoping out the project and eliciting the requirements.
- Percentage of attrition in the project. An unusually high value for this could point towards problems in managing the human angle.

Besides metrics to give insight into the development aspects of the project and the general state of the team, there can be a large number of statistics and measures that can be gathered about the usage of the website once it has been put into active use. Some of these have been already touched upon in earlier section under the role of Webmaster. Most of them rely on traffic analysis to draw conclusions about the users and their interests. This can give an insight into what changes should be made to the site or how it can be improved.

We can also look at some metrics from the point of view of maintenance. These could be the average time taken to fix bugs of different severity levels, together with the best and worst case times. Other measures can include –

- User satisfaction index
- Mean time between failures (bugs detected or reported)
- Number and percentage of repeat visitors.

Gathering and calculating metrics is only one part of the work involved here. The crux of the matter is how well the knowledge of metrics is used to shape the functionality and features of the project. The metrics should be aligned to the goals and objectives of the project and the lower level measures should be derived from the top level ones, so that we measure whatever is important for us and do not gather extraneous data. From a measurement deficient situation it is quite possible to go overboard and drown oneself in a mass of low level data that is not meaningful or useful.

1.5 ANALYSIS

The usual principles of analysis continue to apply to a web software application as well. So the requirements have to be, as always, elicited, represented and validated. These are then the basis for further work in the project. Let us look at some of the characteristics that we need to bear in mind while analysing a web application project.

It is particularly important to be clear about the process owner and the stakeholders here. We must not end up getting wish lists and requirements from someone who, it later turns out, is not the one authorised to provide them. While always a point of concern, the issues of ownership of the web application accentuate the problem. From the scope that would have been talked about while setting up the project, the detailed system analysis would need to be done to come up with the requirements.

Besides the normal considerations of what the application has to do, we need to look at how the user will interact with the application even more closely than in a conventional application for the reasons we have already looked at in the beginning of this unit. The user interface has to be simple to use, visually appealing and forgiving of a lay visitor who may not be familiar with computers or software. The actual functionality of the application is something that has to be ensured the same way as it would be for any other software, being the very reason for going to all the trouble. Where things are different is in the importance of content to a web application. We have to work out what will be shown to the user, such as a product brochure, an offering for a tour package or anything else. This has to be placed before the user in a manner that will be eye catching but not garish, and the user has to be able to obtain some benefit from the offering without having to do anything elaborate.

More than in conventional applications, it is hard to distinguish entirely between analysis and design in a web application. The kind of actual content that has to be shown needs to be studied to decide on the best way of displaying it on the site. Whether the user will like to show it as text, graphics, audio or video is something that will depend on the resources available for hosting the site as well as on considerations that have to do with the resources available with the group of people that can be expected to visit the site. It is here that the study of the target audience becomes crucial. This should be done on considerations that have to do with –

- Geographical region,
- Language, beliefs and culture,
- Social strata,
- Age group and gender.

and other similar characteristics. All of these demand that we change our design to appeal to our typical visitor.

1.6 DESIGN AND CONSTRUCTION

In designing the functional aspects of a web application, we work much as we would for a conventional application. Because of the compressed time scales and volatility, we should consciously look to reusing as many things as possible. Reuse is always desirable but becomes imperative in the case of a web application. While code and object libraries come first to mind when we think of reuse, there are elements that can be reused for the design as well. We should, for instance, be seeking out any available design patterns that can be reused.

The advent of application servers and other component-based technologies has facilitated the task. The application server gives the framework under which the application can run and takes care of many mundane details that would otherwise be the burden of the programmer. Security and persistence are two such tasks. The functionality of the application is provided by stringing together beans, many of which may have been already written or can be purchased from third parties. There are also other architectural models from other vendors to help write such applications.

Where we have to lay great emphasis is on the user interface. There are several aspects to creating a good one, some of which are:

1. **Visual appeal and aesthetics:** This is the domain of a graphic artist. Things such as cute sound effects, blinking and moving images that can appear interesting or attractive for the first couple of times can irritate after a while. So a pleasant, clean visual design is perhaps a safe bet to hold continuing interest on repeat visits. However, if there is ever a choice between graphical appeal and functionality, we must plumb for functionality every time.
2. **Ease of entry of data:** Given the fact that most users may be unfamiliar with computers or software, data entry should be as simple as can be made. Actual entry using the keyboard should be kept to the minimum. Allow for using the mouse for selection from lists. It should be easy to ask for information of the application.
3. **Ease of obtaining the application's response:** Whenever we query the application, the results should be easy to see and interpret. If the processing is going to take a while, give visual feedback to the user on how long she can expect to wait. At no point should a user feel lost, not knowing where s/he is in the site or what she should do next. Context sensitive help should be available throughout the site.
4. **Intuitive, easy navigation around the site:** As a visitor navigates around the site, there should be some aids to helping her find her bearings. Things like a site map and your location on it are useful. It should be possible to come back to a reference point like the home page in a single click from anywhere deep in the site. It is our application that should provide all navigation facilities and we must not expect the user to use browser buttons to go around.
5. **Robust and forgiving of errors:** If there is any erroneous input, the consequences should not be too inconvenient. We should expect a discreet and graceful error message, not a disconnection from the server. It should not happen that we land up in some error page that does not give us any explanation of what went wrong.

Website users would not like to have to scroll and so we must try to see that this is not required for any screen resolution. A horizontal scroll is much more irritating than a vertical scroll to most users.

1.7 REVIEWS AND TESTING

In this section we will look at some of the considerations for reviewing testing web applications. The content of a website is the item of interest to the visitor and so we must take care to review all content to make sure that there are no errors of fact or language therein. Besides, the design of the site should be reviewed to catch errors like difficulties in navigation or data entry.

It may not be very useful to merely check out the components of a website on a standalone basis as part of unit testing. While in a conventional application a program is the smallest unit that we test, in web applications it is more likely to be a single web page. We should check out the content, the navigation and the links to other pages as part of testing out a page. There should not be any broken links that leave the user bemused. If there is some functionality being provided by that page, that part should work correctly. This part of the software is close to a conventional application and the actual processing that happens can be unit tested as usual. Where the processing is done by using bought out components, it may not be necessary to unit test those components. We would then only have to concentrate on checking out the full functionality.

Once all the pages of the web application are ready we can perform integration testing. We check out how the pages work together. Whenever an error is caught and fixed we should repeat the tests to ensure that solving a problem has not caused

another problem elsewhere. It is possible that individual pages work well but not together.

Unlike a conventional application, here we do not have much control (except our instructions to users) on all parts of the working system. This is because we do not know what browser, what version of it and what operating system the user will be working with. While it may not be practical to test out with all possible combinations, we should ensure that the application works with most sets of browsers and operating systems. We should also see that the software works at different screen resolutions. The network bandwidth expected should also be clearly decided and we should test things out at all network speeds up to the lowest.

The actual implementation of the application has to be done carefully. In the deployment environment, the configuration has to be checked out to be in tune with what the application was designed and constructed for. Thereafter we may release the software to a limited test group of users to gather their feedback before making it public.

☛ Check Your Progress 3

1) The _____ gives the framework under which the application can run and takes care of many mundane details that would otherwise be the burden of the programmer.

2) In the _____ environment, the configuration has to be checked out to be in tune with what the application was designed and constructed for.

1.8 SUMMARY

This unit has been a very brief introduction to different aspects of engineering web software applications. While a lot more could have been written about each topic, and other topics could have been added here, we have been able to concentrate on only a few important points here that are with in the scope.

- Web applications have many points in common with conventional software applications, but have their own unique characteristics.
- There are differences in the ways web applications are developed, deployed, tested and maintained.
    - Web applications often do not have a clear dividing line between development and maintenance and usually continue to evolve.
    - They have to be developed and put into use quickly and so try to make use of reusable components as much as possible.
    - The user interface is very important because they are used by lay visitors.
    - Testing them can be difficult
- Following good management practices like project management and configuration management of web applications is not straightforward because of their characteristics.
- We need to think through the metrics we will use to evaluate project progress and product quality of web applications.

1.9 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Layering approach
2) Concurrent

Check Your Progress 2

1) Planning
2) Change control

Check Your Progress 3

1) Application server
2) Deployment

1.10 FURTHER READING

Software Engineering – A Practitioner's Approach, Roger S. Pressman; McGraw-Hill International Edition.

Reference Websites

- http://sei.cmu.edu/cmmi
- http://standards.ieee.org

UNIT 2 MOBILE SOFTWARE ENGINEERING

Structure **Page Nos.**

2.0 INTRODUCTION

This unit introduces learner to the subject of development of mobile applications. With the increased usage of mobile devices across the world, more and more applications are being targeted at them. If an application that is developed for computers can also run on the mobile devices, then the entire paradigm of application usage changes as it enables the user to run application from wherever he is.

There were different standards on which the mobile networks are based. One of the popular standard is GSM. In this unit, we introduce the GSM architecture. There were a number of environments under which mobile applications can be developed. One of the popular environments is Java. We introduce J2ME (Java 2 Micro Edition) in this unit in conjunction with wireless application development with it. Any application should be tested before being put to full-fledged use. In this unit, Java Device Test Suite is introduced with the help of which wireless applications can be tested.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- know the GSM architecture;
- know about wireless application development using J2ME, and
- know about Java Device Test Suite.

2.2 INTRODUCTION TO GSM

Mobile devices are used by a number of people across many countries. In this unit, mobile devices refer to mobile phones. Every country is having a set of standards on which the mobile devices in their country are based upon. GSM is one of the popular architectures on which mobile devices are based on. GSM stands for Global System for Mobile Communications.

GSM is a digital wireless network standard. All mobile devices that are based on GSM standard across the world will have similar capabilities.

The following are some of the features of GSM:

- If a mobile device is based on GSM, then it can be used in all those countries where this particular standard is prevailing.

- Almost all the services that are existent in a wireline network are provided by GSM to all the users of mobile devices which are based on it.
- Though the quality of voice telephony is not excellent, it is not inferior to the systems that are analog based.
- It also provides good security as there is an option to encrypt the information that is being exchanged using this standard.
- There is no need for significant modification of wireline networks due to the establishment of networks based on GSM standard.

Architecture of GSM

There are three different parts in a mobile device. They are SIM card, Mobile equipment and Terminal equipment. SIM stands for Subscriber Identity Module. A mobile device can be loaded by any SIM. The number of the mobile device is associated with the SIM. The size of a SIM is similar to that of a smart card. Every SIM is also having a PIN (Personal Identity Number) associated with it. After loading the SIM into the mobile device, the user is asked to enter the PIN. Only on entering the correct PIN, the user will be able to start using the services offered by the mobile operator. Initially, PIN is loaded by the operator. The PIN can be changed by the user. If the user is unable to enter the SIM correctly for the first time, then s/he is given an opportunity to re-enter it for a fixed number of times. In case, s/he is not able to enter it correctly and exhausted all the tries, then the PIN is blocked. So, that particular SIM cannot be used unless the network operator activates it. The SIM can be unblocked only on entering the correct PUK (PIN Unblocking Key). Information related to subscriber, PIN and PUK codes are present in SIM.

The architecture of GSM is composed of Mobile devices, Base Station Systems (BSS), Mobile Switching Center (MSC) etc. The communication between Mobile device and BSS is through radio interface. BSS communicates with MSC by connecting to Network and Switching Subsystem (NSS). An MSC is a telephone exchange that is specifically configured for mobile applications. Mobile devices and Public Switched Telephone Network (PSTN) interface through Base stations. BSS consists of a Base Transceiver Station (BTS) and Base Station Controller (BSC). Equipment related to transmission, reception and signalling is part of BTS and this equipment is used by it to contact Mobile devices. BSC deals with the allocation of radio channel and its release apart from hand off management. Using ISDN protocols, BSC communicates with BTS.

Apart from GSM, there are other standards such as CDMA etc.

2.3 WIRELESS APPLICATION DEVELOPMENT USING J2ME

Java supports mobile application development using its J2ME. J2ME stands for Java 2 Platform, Micro Edition. J2ME provides an environment under which application development can be done for mobile phone, personal digital assistants and other embedded devices. As like any other Java platform, J2ME includes API's (Application Programming Interface) and Java Virtual Machines. It includes a range of user interfaces, provides security and supports a large number of network protocols. J2ME also supports the concept of *write once, run any where* concept. Initially, an application can be developed using J2ME targeting a specific type of devices. Then, the same application can be used for different types of devices. Also, it is possible to use the native capabilities of these devices. J2ME is one of the popular platforms that is being used across the world for a number of mobile devices, embedded devices etc. There is a huge market for applications that target wireless world. Some of the

wireless applications are Games, Applications that access databases, Location based services etc.

The architecture of J2ME consists of a number of components that can be used to construct a suitable Java Runtime Environment (JRE) for a set of mobile devices. When right components are selected, it will lead to a good memory, processing strength and I/O capabilities for the set of devices for which JRE is being constructed.

Currently, two configurations are supported by J2ME. They are Connected Limited Device Configuration (CLDC) and Connected Device Configuration (CDC). Each configuration consists of a virtual machine and a set of class libraries. Each configuration provides functionality of a particular level to the set of devices that have similar features. One of the features is network connectivity. To provide a full set of functionalities, a profile and an additional set of APIs should be added to the configuration. These additional APIs will enable the usage of native properties of devices as well as define suitable user interface. Each profile will support a smaller set of devices among the devices that are supported by the chosen configuration.

In the case of J2ME, often CLDC is combined with MIDP. MIDP stands for Mobile Information Device Profile. This combination provides a Java based environment for cell phones in which applications can be developed. It is always possible to add optional packages to the configuration that was already selected to support an additional set of services. An optional package consists of a set of standard APIs that help us to use latest technologies such as connecting to databases from cell phones, blue tooth, multimedia etc. It is never mandatory to use optional packages. In fact, it is to the developer to select those optional packages which provide the additional functionality as per his desire.

CDC is the larger of the two configurations supported by J2ME. CLDC is designed for devices that have less memory, slow processor and unstable network connections. CLDC is suitable for cell phones. Such devices have a memory that ranges from 128KB to 512KB. Java has to be implemented within this memory. CDC is designed for devices that have larger memory, faster processors and a significant network bandwidth. TV set top boxes, high range PDAs (Personal Digital Assistant) are examples of the devices that are suitable for CDC. CDC consists of a JVM (Java Virtual Machine) and Java software that includes a significant portion of J2SE (Java 2 Standard Edition) platform. Usually, devices that are suitable CDC at least possess 2MB of memory in which Java software can be loaded.

MIDP is designed for cell phones and other lower level PDAs. User interface, network connectivity, local data storage and application management are offered by MIDP for applications that are based on mobile devices. The combination of MIDP and CLDC provides a full-fledged Java runtime environment for the mobile devices. The environment provided by their combination will add to the features of the mobile devices. This will lead to a better power consumption and efficiency in memory utilisation. The lowest level profile for CDC is known as Foundation Profile (FP). FP provides embedded devices which does not have a user interface with network capability. There are two more profiles, namely, Personal Basis Profile (PBP) and Personal Profile (PP). FP can be combined with these profiles to provide GUI (Graphical User Interface) for the devices that require it. All the profiles of CDC are in a layered manner. This will facilitate adding as well as removing profiles as per the need of features.

PP is useful for all the devices that require a GUI, Internet support etc. Usually, devices such as higher level PDAs, Communicators etc. need such features. Java AWT (Abstract Window Toolkit) is contained in this profile. It also offers web fidelity, applets etc. PBP is a subset of PP. PBP supports application environment to those devices in the network that support at least a basic graphical representation. PP as well as PBP are layered to the top of CDC and FP.

There are several reasons for the usage of Java technology for wireless application development. Some of them are given below:

- Java platform is secure. It is safe. It always works within the boundaries of Java Virtual Machine. Hence, if something goes wrong, then only JVM is corrupted. The device is never damaged.
- Automatic garbage collection is provided by Java. In the absence of automatic garbage collection, it is to the developer to search for the memory leaks.
- Java offers exception handling mechanism. Such a mechanism facilitates the creation of robust applications.
- Java is portable. Suppose that you develop an application using MIDP. This application can be executed on any mobile device that implements MIDP specification. Due to the feature of portability, it is possible to move applications to specific devices over the air.

You can use J2ME wireless toolkit for development of wireless applications using J2ME. It is possible to set up a development environment by using the following software:

- J2SE (Java 2 Standard Edition) SDK (Software Development Kit) of v1.4.2 or upwards.
- J2ME (Java 2 Micro Edition) Wireless toolkit. Using this toolkit, MIDlets can be developed.
- Any Text editor.

MIDlets are programs that use the Mobile Information Device Profile (MIDP).
The first step is to install J2SE SDK. J2ME wireless tool kit runs on the Java platform provided by J2SE. J2SE needs to be installed because it consists of Java compiler as well as other requisite software which is used by J2ME wireless tool kit to build programs.

The second step is to install J2ME wireless toolkit. It is possible to develop MIDP applications using J2ME wireless toolkit. The J2ME wireless toolkit uses the concept of projects rather than files. Once J2ME wireless toolkit is installed and run, we can start creating projects. At the end of the creation of the project, you have on MIDP suite. Once the project is created, its properties can be changed. Also, the project can be build and executed in the device emulator. Let us create a new project. Let the title of the project be project1. Along with the title of the project, MIDlet class name should also be given. Let it be projectlet1.

Once these names are confirmed, the process of creating the project project1 commences. As the result of creation, a directory titled project1 will be created in which the following subdirectories will be present:

- bin
- lib
- res
- src

A compiled MIDlet suite (a .jar file) and the MIDlet suite descriptor (a .jad file) are created in the bin directory. Extra JAR that are to be used in the project may be placed in the lib directory. Any resource files such as images, text files etc. may be placed in the RES directory. The source code created by you will be in SRC directory. Apart from the above mentioned subdirectories, the following subdirectories are also created:

- tmpclasses
- tmplib

But, these subdirectories are not used by the developer explicitly. They are basically used internally.

The details of code with which MIDlets can be developed is a larger topic and references may be studied for it.

Once the MIDlet projectlet1 is developed, it should be saved as projectlet1.java file in the SRC directory. The following is the exact location of this file:

- D:\WTK22\apps\project1\src\projectlet1.java

Now, build the project. After successful build, run it. Upon running, a mobile phone emulator will pop up. In the emulator, the title of the project will be displayed along with a button labelled Launch. On clicking it, the MIDlet developed will be executed and the result is displayed on the screen. There is a button labelled Exit at the bottom of the display on which the MIDlet is executed. If the Exit button is clicked, then, we exit the MIDlet that is executed. Now, there are two ways to exit the emulator. Either the emulator window can be closed or ESC key can be pressed.

There are a number of emulators provided by J2ME wireless toolkit. One of them can be selected from the KT (ToolKit) tool bar. After selection, we need to execute the project again.

When the project is build, the toolkit will compile all the .java files in the src directory. The compilation is performed in the MIDP environment. There were different APIs in the MIDP and J2SE environment. When the project is being compiled in the MIDP environment, then the APIs from MIDP should be considered for the classes that are used in MIDlets. The corresponding APIs in J2SE are not used. All MIDP classes are verified in two phases before they are loaded and executed. At build time, the first verification is done by J2ME wireless toolkit. When classes are loaded, the second verification is done by device's runtime system. Finally, all MIDlets are packaged into MIDlet suites. These suites are distributed to actual devices.

Along with the J2ME wireless toolkit, information about the following is also provided:

- Application development cycle
- Attributes of MIDlet
- Files in the installed directories
- Device types
- Portability
- Guidelines on the process of configuring an emulator
- Usage of J2ME wireless toolkit.

Until now, we focused on the development of MIDlet. It is possible to do a bit of extra work and make these MIDlets work across networks. For this to happen, we may need to develop servlets to whom MIDlets can establish network connections.

For developing servlets, we need server. Tomcat is the server that can be used for this purpose. There are a number of servers available and every server is having its own advantages and disadvantages.

The first step is to install and run Tomcat. The latest version can be downloaded from the website of Apache Jakarta Project. It comes as a ZIP archive. After downloading, it can be installed into any directory. Tomcat is written in Java. Tomcat needs the location of J2SE to run. Hence, the location of J2SE should be set in the

JAVA_HOME environment variable. Once, all these steps are performed, then, open the command window. Change to the bin directory of Tomcat. Type *startup* at the prompt. That's it. Tomcat starts running. Don't worry about the things that are displayed on the screen once you start it.

It is possible to check whether Tomcat is really running or not. For that, just open the browser window. Try to open http://localhost:8080/. If a default page from Tomcat with links to servlet and JSP examples is displayed, then, it means that the Tomcat is running. To shutdown the Tomcat, open the command window. Change to bin directory of Tomcat and then type the command *shutdown*. That's it. Tomcat starts shutting down.

Now, suppose that a servlet called *counter* is developed. The name of the file will be counter.java. It should be saved to the root directory of Tomcat. Once it is saved to the root directory, it can be compiled. To compile servlets, there is need for servlet API. This should be added to the CLASSPATH before compiling servlets. The servlet API is present in common/lib/servlet.jar under the Tomcat directory. Now, the counter.java servlet can be compiled using javac as follows:

D:\>javac counter.java

Let us place this servlet in a web application. The servlet can be saved to a directory in webapps directory of Tomcat. Let this directory be abcd. Tomcat will know about the new web application through its configuration files. The necessary changes to the configuration files needs to be done by the developer. This can be done by opening the server.xml file from conf directory and adding a context entry.

The reason for performing all these steps is to handle the incoming http requests. If the request begins with /abcd, then the request will be sent to the web application that is located at webapps/abcd. The most important file in any web application is web.xml. It is always stored in WEB-INF directory. This is the time when we start thinking about making the servlet counter.java accessible to the entire world. Suppose that counter.java is placed in a directory called count.

Now, the following is the full path to the servlet:

http://localhost:8080/abcd/counts

Now, necessary changes are to be done to web.xml file so that Tomcat will know that counter servlet should be mapped to abcd/counts.

As we have seen, both servlets and MIDlets can connect to the world via http. Hence, connecting MIDlet to servlet is not a complicated issue.

The following are different steps in the process of connecting MIDlet to a servlet:

- Start Ktoolbar. This is part of J2ME wireless toolkit.
- Open the project1.
- Write the code for the MIDlet (let it be *countmidlet.java*) that connects to counter servlet.
- The screen of countmidlet.java (after executing it) consists of two commands namely Exit and Connect. Clicking on Connect will lead to the invocation of connect( ) method that will establish a network connection. It will also transport the result.
- The countmidlet.java should be saved to apps/project1/src directory under J2ME wireless toolkit directory.
- Now, J2ME wireless toolkit should know that a new MIDlet is added to it. This can be done by going to Settings→ MIDlets→Add. Enter countmidlet for both the MIDlet name and class name. Click on OK.
- Go to Settings→ User Defined. Add the property name as countmidlet.URL. This URL will invoke counter servlet.

☞ Check Your Progress 1

1) J2ME never needs J2SE. True ☐ False ☐

2) Tomcat can be used for developing _____.

3) MIDP stands for _____.

2.4 INTRODUCTION TO JAVA DEVICE TEST SUITE

Using Java Device Test Suite (JDTS), it is possible to test the J2ME applications. The implementations of CLDC and MIDP can be evaluated using this test suite. They can also be validated and verified.

The following are different types of tests can be conducted using JDTS:

- Functional tests
- Stress tests
- Performance tests
- Security tests.

During functional testing, the application is tested whether it behaves as intended or not. It's behaviour with both internal and external components is checked. During stress tests, the application will be run under the upper limits of the memory and the processing power. During performance tests, response time etc. are evaluated. Under security tests, the MIDlets are checked for their security model which also includes access permissions.

It is possible to include additional test suites to JDTS depending on the requirement. This will enable the developer to execute a select set of test cases. It is also possible to automate the entire testing process. But, in this case, test results does not include the cases where problems may arise due to interaction with application. In JDTS, there is a Test Console as well as a Test Manager which help developers to test their J2ME applications.

The following are some of the features of JDTS:

- More than one test suite can be executed at once.
- Test Manager can accept connections from multiple test consoles concurrently. Each test console might be testing several devices with the same configuration.
- Test reports are generated in HTML format.

☞ Check Your Progress 2

1) Using Java Device Test Suite, _____ applications can be tested.

2) Only one test suite can be executed at a time in JDTS. True ☐ False ☐

2.5 SUMMARY

The most basic issue that governs mobile devices is the standard on which they are based on. GSM, CDMA etc. are some of such standards. In this unit, GSM is introduced. Its architecture is also discussed. One of the most popular environments under which wireless applications can be developed is Java. J2ME is the platform under which wireless applications can be developed. The entire process of wireless application development using J2ME is discussed in this unit. Finally, Java Device

Test Suite, which is used to test wireless applications developed using Java is discussed. Different types of tests that can be conducted using are also explained.

2.6 SOLUTIONS/ANSWERS

Check Your Progress 1

1) False

2) Servlets

3) Mobile Information Device Profile.

Check Your Progress 2

1) J2ME

2) False.

2.7 FURTHER READINGS

1) *Programming Wireless Devices with the Java 2 Platform*, Micro Edition, 2nd Edition, Riggs, Roger/ Taivalsaari, Antero/ Van Peursem, Jim/ Huopaniemi, Jyri/ Patel, Mark/ Uotila, Aleksi; Addison Wesley.

2) *Wireless and Mobile Network Architectures*, Yi-Bing Lin and Imrich Chlamtac; Wiley

Reference Websites

- http://java.sun.com
- http://www.rspa.com

2.8 GLOSSARY

- **API** stands for Application Programming Interface. It is a set of classes that can be used in application.
- **CDMA** stands for Code-Division Multiple Access. It is one of the cellular technologies. There are 3 CDMA standards, namely CDMA One, CDMA2000 and W-CDMA.
- **CDC** stands for Connected Device Configuration. It is a specification for J2ME configuration.
- **CLDC** stands for Connected Limited Device Configuration. It is a specification for J2ME configuration.
- **CVM** stands for Compact Virtual Machine (CVM). It is used by CDC and is an optimised Java Virtual Machine (JVM).
- **J2ME** stands for Java 2 Micro Edition. It is a group of specifications and technologies related to Java on small devices.
- **MIDP** stands for Mobile Information Device Profile. It is a specification for J2ME.

- **MIDlet** is an application written for MIDP. They are derived form javax.microedition.midlet.MIDlet class.
- **MSC** stands for Mobile Switching Center. It is a part of cellular phone network which coordinates and switches calls in a given cell.
- **Optional package** is a set of J2ME APIs which provide specific functions such as database access etc.

UNIT 3 CASE TOOLS

Structure **Page Nos.**

3.0 INTRODUCTION

Software Engineering is broadly associated with the development of quality software with increasing use of software preparation standards and guidelines. Software is the single most expensive item in a computer system as the cost of software during the life time of a machine is equivalent to more than 95% of the total cost (including hardware). Software Engineering requires a lot of data collection and information generation. Since the computer itself is a very useful device as the information processor, it may be a good idea to automate software engineering tasks. Computer Aided Software Engineering (CASE) tools instill many software engineering tasks with the help of information created using computer. CASE tools support software engineering tasks and are available for different tasks of the Software Development Life Cycle (SDLC). You have been introduced to CASE tools in Unit 10 of MCS-014. This unit covers various aspects of these CASE tools and their functionality for various phases of software development.

3.1 OBJECTIVES

After going through this unit, you should be able to:

- define different kinds of CASE tools and the needs of the CASE tools;
- describe the features of analysis, design, tools use in Software Engineering; and
- identify software configuration management, and project management tool.

3.2 WHAT ARE CASE TOOLS?

Computer Aided Software Engineering (CASE) tools are gradually becoming popular for the development of software as they are improving in the capabilities and functionalities and are proving to be beneficial for the development of quality software. But, what are the CASE tools? And how do they support the process of development of software?

CASE tools are the software engineering tools that permit collaborative software development and maintenance. CASE tools support almost all the phases of the

software development life cycle such as analysis, design, etc., including umbrella activities such as project management, configuration management etc. The CASE tools in general, support standard software development methods such as Jackson Structure programming or structured system analysis and design method. The CASE tools follow a typical process for the development of the system, for example, for developing data base application, CASE tools may support the following development steps:

- Creation of data flow and entity models
- Establishing a relationship between requirements and models
- Development of top-level design
- Development of functional and process description
- Development of test cases.

The CASE tools on the basis of the above specifications can help in automatically generating data base tables, forms and reports, and user documentation.

Thus, the CASE tools –

- support contemporary development of software systems, they may improve the quality of the software
- help in automating the software development life cycles by use of certain standard methods
- create an organsiation wide environment that minimizes repetitive work
- help developers to concentrate more on top level and more creative problem-solving tasks
- support and improve the quality of documentation (Completeness and non-ambiguity), testing process (provides automated checking), project management and software maintenance.

Most of the CASE tools include one or more of the following types of tools:

- Analysis tools
- Repository to store all diagrams, forms, models and report definitions etc.
- Diagramming tools
- Screen and report generators
- Code generators
- Documentation generators
- Reverse Engineering tools (that take source code as input and produce graphical and textual representations of program design-level information)
- Re-engineering tools (that take source code as the input analyse it and interactively alters an existing system to improve quality and/or performance).

Some necessary features that must be supported by CASE tools in addition to the above are:

- It should have Security of information. The information may be visible/ changeable by authorised users only.
- Version Control for various products
- A utility to Import/Export information from various external resources in a compatible fashion
- The process of Backup and Recovery as it contains very precious data.

Figure 3.1 shows an environment having CASE.

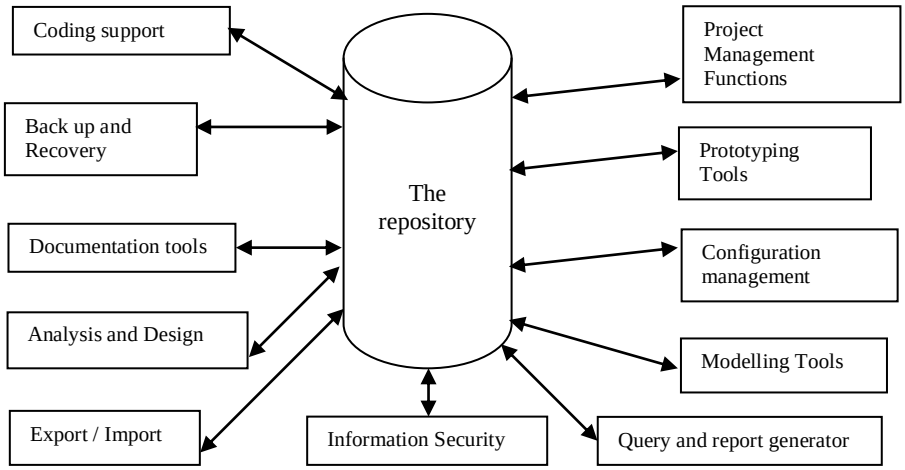

Figure 3.1: CASE Tools

3.2.1 Categories of CASE Tools

On the basis of their activities, sometimes CASE tools are classified into the following categories:

1. Upper CASE tools
2. Lower CASE tools
3. Integrated CASE tools.

Upper CASE: Upper CASE tools mainly focus on the analysis and design phases of software development. They include tools for analysis modeling, reports and forms generation.

Lower CASE: Lower CASE tools support implementation of system development. They include tools for coding, configuration management, etc.

Integrated CASE Tools: Integrated CASE tools help in providing linkages between the lower and upper CASE tools. Thus creating a cohesive environment for software development where programming by lower CASE tools may automatically be generated for the design that has been developed in an upper CASE tool.

Figure 3.2 shows the positioning of CASE tools in a Software Application development.

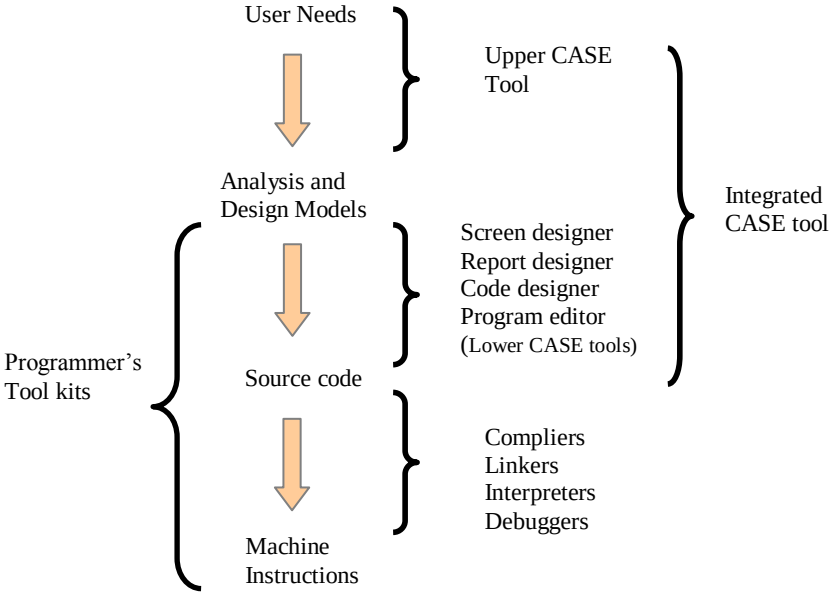

Figure 3.2: The CASE tools and Application Development

3.2.2 Need of CASE Tools

The software development process is expensive and as the projects become more complex in nature, the project implementations become more demanding and expensive. The CASE tools provide the integrated homogenous environment for the development of complex projects. They allow creating a shared repository of information that can be utilised to minimise the software development time. The CASE tools also provide the environment for monitoring and controlling projects such that team leaders are able to manage the complex projects. Specifically, the CASE tools are normally deployed to –

- Reduce the cost as they automate many repetitive manual tasks.
- Reduce development time of the project as they support standardisation and avoid repetition and reuse.
- Develop better quality complex projects as they provide greater consistency and coordination.
- Create good quality documentation
- Create systems that are maintainable because of proper control of configuration item that support traceability requirements.

But please note that CASE tools cannot do the following:

- Automatic development of functionally relevant system
- Force system analysts to follow a prescribed methodology
- Change the system analysis and design process.

There are certain disadvantages of CASE tools. These are:

- Complex functionality
- Many project management problems are not amenable to automation. Hence, CASE tools cannot be used in such cases.

3.2.3 Factors that affect deployment of CASE Tools in an organisation

A successful CASE implementation requires the following considerations in an organisation:

1. Training all the users in typical CASE environment that is being deployed, also giving benefits of CASE tools.
2. Compulsory use of CASE initially by the developers.
3. Closeness of CASE tools methodology to the Software Development Life Cycle
4. Compatibility of CASE tools with other development platforms that are being used in an organisation.
5. Timely support of vendor with respect to various issues relating to CASE tools:
    - Low cost support.
    - Easy to use and learn CASE tools having low complexity and online help.
    - Good graphic support and multiple users support.
6. Reverse Engineering support by the CASE tools: It is important that a CASE tool supports complicated nature of reverse engineering.

3.2.4 Characteristics of a successful CASE Tools

A CASE tool must have the following characteristics in order to be used efficiently:

- **A standard methodology:** A CASE tool must support a standard software development methodology and standard modeling techniques. In the present scenario most of the CASE tools are moving towards UML.

- **Flexibility**: Flexibility in use of editors and other tools. The CASE tool must offer flexibility and the choice for the user of editors' development environments.
- **Strong Integration**: The CASE tools should be integrated to support all the stages. This implies that if a change is made at any stage, for example, in the model, it should get reflected in the code documentation and all related design and other documents, thus providing a cohesive environment for software development.
- **Integration with testing software**: The CASE tools must provide interfaces for automatic testing tools that take care of regression and other kinds of testing software under the changing requirements.
- **Support for reverse engineering**: A CASE tools must be able to generate complex models from already generated code.
- **On-line help**: The CASE tools provide an online tutorial.

Now let us discuss some of the important characteristics for various types of CASE tools in the subsequent sections.

☞ Check Your Progress 1

1) What is the need of CASE tools?

……..

……..

……

2) What are the important characteristics of CASE tools?

……..

……..

……

3.3 CASE SOFTWARE DEVELOPMENT ENVIRONMENT

CASE tools support extensive activities during the software development process. Some of the functional features that are provided by CASE tools for the software development process are:

1. Creating software requirements specifications
2. Creation of design specifications
3. Creation of cross references.
4. Verifying/Analysing the relationship between requirement and design
5. Performing project and configuration management
6. Building system prototypes
7. Containing code and accompanying documents.
8. Validation and verification, interfacing with external environment.

Some of the major features that should be supported by CASE development environment are:

- a strong visual support
- prediction and reporting of errors
- generation of content repository
- support for structured methodology
- integration of various life cycle stages
- consistent information transfer across SDLC stages
- automating coding/prototype generation.

Present CASE tools support Unified Model Language (UML).

We will elaborate on the features of CASE tools for various stages of software development process in coming sub-sections.

CASE and Web Engineering

CASE Tools are also very useful in the design, development and implementation of web site development.

Web Engineering requires tools in many categories. They are:

- Site content management tools
- Site version control tools
- Server management tool
- Site optimisation tools
- Web authoring and deployment tools
- Site testing tools that include load and performance testing
- Link checkers
- Program checkers
- Web security test tools.

A detailed discussion on these tools is beyond the scope of this unit. However, various stages of development of a web project also follows the normal SDLC. These are discussed in the subsequent sections.

3.4 CASE TOOLS AND REQUIREMENT ENGINEERING

Let us first answer the question:

Which is the most Common Source of Risk during the process of software development?

One of the major risk factors that affect project schedule, budget and quality can be defined as the ability to successfully elicit requirements to get a solution.

Statistically it has been seen that about 80% of rework in projects is due to requirement defects.

How can a CASE tools help in effective Requirements Engineering (RE)

A good and effective requirements engineering tool needs to incorporate the best practices of requirements definition and management.

The requirements Engineering approach should be highly iterative with the goal of establishing managed and effective communication and collaboration.

Thus, a CASE tool must have the following features from the requirements engineering viewpoint:

- a dynamic, rich editing environment for team members to capture and manage requirements
- to create a centralised repository
- to create task-driven workflow to do change management, and defect tracking.

But, what is a good process of Requirements Engineering for the CASE?

A simple four-step process for requirements engineering is given in *Figure 3.3*.

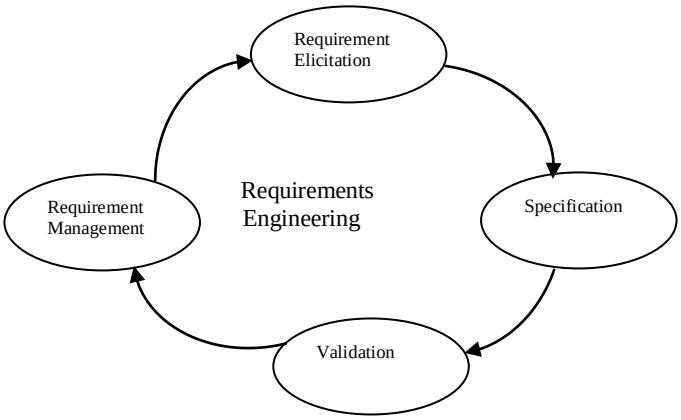

Figure 3.3: The four-step requirement engineering process

Requirement Elicitation

A simple technique for requirements elicitation is to ask "WHY".

CASE tools support a dynamic, yet intuitive, requirements capture and management environment that supports content and its editing. Some of the features available for requirement elicitation are:

- Reusable requirements and design templates for various types of system
- Keeping track of important system attributes like performance and security
- It may also support a common vocabulary of user-defined terms that can be automatically highlighted to be part of glossary.
- Provide feature for the assessment of the quality of requirements
- Separate glossary for ambiguous terms that can be flagged for additional clarification.

What do we expect from the tool?

- It should have rich support for documentation that is easily understandable to stakeholders and development teams.
- It should have the capability of tracking of requirements during various SDLC systems.
- It should help in the development of standard technical and business terminology for end clients.

Software Analysis and Specification

One of the major reasons of documenting requirements is to remove the ambiguity of information. A good requirement specification is testable for each requirement.

One of the major features supported by CASE tools for specification is that the design and implementation should be traceable to requirements. A good way to do so is to support a label or a tag to the requirements. In addition it should have the following features:

- Must have features for storing and documenting of requirements.
- Enable creation of models that are critical to the development of functional requirements.
- Allow development of test cases that enable the verification of requirements and their associated dependencies.
- Test cases help in troubleshooting the correlation between business requirements and existing system constraints.

What do we expect from the tool?

- It should allow development of a labeled requirements document that helps in traceability of requirements.
- It should allow both functional and non-functional requirements with related quality attributes to be made.
- We should be able to develop the associated models.

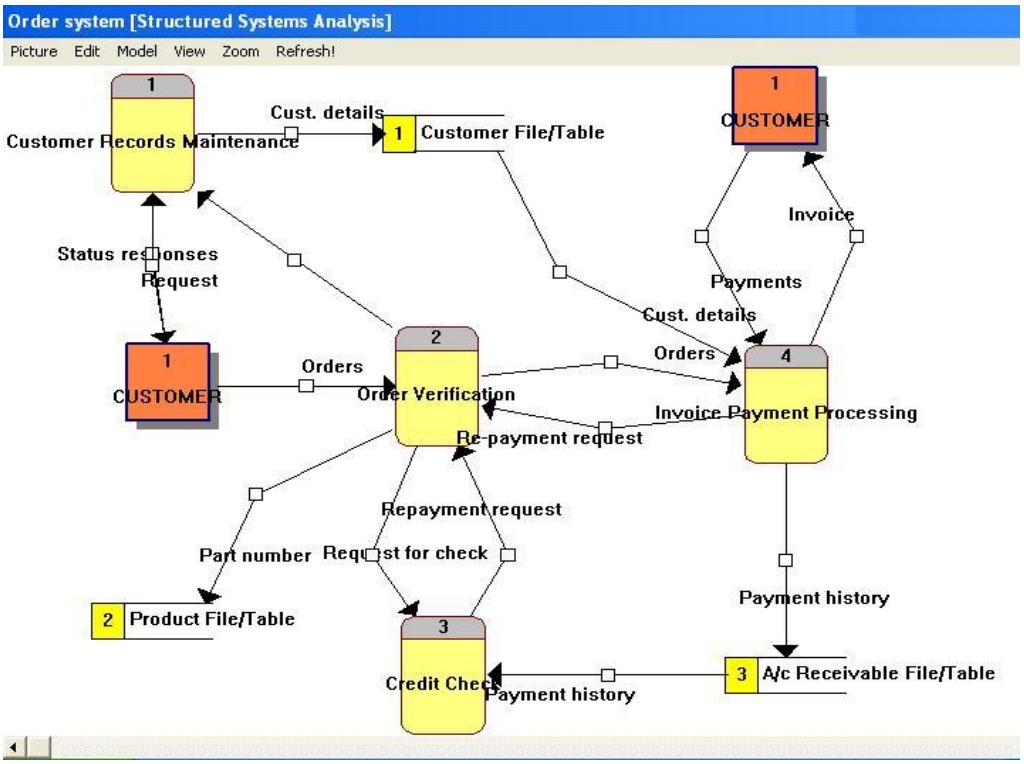

Figure 3.4: A sample DFD using CASE Tools

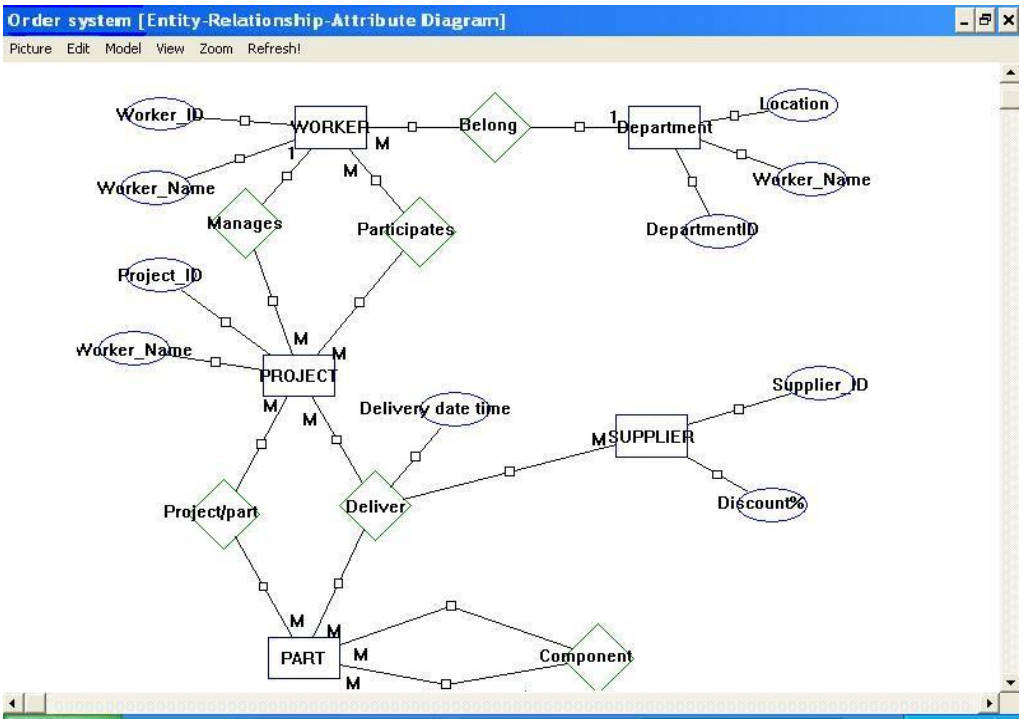

Figure 3.5: A Sample ER Diagram using CASE Tools

Figures 3.4 and 3.5 show some of the models developed with the help of a sample CASE Tool.

"The automated approach of software development needs to have a spirit of collaboration, bringing together world-renowned principal investigators, business analysts, and software development teams."

Validation of Requirements

A very important feature in this regard is to allow collaboration yet customizable workflows for the software development team members. Also facilitating approvals and electronic signatures to facilitate audit trails. Assigning owner of requirement may be helpful if any quality attributes may need changes. Thus, a prioritised validated documented and approved requirements can be obtained.

Managing Requirements

The requirements document should have visibility and help in controlling the software delivery process. Some such features available in CASE tools are:

- estimation of efforts and cost
- specification of project schedule such as deadline, staff requirements and other constraints
- specification of quality parameters.

Software Change Management

An incomplete or ambiguous requirement if detected early in the analysis phase can be changed easily with minimum cost. However, once they are converted to baselines after requirements validations the change should be controlled.

One of the major requirements of change management is:

Any change to these requirements should follow a process which is defined as the ability to track software requirements and their associated models/documents, etc. (e.g., designs, DFDs and ERDs, etc.). This helps in determining the components that will be impacted due to change. This also helps tracking and managing change. The whole process starts with labeling the requirements properly.

Most CASE Tools store requirement baselines, including type, status, priority and change history of a software item. Such traceability may be bi-directional in nature.

Static word processing documents or spreadsheet recedes communication issues because those tools do not allow sharing and collaboration on up-to-date requirements. To overcome this a CASE enabled requirements management infrastructure offers the familiarity of environments such as Microsoft Word, with added communication methods, such as email. Thus, CASE is a very useful tool for requirement engineering.

What are the features of high quality requirements?

- Correctness
- Unambiguous
- Must
- Consistency
- Known constraints
- Modifiable
- Priority Assigned
- Traceable
- Verifiable

3.5 CASE TOOLS AND DESIGN AND IMPLEMENTATION

In general, some CASE tools support the analysis and design phases of software development. Some of the tools supported by the design tools are:

- Structured Chart.
- Pogram Document Language (PDL).
- Optimisation of ER and other models.
- Flow charts.
- Database design tools.
- File design tools.

Some of functions that these diagrams tool support are simple but are very communicative as far as representations of the information of the analysis and design phases are concerned. CASE Tools also support standard representation of program architecture; they also contain testing items related to the design and debugging. Automatic support for maintenance is provided in case any of the requirements or design items is modified using these diagrams. These CASE tools also support

error-checking stages. They allow checks for completeness and consistency of the required functional design types and consistency at various levels of cross referencing among the documents consistency checking for these documents during the requirements analysis and design phases.

Proper modeling helps in proper design architecture. All the CASE tools have strong support for models. They also help in resolving conflicts and ambiguity among the models and help in optimising them to create excellent design architecture and process implementation.

But why do we need to model?

Can you understand code? If you understand it, then why would you call it code?

The major advantages for modeling are:
A model enhances communication, as it is more pictorial and less code.

Even early humans have used pictures to express ideas in a better way.

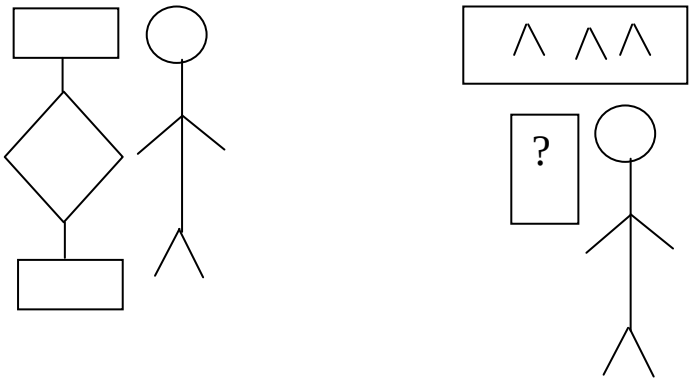

Figure 3.6: Model versus code

A model

- helps the users
- may reduce the cost of project
- can convey a broad spectrum of information.

Some standard models may need to represent:

- Overall system architecture of the software
- Software dependencies
- Flow of information through a software system
- Database organisation and structure.
- Deployment configuration, etc.

Models also help in better planning and reduction of risk as one can make top down models, thus controlling complexity.

So what is a good modeling tool?

The following are some of the characteristics of a good modeling tool:

CASE tools provide continuously synchronized models and code, thus also help in consistent understanding of information and documentation. It also helps other software developers to understand the portions and relationships to portions other team members are working on.

Help in management of source code through a more visual model.

Refactoring of the code allows a re-look and improvement of code, as modeling tools contains, thus are most continuously sychronised code and models suitable for refactoring.

Modeling can help in creating good patterns including modeling patterns, language specific patterns and basic patterns for common operations such as implementing interfaces, etc.

Models also facilitate reverse engineering.

Given the task of maintaining software that was built by a team that has moved on to a new project, there may be very little documentation available for the project.

The value of a modeling tool can be tremendous in such cases. The developer can take the existing code, create models for it to understand the software. This can save a lot of time for the developer.

A modeling tool should have been integrated with requirements tools, so that architects see consistently unambiguous information.

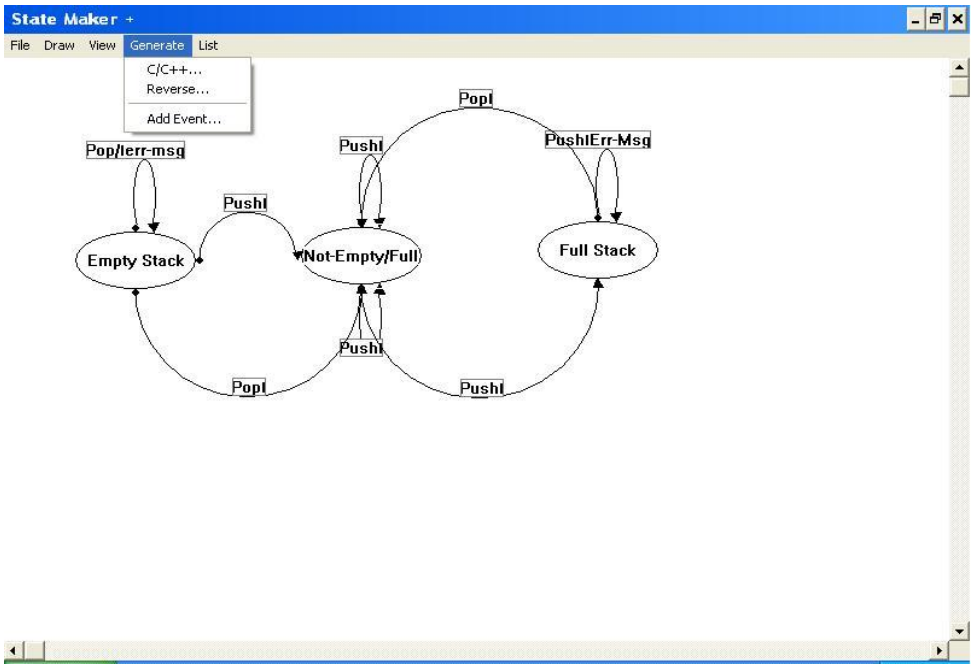

Figure 3.7: A State Model

CASE Repository

CASE Repository stores software system information. It includes analysis and design specifications and helps in analysing these requirements and converting them into program code and documentation. The following is the content of CASE repository:

- Data
- Process
- Models
- Rules/ Constraints.

Data: Information and entities/object relationship attributes, etc.

Process: Support structured methodology, link to external entities, document structured software development process standards.

Models: Flow models, state models, data models, UML document etc.

Rules/Constraints: Business rules, consistency constraints, legal constraints.

The CASE repository has two primary segments.

1. Information repository
2. Data dictionary.

Information Repository includes information about an organisation's business information and its applications.

The CASE tools manage and control access to repository. Such information data can also be stored in corporate database.

Data dictionary contains all the data definitions for all organisational applications, along with cross-referencing if any.

Its entries have a standard definition, viz., element name and alias; textual description of the element; related elements; element type and format; range of acceptable values and other information unique to the proper processing of the element.

CASE Repository has additional advantages such that it assists the project management tasks; aids in software reusability by enabling software modules in a manner so that they can be used again.

Implementation tools and CASE

CASE tools provide the following features for implementation:

- Diagramming tools enable visual representation of a system and its components
- Allow representing process flows.
- Help in implementing the data structures and program structures
- Support automatic creation of system forms and reports.
- Ready prototype generation.
- Create of both technical and user documentation.
- Create master templates used to verify documentation and its conformance to all stages of the Software Development Life Cycle (SDLC).
- Enable the automatic generation of program and database from the design documents, diagrams, forms and reports stored in the repository.

3.6 SOFTWARE TESTING

CASE tools may also support software testing. One of the basic issues of testing is management, execution and reporting. Testing tools can help in automated unit testing, functional regression testing, and performance testing. A testing tool ensures the high visibility of test information, types of common defects, and application readiness.

Features Needed for Testing

The tool must support all testing phases, viz., plan, manage and execute all types of tests viz., functional, performance, integration, regression, testing from the same interface.

It should integrate with other third party testing tools.

It should support local and remote test execution.

It should help and establish and manage traceability by linking requirements to test cases. This also helps when requirements change; in such as case the test cases traced to the requirement are automatically flagged as possible candidates for change.

It should create a log of test run.

It should output meaningful reports on test completeness, test analysis.

It may provide automated testing.

☞ Check Your Progress 2

1) List any four tools that are used in web software engineering but are not used in general software projects.

……

……

2) Which of the requirements engineering processes is not supported by CASE tools?

……

……

3) List any four major requirements of a design tool.

……

……

4) List four important features of a testing tool.

……

……..

3.7 SOFTWARE QUALITY AND CASE TOOLS

Software quality is sacrificed by many developers for more functionality, faster development and lower cost. However, one must realise that a good quality product actually enhances the speed of software development. It reduces the cost and allows enhancement and functionality with ease as it is a better structured product. You need to pay for a poor quality in terms of more maintenance time and cost. Can the good quality be attributed to a software by enhancing the testing? The answer is NO. The high quality software development process is most important for the development of quality software product. Software quality involves functionality for software usability, reliability, performance, scalability, support and security.

Integrated CASE tools:

- help the development of quality product as they support standard methodology and process of software development
- supports an exhaustive change management process
- contains easy to use visual modeling tools incorporating continuous quality assurance.

Quality in such tools is represented in all life cycle phases viz., Analysis/ Design development, test and deployment. Quality is essential in all the life cycle phases.

Analysis: A poor understanding of analysis requirements may lead to a poor product. CASE tools help in reflecting the system requirements clearly, accurately and in a simple way. CASE tools support the requirements analysis and coverage also as we have modeling. CASE also helps in ambiguity resolution of the requirements, thus making high quality requirements.

Design: In design the prime focus of the quality starts with the testing of the architecture of the software. CASE tools help in detecting, isolating and resolving structure deficiency during the design process. On an average, a developer makes 100 to 150 errors for every thousand lines of code. Assuming only 5% of these errors are

serious, if software has ten thousand lines of code you may still have around 50 serious coding errors in your system. One of the newer software development processes called the Agile process helps in reducing such problems by asking the developer to design their test items first before the coding.

A very good approach that is supported by CASE tools specially running time development of C, C++, JAVA or .NET code is to provide a set of automatic run time Language tools for development of reliable and high performance applications.

Testing: Functionality and performance testing is an integrated part of ensuring high quality product. CASE support automated testing tools that help in testing the software, thus, helping in improving the quality of testing. CASE tools enhance the speed breadth and reliability of these design procedures. The design tools are very important specifically in case of a web based system where scalability and reliability are two major issues of design.

Deployment: After proper testing a software goes through the phase of deployment where a system is made operational. A system failure should not result in a complete failure of the software on restart. CASE tools also help in this particular place. In addition, they support configuration management to help any kind of change thus to be made in the software.

Quality is teamwork: It involves integration of workflow of various individuals. It establishes a traceability and communication of information, all that can be achieved by sharing workload documents keeping their configuration items.

3.8 SOFTWARE CONFIGURATION MANAGEMENT

Software Configuration Management (SCM) is extremely important from the view of deployment of software applications. SCM controls deployment of new software versions. Software configuration management can be integrated with an automated solution that manages distributed deployment. This helps companies to bring out new releases much more efficiently and effectively. It also reduces cost, risk and accelerates time.

A current IT department of an organisation has complex applications to manage. These applications may be deployed on many locations and are critical systems. Thus, these systems must be maintained with very high efficiency and low cost and time. The problem for IT organisations has increased tremendously as some of the organisations may need to rebuild and redeploy new software versions a week over multi-platform global networks.

In such a situation, if rebuilding of application versions is built or deployed manually using a spreadsheet, then it requires copying of rebuilt software to many software locations, which will be very time consuming and error prone. What about an approach where newer software versions are automatically built and deployed into several distributed locations through a centralised control. For example, assume that IGNOU has a data entry software version 1.0 for entering assignment marks which is deployed all the RCs. In case its version 1.1 is to be deployed, if the re-built software need to be sent and deployed manually, it would be quite troublesome. Thus, an automatic deployment tool will be of great use under the control of SCM.

What do we need?

We need an effective SCM with facilities of automatic version control, access control, automatic re-building of software, build audit, maintenance and deployment. Thus, SCM should have the following facilities:

- Creation of configuration

- This documents a software build and enables versions to be reproduced on demand
- Configuration lookup scheme that enables only the changed files to be rebuilt. Thus, entire application need not be rebuilt.
- Dependency detection features even hidden dependencies, thus ensuring correct behaviour of the software in partial rebuilding.
- Ability for team members to share existing objects, thus saving time of the team members.

Figure 3.8 shows a simple Configuration Management based rebuilding and deployment process.

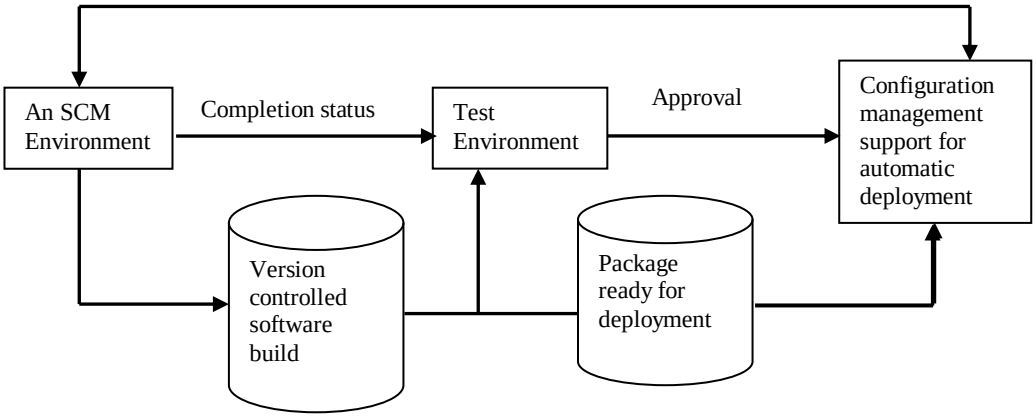

Figure 3.8: Simple configuration management environment

3.9 SOFTWARE PROJECT MANAGEMENT AND CASE TOOLS

Developing commercial software is not a single-player activity. It is invariably a collaborative team activity. The development of business-critical systems are also too risky, or are too large for a single developer to deliver the product in a stipulated time and quality frame.

A software team involves designers, developers, and testers who work together for delivering the best solution in the shortest time. Sometimes, these teams can be geographically dispersed. Managing such a team may be a major change requests, and task management.

The CASE tools help in effective management of teams and projects. Let us look into some of the features of CASE in this respect:

- Sharing and securing the project using the user name and passwords.
- Allowing reading of project related documents
- Allowing exclusive editing of documents
- Linking the documents for sharing
- Automatically communicative change requests to approver and all the persons who are sharing the document.
- You can read change requests for yourself and act on them accordingly.
- Setting of revision labels so that versioning can be done.
- Any addition or deletion of files from the repository is indicated.
- Any updating of files in repository is automatically made available to users.
- Conflicts among versions are reported and avoided
- Differences among versions can be visualized.

- The linked folder, topics, and change requests to an item can be created and these items if needed can be accessed.
- It should have reporting capabilities of information

The project management tools provide the following benefits:

- They allow control of projects through tasks so control complexity.
- They allow tracking of project events and milestones.
- The progress can be monitored using Gantt chart.
- Web based project related information can be provided.
- Automatic notifications and mails can be generated.

Some of the features that you should look into project management software are:

- It should support drawing of schedules using PERT and Gantt chart.
- It should be easy to use such that tasks can be entered easily, the links among the tasks should be easily desirable.
- Milestones and critical path should be highlighted.
- It should support editing capabilities for adding/deleting/moving tasks.
- Should map timeline information against a calendar.
- Should allow marking of durations for each task graphically.
- It should provide views tasks, resources, or resource usage by task.
- Should be useable on network and should be able to share information through network.

☞ **Check Your Progress 3**

1) How do CASE tools support quality?

……

……

2) What is the role of CASE in software configuration management?

……

……

3) Can CASE tools result in perfect project management?

……

……

3.10 SUMMARY

This unit provides an introduction to CASE tools in action. The CASE tools are primarily used to automate the stages of the Software Development Life Cycle. It automates most of the tasks of software engineering such as requirements analysis, modeling, designing, support for testing, and implementation, project management, software configuration management, software quality management, etc. CASE tools also help in traceability of requirements. They are useful tools for reverse engineering. Presently, many CASE tools which integrate the complete Software Development Life Cycle are available and can be deployed successfully in an organisation. These CASE tools support standard methodology, provide flexibility of environment, strong integration of various software development stages, reverse engineering, project management, and configuration management under one single environment. However, it is to be noted that all deployments of CASE have not succeeded.

Successful CASE tools are those which are simple to use support standard methodology, and reverse engineering and have a strong vendor support including training.

3.11 SOLUTIONS/ANSWERS

Check Your Progress 1

1) CASE tools are needed for
   - Development of cost effective software
   - Minimisation of development time
   - Standardising the software development process
   - Avoiding repetition and maximising reuse
   - Development of better quality product
   - Collaborative developments etc.

2) CASE tools
   - Follow standard methodology
   - Allow integration of information
   - Allow traceability
   - Help improving of quality
   - Reduce cost
   - Support reverse engineering
   - Provide on-line help
   - Check for inconsistency of information
   - Provide tools for configuration management and project management.

Check Your Progress 2

1) Web security test tools, link checkers, site optimization tools, server management tools

2) Although requirements engineering phases viz., requirement elicitation, specification, validation, requirements management, have some part being played by CASE tools, yet it is the stage of requirement elicitation where CASE tools are least helpful as this stage requires interaction with the users.

3) The design tools support the following:
   a. A strong consistent data dictionary
   b. Support for presenting design architecture and data design.
   c. Support for traceability and integration with test scenarios
   d. Optimisation of models and process flow charts.

4) Test planning, Integration with automatic testing tool, Traceability, Flagging the possible candidates for change.

Check Your Progress 3

1) The quality of the software is supported at all the Life Cycle stages of software development with the help of CASE tools. For example, in the analysis phase CASE tools may automatically flag requirement ambiguity in

the design phase they may point out any inconsistency, in the test phase they may point out the reliability of software, etc.

2) CASE tools control the baseline configuration items thus support a strict change control such that any changes made during any life cycle phase results in automatic flagging of the various documents of various stages of SDLC affected by that change. They also help in version control.

3) No. They are only predictors of various failures to meet the schedule of the project.

3.12 FURTHER READINGS

1) *Software Engineering, Sixth Edition, 2001*, Ian Sommerville; Pearson Education.
2) *Software Engineering – A Practitioner's Approach*, Roger S. Pressman; McGraw-Hill International Edition.

Reference Websites

- **http://www.rspa.com**
- **http://www.ieee.org**

UNIT 4 ADVANCED TOPICS IN SOFTWARE ENGINEERING

Structure **Page Nos.**

4.0 INTRODUCTION

In the first few units we have studied the various methods for conventional software engineering. Software engineering methods can be categorised on a "formality spectrum." The analysis and design methods discussed previously would be placed at the informal to moderately rigorous end of the spectrum. In these methods, a combination of diagrams, text, tables, and simple notation is used to create analysis and design models.

At the other end of the formality spectrum are formal methods, which are completely mathematical in form. A specification and design are described using a formal syntax and semantics that specify system function and behaviour. Other specialised models being the Cleanroom software engineering, component-based software engineering, re-engineering and reverse engineering.

Harlan Mills has developed the Cleanroom software engineering methodology, which is an approach for integrating formal methods into the lifecycle. The Cleanroom approach combines formal methods and structured programming with Statistical Process Control (SPC), the spiral lifecycle and incremental releases, inspections, and software reliability modeling. It fosters attitudes, such as emphasising defect prevention over defect removal, that are associated with high quality products in non-software fields.

In most engineering disciplines, systems are designed by composition (building system out of components that have been used in other systems). Software engineering has focused on custom development of components. To achieve better software quality, more quickly, at lower costs, software engineers are beginning to adopt *systematic reuse* as a design process. There are various ways to achieve this and one of the models is the Component-based software engineering.

Let us study the various specialised software engineering models like the formal method, Cleanroom engineering, component-based engineering, re-engineering and reverse engineering in this unit.

4.1 OBJECTIVES

After going through this unit, you should be able to:

- understand the pitfalls of the conventional software engineering;
- know various advanced methods of software engineering – their advantages and drawbacks;
- describe the methodology of formal methods and Cleanroom software engineering, and
- discuss the Re-engineering and Reverse engineering process.

4.2 EVOLUTION OF FORMAL METHODS

The seventies witnessed the structured programming revolution. After much debate, software engineers became convinced that better programs result from following certain precepts in program design. Recent imperative programming languages provide constructs supporting structured programming. Achieving this consensus did not end debate over programming methodology. On the contrary, a period of continuous change began, with views on the best methods of software development mutating frequently. Top-down development, modular decomposition, data abstraction, and, most recently, object oriented design are some of the jargon terms that have arisen to describe new concepts for developing large software systems. Both researchers and practitioners have found it difficult to keep up with this onslaught of new methodologies.

There is a set of core ideas that lies at the base of these changes. Formal methods have provided a unifying philosophy and central foundation upon which these methodologies have been built. Those who understand this underlying philosophy can more easily adopt these and other programming techniques.

One way to improve the quality of software is to change the way in which software is documented: at the design stage, during development, and after release. Existing methods of documentation offer large amounts of text, pictures, and diagrams, but these are often imprecise and ambiguous. Important information is hidden amongst irrelevant details, and design flaws are discovered too late, making them expensive or impossible to correct.

There is an alternative. *Formal* methods, based upon elementary mathematics, can be used to produce precise, unambiguous documentation, in which information is structured and presented at an appropriate level of abstraction. This documentation can be used to support the design process, and as a guide to subsequent development, testing, and maintenance.

It seems likely that the use of formal methods will become standard practice in software engineering. The mathematical basis is different from that of civil or mechanical engineering, but it has the same purpose: to add precision, to aid understanding, and to reason about the properties of a design. Whatever the discipline, the use of mathematics can be expensive, but it is our experience that it can actually reduce costs.

Existing applications of formal methods include: the use of the probability theory in performance modeling; the use of context-free grammars in compiler design; the use of the relational calculus in database theory. The formal method described in this book has been used in the specification and design of large software systems. It is intended for the description of state and state-based properties, and includes a theory of refinement that allows mathematics to be used at every stage of program development. Let us see the details of the Formal methods in the next section.

4.3 USE OF MATHEMATICS IN SOFTWARE DEVELOPMENT

Mathematics supports *abstraction* and therefore is a powerful medium for modeling. Because they are exact, mathematical specifications are *unambiguous* and can be validated to uncover *contradictions* and *incompleteness*. It allows a developer to *validate* a specification for functionality. It is possible to demonstrate that a design matches a specification, and that some program code is a correct reflection of a design.

How is the mathematics of formal languages applied in software development? What engineering issues have been addressed by their application? Formal methods are of global concern in software engineering. They are directly applicable during the requirements, design, and coding phases and have important consequences for testing and maintenance. They have influenced the development and standardisation of many programming languages, the programmer's most basic tool. They are important in ongoing research that may change standard practice, particularly in the areas of specifications and design methodology. They are entwined with lifecycle models that may provide an alternative to the waterfall model, namely rapid prototyping, the Cleanroom variant on the spiral model, and "transformational" paradigms. The concept of formalism in formal methods is borrowed from certain trends in 19th and 20th century mathematics. Formal methods are merely an adoption of the axiomatic method, as developed by these trends in mathematics, for software engineering. Mastery of formal methods in software requires an understanding of this mathematics background. Mathematical topics of interest include formal logic, both the propositional calculus and predicate logic, set theory, formal languages, and automata such as finite state machines.

4.4 FORMAL METHODS

A **formal method** *in software development is a method that provides a formal language for describing a software artifact (e.g., specifications, designs, source code) such that formal proofs are possible, in principle, about properties of the artifact so expressed.*

The definition is based on two essential components. First, formal methods involve the essential use of a formal language. A formal language is a set of strings over some well-defined alphabet. Rules are given for distinguishing those strings, defined over the alphabet, that belong to the language from strings that do not. Secondly, formal methods in software support formal reasoning about formulae in the language. These methods of reasoning are exemplified by formal proofs. A proof begins with a set of axioms, which are to be taken as statements postulated to be true. Inference rules state that if certain formula, known as premises, are derivable from the axioms, then another formula, known as the consequent, is also derivable. A set of inference rules must be given in each formal method. A proof consists of a sequence of well-defined formulae in the language in which each formula is either an axiom or derivable by an inference rule from previous formulae in the sequence. The last formula in the sequence is said to be proven.

The major concerns of the formal methods are as follows:

i) The correctness of the problem
   - producing software that is "correct" is famously difficult;
   - by using rigorous mathematical techniques, it may be possible to make probably correct software.

ii) Programs are mathematical objects
   - they are expressed in a **formal** language;
   - they have a **formal** semantics;
   - programs can be treated as mathematical theories.

Formal methods support precise and rigorous specifications of those aspects of a computer system capable of being expressed in the language. Since defining what a system should do, and understanding the implications of these decisions, are the most troublesome problems in software engineering, this use of formal methods has major benefits. In fact, practitioners of formal methods frequently use formal methods solely for recording precise specifications, not for formal verifications.

Formal methods were originally developed to support verifications, but higher interest currently exists in specification methods. Several methods and languages can be used for specifying the functionality of computer systems. No single language, of those now available, is equally appropriate for all methods, application domains, and aspects of a system. Thus, users of formal specification techniques need to understand the strength and weaknesses of different methods and languages before deciding on which to adopt. The distinction between a specification method and a language is fundamental. A method states what a specification must say. A language determines in detail how the concepts in a specification can be expressed. Some languages support more than one method, while most methods can be used in several specification languages.

Some of the most well-known formal methods consist of or include specification languages for recording a system's functionality. These methods include:

- Z (pronounced "Zed")
- Communicating Sequential Processes (CSP)
- Vienna Development Method (VDM)
- Larch
- Formal Development Methodology (FDM)

4.4.1 What can be Formally Specified?

Formal methods can include graphical languages. Data Flow Diagrams (DFDs) are the most well-known graphical technique for specifying the function of a system. DFDs can be considered a semi-formal method, and researchers have explored techniques for treating DFDs in a completely formal manner. Petri nets provide another well-known graphical technique, often used in distributed systems. Petri nets are a fully formal technique.

Finally, finite state machines are often presented in tabular form. This does not decrease the formalism in the use of finite state machines. So the definition of formal methods provided earlier is quite encompassing.

Software engineers produce models and define the properties of systems at several levels of abstraction. Formal methods can be employed at each level. A specification should describe what a system should do, but not how it is done. More details are provided in designs, with the source code providing the most detailed model.

For example, Abstract Data Types (ADTs) frequently are employed at intermediate levels of abstraction. ADTs, being mathematical entities, are perfect candidates for formal treatment and are often so treated in the literature.

Formal methods are not confined to the software components of large systems. System engineers frequently use formal methods. Hardware engineers also use formal methods, such as VHSIC Hardware Description Language (VHDL) descriptions, to

model integrated circuits before fabricating them. The following section lists the goals of the formal specification.

4.4.2 Goals of Formal Specification

Once a formal description of a system has been produced, what can be done with it? Usable formal methods provide a variety of techniques for reasoning about specifications and drawing implications. The completeness and consistency of a specification can be explored. Does a description imply a system should be in several states simultaneously? Do all legal inputs yield one and only one output? What surprising results, perhaps unintended, can be produced by a system? Formal methods provide reasoning techniques to explore these questions. Do lower level descriptions of a system properly implement higher level descriptions? Formal methods support formal verification, the construction of formal proofs that an implementation satisfies a specification. The possibility of constructing such formal proofs was historically the principal driver in the development of formal methods. Prominent technology for formal verification includes E Dijkstra's "weakest precondition" calculus and Harlan Mills' "functional correctness" approach which we are not going to discuss here. The following are the goals of the formal specification:

- **Removal of ambiguity:** The formal syntax of a specification language enables requirements or design to be interpreted in only one way, eliminating the ambiguity that often occurs when using natural language. Removes ambiguity and encourages greater rigour in the early stages of the software engineering process.
- **Consistency:** Facts stated in one place should not be contradicted in another. This is ensured by mathematically proving that initial facts can be formally mapped (by inference) into later statements later in the specification.
- **Completeness:** This is difficult, even when using formal methods. Some aspects of a system may be left undefined by mistake or by intention. It is impossible to consider every operational scenario in a large, complex system.

4.5 APPLICATION AREAS

Formal methods can be used to specify aspects of a system other than functionality. Software safety and security are other areas where formal methods are sometimes applied in practice. The benefits of proving that unsafe states will not arise, or security will not be violated, can justify the cost of complete formal verifications of the relevant portions of a software system. Formal methods can deal with many other areas of concern to software engineers, but have not been much used, other than in research organisations, for dealing with issues unrelated to functionality, safety, and security. Areas in which researchers are exploring formal methods include fault tolerance, response time, space efficiency, reliability, human factors, and software structure dependencies. The following are the Formal specifications application areas:

- safety critical systems
- security systems
- The definition of standards
- Hardware development
- Operating systems
- Transaction processing systems
- Anything that is hard, complex, or critical.

Let us see the drawbacks of the formal specifications in the next section.

4.6 LIMITATIONS OF FORMAL SPECIFICATION USING FORMAL METHODS

Some problems exist, however formal specification focuses primarily on function and data. Timing, control, and behavioural aspects of a problem are more difficult to represent. In addition, there are elements of a problem (e.g., HCI, the human-machine interface) that are better specified using graphical techniques. Finally, a specification using formal methods is more difficult to learn than other analysis methods. For this reason, it is likely that formal mathematical specification techniques will be incorporated into a future generation of CASE tools. When this occurs, mathematically-based specification may be adopted by a wider segment of the software engineering community. The following are the major drawbacks:

Cost: It can be almost as costly to do a full formal specification as to do all the coding. The solution for this is, don't formally specify everything, just the subsystems you need to (hard, complex, or critical).

Complexity: Not everybody can read formal specifications (especially customers and users). The solution is to provide a very good and detailed documentation. No one wants to read pure formal specifications–formal specifications should always be interspersed with natural language (e.g. English) explanation.

Deficiencies of Less Formal Approaches: The methods of structured and object-oriented design discussed previously make heavy use of natural language and a variety of graphical notations. Although careful application of these analysis and design methods can and does lead to high–quality software, sloppiness in the application of these methods can create a variety of problems which are:

- **Contradictions**: statements that are at variance with one another.
- **Ambiguities** : statements that can be interpreted in a number of ways
- **Vagueness** : statements that lack precision, and contribute little information.
- **Incomplete statements** : a description is not functionally complete.
- **Mixed levels of abstraction** : statements with high and low levels of detail are interspersed, which can make it difficult to comprehend the functional architecture.

This method gained popularity among the software developers who must build safety-critical software like aircraft avionics and medical devices. But the applicability in the business environment has not clicked.

Let us study the Cleanroom Software Engineering in the next section.

☞ Check Your Progress 1

1) What are Formal methods?

……

……

2) What are the main parts of the Formal methods?

……

……

4.7 CLEANROOM SOFTWARE ENGINEERING

Cleanroom software engineering is an engineering and managerial process for the development of high-quality software with certified reliability. Cleanroom was originally developed by Dr. Harlan Mills. The name "Cleanroom" was taken from the electronics industry, where a physical clean room exists to prevent introduction of defects during hardware fabrication. It reflects the same emphasis on defect

prevention rather than defect removal, as well as Certification of reliability for the intended environment of use. It combines many of the formal methods and software quality methods we have studied so far. The focus of Cleanroom involves moving from traditional software development practices to rigorous, engineering-based practices.

Cleanroom software engineering yields software that:

- Is correct by mathematically sound design
- Is certified by statistically-valid testing
- Reduces the cycle time results from an incremental development strategy and the avoidance of rework
- Is well-documented
- Detects errors as early as possible and reduces the cost of errors during development and the incidence of failures during operation; thus the overall life cycle cost of software developed under Cleanroom can be expected to be far lower than the industry average.

The following principles are the foundation for the Cleanroom-based software development:

Incremental development under statistical quality control (SQC): Incremental development as practiced in Cleanroom provides a basis for statistical quality control of the development process. Each increment is a complete iteration of the process, and measures of performance in each increment (feedback) are compared with pre-established standards to determine whether or not the process is "in control." If quality standards are not met, testing of the increment ceases and developers return to the design stage.

Software development based on mathematical principles: In Cleanroom software engineering development, the key principle is that, a computer program is an expression of a mathematical function. The Box Structure Method is used for specification and design, and functional verification is used to confirm that the design is a correct implementation of the specification. Therefore, the specification must define that function before design and functional verification can begin. Verification of program correctness is performed through team review based on correctness questions. There is no execution of code prior to its submission for independent testing.

Software testing based on statistical principles: In Cleanroom, software testing is viewed as a statistical experiment. A representative subset of all possible uses of the software is generated, and performance of the subset is used as a basis for conclusions about general operational performance. In other words, a "sample" is used to draw conclusions about a "population." Under a testing protocol that is faithful to the principles of applied statistics, a scientifically valid statement can be made about the expected operational performance of the software in terms of reliability and confidence.

4.8 CONVENTIONAL SOFTWARE ENGINEERING MODELS Vs. CLEANROOM SOFTWARE ENGINEERING MODEL

Cleanroom has been documented to be very effective in new development and reengineering like whole systems or major subunits. The following points highlight the areas where Cleanroom affects or differs from more conventional software development engineering practice:

Small team of software engineers: A Cleanroom project team is small, typically six to eight qualified professionals, and works in an organised way to ensure the intellectual control of work in progress. The composition of the Cleanroom Process Teams are as follows:

- **Specification team** develops and maintains the system specification
- **Development team** develops and verifies software
- **Certification team** develops set of statistical tests to exercise software after development and reliability growth models used to assess reliability.

Time allocation across the life cycle phases: Because one of the major objectives of Cleanroom is to prevent errors from occurring, the amount of time spent in the design phase of a Cleanroom development is likely to be greater than the amount of time traditionally devoted to design.

Existing organisational practices: Cleanroom does not preclude using other software engineering techniques as long as they are not incompatible with Cleanroom principles. Implementation of the Cleanroom method can take place in a gradual manner. A pilot project can provide an opportunity to "tune" Cleanroom practices to the local culture, and the new practices can be introduced as pilot results to build confidence among software staff.

The following are some of the examples of the projects implemented using the Cleanroom engineering method:

- IBM COBOL/SF product
- Ericsson OS-32 operating system project
- USAF Space Command and Control Architectural Infrastructure (SCAI) STARS Demonstration Project at Peterson Air Force Base in Colorado Springs, Colorado.
- US Army Cleanroom project in the Tank-automotive and Armaments Command at the U.S. Army Picatinny Arsenal.

4.9 CLEANROOM SOFTWARE ENGINEERING PRINCIPLES, STRATEGY AND PROCESS OVERVIEW

This software development is based on mathematical principles. It follows the box principle for specification and design. Formal verification is used to confirm correctness of implementation of specification. Program correctness is verified by team reviews using questionnaires. Testing is based on statistical principles. The test cases are randomly generated from the usage model –failure data is interpreted using statistical models. The following is the phase-wise strategy followed for Cleanroom software development.

Increment planning: The project plan is built around the incremental strategy.

Requirements gathering: Customer requirements are elicited and refined for each increment using traditional methods.

Box structure specification: Box structures isolate and separate the definition of behaviour, data, and procedures at each level of refinement.

Formal design: Specifications (black-boxes) are iteratively refined to become architectural designs (state-boxes) and component-level designs (clear boxes).

Correctness verification: Correctness questions are asked and answered, formal mathematical verification is used as required.

Code generation, inspection, verification: Box structures are translated into program language; inspections are used to ensure conformance of code and boxes, as well as syntactic correctness of code; followed by correctness verification of the code.

Statistical test planning: A suite of test cases is created to match the probability distribution of the projected product usage pattern.

Statistical use testing: A statistical sample of all possible test cases is used rather than exhaustive testing.

Certification: Once verification, inspection, and usage testing are complete and all defects removed, the increment is certified as ready for integration.

Figure 4.1 depicts the Cleanroom software engineering development overview:

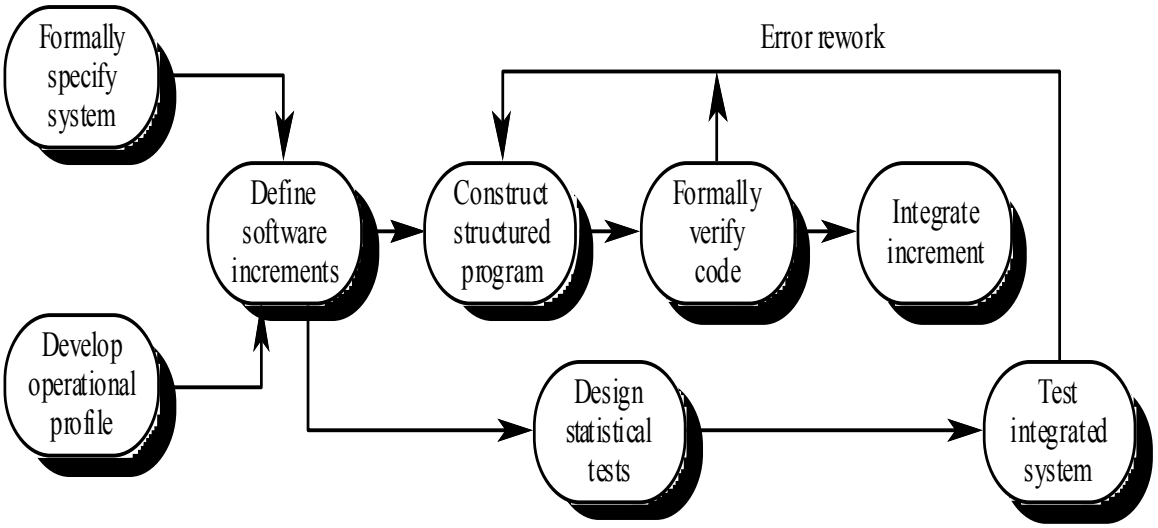

Figure 4.1: Overview of Cleanroom Software Engineering Development

4.10 LIMITATIONS OF CLEANROOM ENGINEERING

The following are some of the limitations of the Cleanroom software engineering method.

- Some people believe cleanroom techniques are too theoretical, too mathematical, and too radical for use in real software development.
- Relies on correctness verification and statistical quality control rather than unit testing (a major departure from traditional software development).
- Organisations operating at the ad hoc level of the Capability Maturity Model do not make rigorous use of the defined processes needed in all phases of the software lifecycle.

4.11 SIMILARITIES AND DIFFERENCES BETWEEN CLEANROOM AND OO PARADIGM

The following are the similarities and the differences between the Cleanroom software engineering development and OO software engineering paradigm.

Similarities

- Lifecycle - both rely on incremental development
- Usage - cleanroom usage model similar to OO use case
- State Machine Use - cleanroom state box and OO transition diagram
- Reuse - explicit objective in both process models

Key Differences

- Cleanroom relies on decomposition and OO relies on composition
- Cleanroom relies on formal methods while OO allows informal use case definition and testing
- OO inheritance hierarchy is a design resource whereas cleanroom usage hierarchy is system itself

- OO practitioners prefer graphical representations while cleanroom practitioners prefer tabular representations
- Tool support is good for most OO processes, but usually tool support is only found in cleanroom testing, not design.

Let us study the component based software engineering in the next section.

☞ Check Your Progress 2

1) What is Cleanroom Engineering and what are its objectives?

……

……

2) What is Statistical Process Control?

……

……

3) What are the benefits of Cleanroom software engineering?

……

……

……

4) What are the limitations of Cleanroom software engineering?

……

……

4.12 SOFTWARE REUSE AND ITS TYPES

In most engineering disciplines, systems are designed by composition (building system out of components that have been used in other systems). Software engineering has focused on custom development of components. To achieve better software quality, more quickly, at lower costs, software engineers are beginning to adopt *systematic reuse* as a design process. The following are the types of software reuse:

- **Application System Reuse**
  - reusing an entire application by incorporation of one application inside another (COTS reuse)
  - development of application families (e.g., MS Office)
- **Component Reuse**
  - components (e.g., subsystems or single objects) of one application reused in another application
- **Function Reuse**
  - reusing software components that implement a single well-defined function.

Let us take into consideration the ***Component Reuse*** type and the development process of the components in the following sections:

"Suppose you have purchased a Television. Each component has been designed to fit a specific architectural style – connections are standardised, a communication protocol has been pre-established. Assembly is easy because you don't have to build the system from hundreds of discrete parts. Component-based software engineering (CBSE) strives to achieve the same thing. A set of pre-built, standardised software components are made available to fit a specific architectural style for some application domain. The application is then assembled using these components, rather than the "discrete parts" of a conventional programming language. Components may be

constructed with the explicit goal to allow them to be generalised and reused. Also, component reusability should strive to reflect stable domain abstractions, hide state representations, be independent (low coupling) and propagate exceptions via the component interface".

4.13 WHY COMPONENT BASED SOFTWARE ENGINEERING?

The goal of component-based software engineering is to increase the productivity, quality, and decrease time-to-market in software development. One important paradigm shift implied here is to build software systems from standard components rather than "reinventing the wheel" each time. This requires thinking in terms of system families rather than single systems.

CBSE uses Software Engineering principles to apply the same idea as OOP to the whole process of designing and constructing software systems. It focuses on *reusing* and *adapting* existing components, as opposed to just coding in a particular style. CBSE encourages the composition of software systems, as opposed to programming them. Two important aspects of the question are:

- Firstly, several underlying technologies have matured that permit building components and assembling applications from sets of those components. Object oriented and Component technology are examples – especially standards such as CORBA.
- Secondly, the business and organisational context within which applications are developed, deployed, and maintained has changed. There is an increasing need to communicate with legacy systems, as well as constantly updating current systems. This need for new functionality in current applications requires technology that will allow easy additions.

Software Component and its types

A software component is a nontrivial, independent, and replaceable part of a system that fulfils a clear function in the context of a well-defined architecture. In some ways, this is similar to the description of an object in OOP. Components have an interface. They employ inheritance rules. Components can be characterised based on their use in the CBSE process. One category is *commercial off-the-shelf* (or COTS) components. These are components that can be purchased, pre-built, with the disadvantage that there is (usually) no source code available, and so the definition of the use of the component given by the manufacturer, and the services which it offers, must be relied upon or thoroughly tested, as they may or may not be accurate. The advantage, though, is that these types of components should (in theory) be more robust and adaptable, as they have been used and tested (and reused and retested) in many different applications.

In addition to COTS components, the CBSE process yields:

- qualified components
- adapted components
- assembled components
- updated components.

4.14 COMPONENT BASED SOFTWARE ENGINEERING (CBSE) PROCESS

CBSE is in many ways similar to conventional or object-oriented software engineering. A software team establishes requirements for the system to be built using conventional requirements elicitation techniques. An architectural design is established. Here though, the process differs. Rather than a more detailed design task,

the team now examines the requirements to determine what subset is directly amenable to *composition*, rather than *construction*.

For each requirement, we should question:

- Whether any commercial off-the-shelf (COTS) components available to implement the requirement?
- Whether internally developed reusable components available to implement the requirement?
- Whether interfaces for available components compatible within the architecture of the system to be built?

The team will attempt to modify or remove those system requirements that cannot be implemented with COTS or in-house components. This is not always possible or practical, but reduces the overall system cost and improves the time to market of the software system. It can often be useful to prioritise the requirements, or else developers may find themselves coding components that are no longer necessary as they have been eliminated from the requirements already.

The CBSE process identifies not only candidate components but also qualifies each component's interface, adapts components to remove architectural mismatches, assembles components into selected architectural style, and updates components as requirements for the system change.

Two processes occur in parallel during the CBSE process. These are:

- Domain Engineering
- Component Based Development.

4.14.1 Domain Engineering

This aims to identify, construct, catalogue, and disseminate a set of software components that have applicability to existing and future software in a particular application domain. An application domain is like a *product family* – applications with similar functionality or intended functionality. The goal is to establish a mechanism by which software engineers can share these components in order to reuse them in future systems. As defined by Paul Clements, *domain engineering is about finding commonalities among systems to identify components that can be applied to many systems and to identify program families that are positioned to take fullest advantage of those components.*

Some examples of application domains are as follows:

- Air traffic control systems
- Defence systems
- Financial market systems.

Domain engineering begins by identifying the domain to be analysed. This is achieved by examining existing applications and by consulting experts of the type of application you are aiming to develop. A *domain model* is then realised by identifying operations and relationships that recur across the domain and therefore being candidates for reuse. This model guides the software engineer to identify and categorise components, which will be subsequently implemented.

One particular approach to domain engineering is *Structural Modeling*. This is a pattern-based approach that works under the assumption that every application domain has repeating patterns. These patterns may be in function, data, or behaviour that have reuse potential. This is similar to the pattern-based approach in OOP, where a particular style of coding is reapplied in different contexts. An example of *Structural Modeling* is in Aircraft avionics: The systems differ greatly in specifics, but all modern software in this domain has the same structural model.

4.14.2 Component Based Development

There are three stages in this process. These are:

- Qualification,
- Adaptation (also known as wrapping),
- Composition (all in the context of components).

Component qualification: examines reusable components. These are identified by characteristics in their interfaces, i.e., the services provided, and the means by which consumers access these services. This does not always provide the whole picture of whether a component will fit the requirements and the architectural style. This is a process of discovery by the Software Engineer. This ensures a candidate component will perform the function required, and whether it is compatible or adaptable to the architectural style of the system. The three important characteristics looked at are *performance, reliability and usability*.

Component Adaptation is required because very rarely will components integrate immediately with the system. Depending on the component type(e.g., COTS or in-house), different strategies are used for adaptation (also known as *wrapping*). The most common approaches are:

- **White box wrapping:** The implementation of the component is directly modified in order to resolve any incompatibilities. This is, obviously, only possible if the source code is available for a component, which is extremely unlikely in the case of COTS.
- **Grey box wrapping:** This relies on the component library providing a component extension language or API that enables conflicts to be removed or masked.
- **Black box wrapping:** This is the most common case, where access to source code is not available, and the only way the component can be adapted is by pre / post-processing at the interface level.

It is the job of the software engineer to determine whether the effort required to wrap a component adequately is justified, or whether it would be "cheaper" (from a software engineering perspective) to engineer a custom component which removes these conflicts. Also, once a component has been adapted it is necessary to check for compatibility for integration and to test for any unexpected behaviour which has emerged due to the modifications made.

Component Composition: integrated the components (whether they are qualified, adapted or engineered) into a working system. This is accomplished by way of an infrastructure which is established to bind the components into an operational system. This infrastructure is usually a library of specialised components itself. It provides a model for the coordination of components and specific services that enable components to coordinate with one another and perform common tasks.

There are many mechanisms for creating an effective infrastructure, in order to achieve component composition, but in particular there is the *Data Exchange Model*, which allows users and applications to interact and transfer data (an example of which is *drag-and-drop* and *cut-and-paste*). These types of mechanisms should be defined for all reusable components. Also important is the *Underlying object model*. This allows the interoperation of components developed in different programming languages that reside on different platforms.

4.15 COMPONENT TECHNOLOGIES AVAILABLE

Common methods of achieving this is by the use of technologies such as *Corba* which allows components to communicate remotely and transparently over a network, *Java*

Beans and Microsoft COM which allows components from different vendors to work together within Windows. Sometimes it is necessary to bridge these technologies (e.g., COM/Corba) to facilitate integration of software developed by different vendors, or for integration with legacy systems.

It must be noted that as the requirements for a system change, components may need to be updated. Usually, some sort of *change control procedure* should be in place to facilitate configuration management, and to ensure updates to the components follow the architectural style of the system being developed.

4.16 CHALLENGES FOR CBSE

CBSE is facing many challenges today, some of these are summarised as follows:

- **Dependable systems and CBSE:** The use of CBD in safety-critical domains, real-time systems, and different process-control systems, in which the reliability requirements are more rigorous, is particularly challenging. A major problem with CBD is the limited possibility of ensuring the quality and other non-functional attributes of the components and thus our inability to guarantee specific system attributes.
- **Tool support:** The purpose of Software Engineering is to provide practical solutions to practical problems, and the existence of appropriate tools is essential for a successful CBSE performance. Development tools, such as Visual Basic, have proved to be extremely successful, but many other tools are yet to appear – component selection and evaluation tools, component repositories and tools for managing the repositories, component test tools, component-based design tools, run-time system analysis tools, component configuration tools, etc. The objective of CBSE is to build systems from components simply and efficiently, and this can only be achieved with extensive tool support.
- **Trusted components:** Because the trend is to deliver components in binary form and the component development process is outside the control of component users, questions related to component trustworthiness become of great importance.
- **Component certification:** One way of classifying components is to certificate them. In spite of the common belief that certification means absolute trustworthiness, it is in fact only gives the results of tests performed and a description of the environment in which the tests were performed. While certification is a standard procedure in many domains, it is not yet established in software in general and especially not for software components.
- **Composition predictability:** Even if we assume that we can specify all the relevant attributes of components, it is not known how these attributes determine the corresponding attributes of systems of which they are composed. The ideal approach, to derive system attributes from component attributes is still a subject of research. A question remains - "Is such derivation at all possible? Or should we not concentrate on the measurement of the attributes of component composites?"
- **Requirements management and component selection:** Requirements management is a complex process. A problem of requirements management is that requirements in general are incomplete, imprecise and contradictory. In an in-house development, the main objective is to implement a system which will satisfy the requirements as far as possible within a specified framework of different constraints. In component-based development, the fundamental approach is the reuse of existing components. The process of engineering requirements is much more complex as the possible candidate components are usually lacking one or more features which meet the system requirements exactly. In addition, even if some components are individually well suited to the

system, it is not necessary that they do not function optimally in combination with others in the system- or perhaps not at all. These constraints may require another approach in requirements engineering – an analysis of the feasibility of requirements in relation to the components available and the consequent modification of requirements. As there are many uncertainties in the process of component selection there is a need for a strategy for managing risks in the components selection and evolution process.

- **Long-term management of component-based systems:** As component-based systems include sub-systems and components with independent lifecycles, the problem of system evolution becomes significantly more complex. CBSE is a new approach and there is little experience as yet of the maintainability of such systems. There is a risk that many such systems will be troublesome to maintain.
- **Development models:** Although existing development models demonstrate powerful technologies, they have many ambiguous characteristics, they are incomplete, and they are difficult to use.
- **Component configurations:** Complex systems may include many components which, in turn, include other components. In many cases compositions of components will be treated as components. As soon as we begin to work with complex structures, the problems involved with structure configuration pop up.

☞ Check Your Progress 3

1) What are the benefits of reuse of software?

……

……

2) What is Commercial off the Shelf Software (COTS)?

……

……

……

4.17 REENGINEERING – AN INTRODUCTION

Over the past few years, legacy system reengineering has emerged as a business critical activity. Software technology is evolving by leaps and bounds, and in most cases, legacy software systems need to operate on new computing platforms, be enhanced with new functionality, or be adapted to meet new user requirements. The following are some of the reasons to reengineer the legacy system:

- Allow legacy software to quickly adapt to changing requirements
- Comply to new organisational standards
- Upgrade to newer technologies/platforms/paradigms
- Extend the software's life expectancy
- Identify candidates for reuse
- Improve software maintainability
- Increase productivity per maintenance programmer
- Reduce reliance on programmers who have specialised in a given software system
- Reduce maintenance errors and costs.

Reengineering applies reverse engineering to existing system code to extract design and requirements. Forward engineering is then used to develop the replacement system.

Let us elaborate the definition. Reengineering is the examination, analysis, and alteration of an existing software system to reconstitute it in a new form, and the subsequent implementation of the new form. The process typically encompasses a

combination of reverse engineering, re-documentation, restructuring, and forward engineering. The goal is to understand the existing software system components (specification, design, implementation) and then to re-do them to improve the system's functionality, performance, or implementation.

The reengineering process is depicted in *Figure 4.2*. Re-engineering starts with the code and comprehensively reverse engineers by increasing the level of abstraction as far as needed toward the conceptual level, rethinking and re-evaluating the engineering and requirements of the current code, then forward engineers using a waterfall software development life-cycle to the target system. In re-engineering, industry reports indicate that approximately 60% of the time is spent on the reverse engineering process, while 40% of the time is spent on forward engineering. Upon completion of the target system, most projects must justify their effort, showing that the necessary functionality has been maintained while quality has improved (implying improved reliability and decreased maintenance costs).

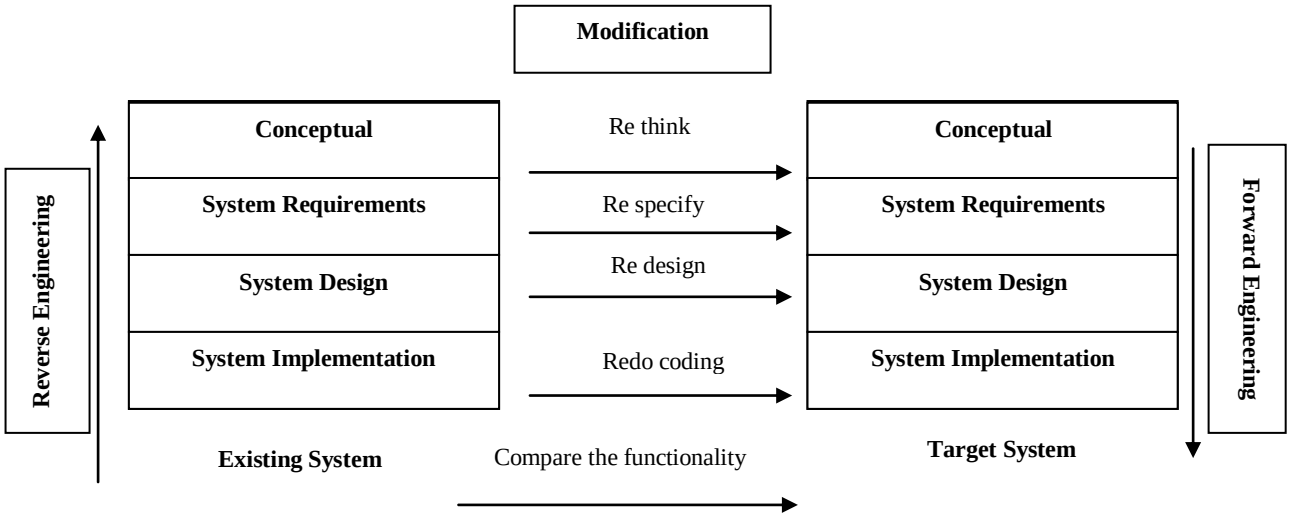

Figure 4.2: Process of Software Reengineering

4.18 OBJECTIVES OF REENGINEERING

The objectives of a specific software reengineering activity are determined by the goals of the clients and users of the system. However, there are two general reengineering objectives as shown below:

- **Improve quality:** Typically, the existing software system is of low quality, due to many modifications. User and system documentation is often out-of-date or no longer in existence. Re-engineering is intended to improve software quality and to produce current documentation. Improved quality is needed to increase reliability, to improve maintainability, to reduce the cost of maintenance, and to prepare for functional enhancement. Object-oriented technology may be applied as a means for improving maintainability and reducing costs.
- **Migration:** Old working software may still meet users' needs, but it may be based on hardware platforms, operating systems, or languages that have become obsolete and thus may need to be re-engineered, transporting the software to a newer platform or language. Migration may involve extensive redesign if the new supporting platforms and operating systems are very different from the original, such as the move from a mainframe to a network-based computing environment.

4.19 SOFTWARE REENGINEERING LIFE CYCLE

In this section, we present an evolutionary view of the software reengineering life-cycle:

Requirements analysis phase: It refers to the identification of concrete reengineering goals for a given software. The specification of the criteria should be specified and illustrated in the new reengineered system (for example, faster performance). Violations that need to be repaired are also identified in this phase.

Model analysis phase: This refers to documenting and understanding the architecture and the functionality of the legacy system being reengineered. In order to understand and to transform a legacy system, it is necessary to capture its design, its architecture and the relationships between different elements of its implementation. As a consequence, a preliminary model is required in order to document the system and the rationale behind its design. This requires reverse engineering the legacy system in order to extract design information from its code.

Source code analysis phase: This phase refers to the identification of the parts of the code that are responsible for violations of requirements originally specified in the system's analysis phase. This task encompasses the design of methods and tools to inspect, measure, rank, and visualize software structures. Detecting error prone code that deviates from its initial requirement specifications requires a way to measure where and by how much these requirements are violated. Problem detection can be based on a static analysis of the legacy system (i.e., analysing its source code or its design structure), but it can also rely on a dynamic usage analysis of the system (i.e., an investigation of how programs behave at run-time).

Remediation phase: It refers to the selection of a target software structure that aims to repair a design or a source code defect with respect to a target quality requirement. Because legacy applications have been evolved in such a way that classes, objects, and methods may heavily depend on each other, a detected problem may have to be decomposed into simpler sub-problems.

Transformation phase: This consists of physically transforming software structures according to the remediation strategies selected previously. This requires methods and tools to manipulate and edit software systems, re-organise and re-compile them automatically, debug and check their consistency, and manage different versions of the software system being reengineered.

Evaluation phase refers to the process of assessing the new system as well as establishing and integrating the revised system throughout the corporate operating environment. This might involve the need for training and possibly the need for adopting a new improved business process model.

Such a reengineering life-cycle yields a reengineering process. First, the source code is represented as an Abstract Syntax Tree. The tree is further decorated with annotations that provide linkage, scope, and type information. Once software artifacts have been understood, classified and stored during the reverse engineering phase, their behaviour can be readily available to the system during the forward engineering phase. Then, the forward engineering phase aims to produce a new version of a legacy system that operates on the target architecture and aims to address specific maintainability or performance enhancements. Finally, we use an iterative procedure to obtain the new migrant source code by selecting and applying a transformation which leads to performance or maintainability enhancements. The transformation is selected from the soft-goal interdependency graphs. The resulting migrant system is then evaluated and the step is repeated until the specific quality requirements are met.

☞ Check Your Progress 4

1) What is Software Reengineering and what are its objectives?

……

……

2) What is Reverse Engineering?

……

……

4.20 SUMMARY

In this unit, we have gone through the Formal Methods, Cleanroom software engineering, Component Based Software Engineering and the Software Reengineering process.

Formal methods promise to yield benefits in quality and productivity. They provide an exciting paradigm for understanding software and its development, as well as a set of techniques for use by software engineers. Formal methods can provide more precise specifications, better internal communication, and an ability to verify designs before executing them during test, higher quality and productivity. To get their full advantages, formal methods should be incorporated into a software organisation's standard procedures. Software development is a social process, and the techniques employed need to support that process. How to fully fit formal methods into the lifecycle is not fully understood. Perhaps there is no universal answer, but only solutions that vary from organisation to organisation.

Harlan Mills has developed the Cleanroom methodology, which is one approach for integrating formal methods into the lifecycle. The Cleanroom approach combines formal methods and structured programming with Statistical Process Control (SPC), the spiral lifecycle and incremental releases, inspections, and software reliability modeling. It fosters attitudes, such as emphasising defect prevention over defect removal, that are associated with high quality products in non-software fields.

Component Based Software Engineering introduced major changes into design and development practices, which introduces extra cost. Software engineers need to employ new processes and ways of thinking – this, too, can introduce extra cost in training and education. However, initial studies into the impact of CBSE on product quality, development quality and cost, show an overall gain, and so it seems likely that continuing these practices in the future will improve software development. It is still not clear how exactly CBSE will mature, but it is clear that it aids in the development of today's large-scale systems, and will continue to aid in the development of future systems, and is the perfect platform for addressing the requirements of modern businesses**.**

Software reengineering is a very wide-ranging term that encompasses a great many activities. As the name suggests, software reengineering is applied to existing pieces of software, in an after-the-fact fashion, via the reapplication of an engineering process.

4.21 SOLUTIONS / ANSWERS

Check Your Progress 1

1) Formal methods is that area of computer science that is concerned with the application of mathematical techniques to the design and implementation of computer software.

2) Following are the main parts to formal methods:

   i) **Formal specification:** Using mathematics to specify the desired properties of a computer system.

   ii) **Formal verification:** Using mathematics to prove that a computer system satisfies its specification.

iii) **Automated programming:** Automating the process of program generation.

Check Your Progress 2

1) The Cleanroom methodology is an iterative, life-cycle approach focused on software quality, especially reliability. Begun by Harlan Mills, it combines formal specifications, structured programming, formal verifications, formal inspections, functional testing based on random selection of test data, software reliability measurement, and Statistical Process Control (SPC) in an integrated whole. The Cleanroom approach fosters attitudes, such as emphasising defect prevention over defect removal, that are associated with high quality products in fields other than software.

2) Statistical Process Control (SPC) is commonly used in manufacturing, involves continuous process monitoring and improvement to reduce the variance of outputs and to ensure that the process remains under control.

3) Benefits of Cleanroom software engineering include significant improvements in *correctness*, *reliability*, and *understandability*. These benefits usually translate into a reduction in field-experienced product failures, reduced cycle time, ease of maintenance, and longer product life.

4) Cleanroom techniques are too theoretical, too mathematical, and too radical for use in real software development. They rely on correctness verification and statistical quality control rather than unit testing (a major departure from traditional software development). Organisations operating at the ad hoc level of the Capability Maturity Model do not make rigorous use of the defined processes needed in all phases of the software life cycle.

Check Your Progress 3

1) Benefits of software reuse are:

- **Increased Reliability**: components already exercised in working systems
- **Reduced Process Risk**: Less uncertainty in development costs
- **Effective Use of components**: Reuse components instead of people
- **Standards Compliance**: Embed standards in reusable components
- **Accelerated Development**: Avoid custom development and speed up delivery.

2) COTS systems usually have a complete applications library and also offers an applications programming interface (API). These are helpful in building large systems by integrating COTS components is a viable development strategy for some types of systems (e.g., E-commerce or video games).

Check Your Progress 4

1) Software reengineering is a process that aims to either

   i) Improve understanding of a piece of software, or
   ii) Prepare for an improvement in the software itself (e.g., increasing its maintainability or reusability).

As the name suggests software reengineering is applied to existing pieces of software, in an after-the-fact fashion, via the reapplication of an engineering process. Software reengineering is a very wide-ranging term that encompasses a great many activities. For example, software reengineering could involve refactoring a piece of software, redocumenting it, reverse engineering it or changing its implementation language.

2) Reverse engineering is a process which is used to improve the understanding of a program. So-called program comprehension is crucial to maintenance. Approximately 50% of the development time of programmers is spent understanding the code that they are working on. Reverse engineering often involves the production of diagrams. It can be used to abstract away irrelevant

implementation and design details to recover the original requirements of the program.

4.22 FURTHER READINGS

1) *Software Engineering, Sixth Edition, 2001*, Ian Sommerville; Pearson Education.

2) *Software Engineering – A Practitioner's Approach*, Roger S. Pressman; McGraw-Hill International Edition.

3) *Component Software –Beyond Object-Oriented Programming,1998,* Szyperski C; Addison-Wesley.

4) Software Engineering, Schaum's Outlines, 2003,David Gustafson,Tata Mc Graw-Hill.

Reference websites

- **http://www.rspa.com**
- **http://www.ieee.org**

4.23 GLOSSARY

Software Reengineering: The examination and alteration of an existing subject system to reconstitute it in a new form. This process encompasses a combination of sub-processes such as reverse engineering, restructuring, redocumentation, forward engineering, and retargeting.

Reverse Engineering: The engineering process of understanding, analysing, and abstracting the system to a new form at a higher abstraction level.

Forward Engineering: Forward engineering is the set of engineering activities that consume the products and artifacts derived from legacy software and new requirements to produce a new target system.

Data Reengineering: Tools that perform all the reengineeering functions associated with source code (reverse engineering, forward engineering, translation, redocumentation, restructuring / normalisation, and retargeting) but act upon data files.

Redocumentation: The process of analysing the system to produce support documentation in various forms including users manuals and reformatting the systems' source code listings.

Restructuring: The engineering process of transforming the system from one representation form to another at the same relative abstraction level, while preserving the subject system's external functional behavior.

Retargeting: The engineering process of transforming and hosting / porting the existing system with a new configuration.

Source Code Translation: Transformation of source code from one language to another or from one version of a language to another version of the same language (e.g., going from COBOL-74 to COBOL-85).

Business Process Reengineering (BPR): The fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service, and speed.