BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 6

DCA3245

SOFTWARE PROJECT MANAGEMENT

Unit 13

Software Re-Engineering

Table of Contents

1. INTRODUCTION

In the previous unit, we have discussed various software testing techniques. The software engineering practices as we have discussed is focused on continuous improvement to software development process. These improvement efforts have resulted in introduction of the practices like structured approach in requirement gathering, design, object oriented principles etc. But we have to think outside these practices when we considering the legacy code. Legacy code is a huge set of software projects that were carried out from 1950's and adopting the newer technologies may not be feasible option. But we cannot totally discard the legacy system as well because of the huge amount of information and investment they carry. Hence from the maintainability, continued performance of the legacy system we turn into Software Re-engineering practices.

In this unit, we shall discuss various software reengineering techniques.

1.1 Objectives:

After studying this unit, you should be able to:

- *Explain various problems associated with software maintenance*
- *Discuss software reengineering*
- *Describe business process reengineering*
- *Explain software reengineering process model*
- *Discuss various technical problems associated with reengineering*
- *Explain various terminologies in software engineering*

2. SOFTWARE MAINTENANCE PROBLEMS

Software reengineering is required for the maintenance of legacy applications. In order to help with reengineering, specific tools are available that will try to capture the design and implementation information from the complicated and convoluted legacy code. Using the tools that help reengineering we can generate the documentation on the legacy code and in some cases the data and the legacy software can be ported onto a new environment or platform with new language.

The software reengineering should serve the purpose of bridging the gap between the source legacy system and the targeted maintenance environment. This means the reengineering should be based on organizational goals and the reengineering project must go in line with organization's strategic plan.

Other reasons of opting for reengineering include:

- Allow legacy applications to adapt to changing requirements in quick time
- Satisfy organizational guidelines on enterprise level architecture to be followed
- Upgrade to newer technologies and paradigms
- Extend the software's life expectancy
- Determine the code or modules that can be reused
- Improve software maintainability
  - improve productivity of software engineer in charge of maintenance
  - Reduce the dependency of organization on resources holding key software skill
  - Improve the software maintenance statistics by reduced errors and cost.

Software maintenance (as defined by ANSI/IEEE-STD-729-1983) is the modification of a software product after delivery to correct errors, improve performance (or other attributes), or adapt to new requirements. Software maintenance is a tough job that has lot of associated problems with it as listed below:

Software errors can be very expensive: Any error that is left undetected in the released product will show up soon after the end customers start using the product. These errors might lead to various problems to the organization like financial loss, dent in brand image of the company, loss of customer base and loyalty to the company etc. Hence managing through scenarios like these with software that has errors will be a tough ask. The recent survey conducted also confirms the fact that top ten most expensive errors are maintenance errors.

Software maintenance is very costly: Maintaining the software product is a costly affair as we need to keep aside a substantial part of organization's budget for IT department. This fact is also supported by various statics available on IT or software maintenance spending from World's leading companies from their corporate budget allocation.

- **Maintenance personnel are getting scarce:** Another problem with the software maintenance is the dependency on the resources who maintain the complicated applications. As the legacy systems are complicated and would have undergone large number of changes over the years, understanding the functionality and changing any functionality will be a very tough job for any resource. Also most of the legacy applications are not having proper documentation on their internals. This makes it difficult to train any new resources on the application. Hence the resources who are currently working on the maintenance are largely in demand; any form of attrition will put maintenance at risk as finding the replacements for such resources is extremely difficult job. Other than these problems the maintenance typically contains issues like ripple effects on errors that are not known to the maintenance team, poor documentation, etc.

All these problems are further compounded when the budget allocation of the organization shrinks down. The management will have to take a call on whether to increase the budget on the software development to improve the quality of the deliverables or to increase the software maintenance cost alone based stability factor of the application being maintained.

Types of Software Maintenance:

There are four distinct kinds of software maintenance, each serving a unique function. At certain points in time, a given piece of software may require one, two, or perhaps all three forms of upkeep.All maintenance activities are classified into four categories. They are-

- Corrective maintenance
- Adaptive maintenance
- Preventive maintenance
- Perfective maintenance

1. Corrective Maintenance : Corrective Maintenance: The goal of corrective maintenance is to fix any lingering bugs regardless of where they originated (in the requirements, design, code, testing, or documentation, for example).This refer to modifications initiated by defects in the software. Corrective maintenance of a software product done after delivery for correction of problems detected. Here the system responds to equipment fault
2. Adaptive Maintenance: Software maintenance includes adapting programmes to new conditions. Maintaining software that can adapt to new technologies and evolving business needs is called adaptive software maintenance. The software foundation is the primary emphasis of this software maintenance style. This is done so that the programme will continue to function properly on emerging operating systems, platforms, and hardware.Adaptive software maintenance involves making modifications to the programme in response to external factors.
3. Preventive Maintenance: Software vulnerability can be avoided with the help of a preventative maintenance solution. Software upgrades that are defined as "preventive maintenance" are made with future security in mind. It's done so that the product can't be hacked or have its code changed in any way. It is much simpler to scale your code and manage your legacy system if you practise preventative maintenance.
4. Perfective Maintenance: Perfective maintenance involves making functional enhancements to the system in addition to the activities to increase the system's performance even when the changes have not been suggested by faults. Perfective Maintenance is a process of modifying all elements, functionalities, and abilities to enhance system operations and performance. The software's receptiveness and

usability are solved by perfective software maintenance. It includes altering current software functionality by improving, removing, or inserting new features or functions.

Maintenance Process:

A proper maintenance plan is prepared for modifications after delivery and during transitions.The implementation processes consists of

- Preparing software and transition activities, such as the conceptualizing and creating the maintenance plan,
- Preparation for handling problems recognized during development,
- Following-up on product configuration management

If the software must be ported to another platform without any change in functionality, this process will be used and a maintenance project team is likely to be assigned to this task.

The process should itself take care of the implementation of the modification.

The process acceptance of the modification, - the modified work is confirmed with the individual who submitted the request in order to make sure the modification provided a solution.

The migration process is unique, and is not related to daily maintenance activity.

The last maintenance process is the retirement of a piece of software.

SELF-ASSESSMENT QUESTIONS - 1

1. Software maintenance is the modification of a software product after delivery to correct errors, improve performance, or adapt to new requirements. (True / False)
2. _____ Allow legacy applications to adapt to changing requirements in quick time.
3. In a recent study, the top 10 most expensive software errors were_____ errors. (Pick right option)
   a) Maintenance
   b) Analysis
   c) Design
   d) Coding

The cause for Software Breakdown:

Most software projects fail completely or partially because they don't meet all their requirements. These requirements can be the cost, schedule, quality, or requirements objectives. According to many studies, the failure rate of software projects ranges between 50% – 80%. The following might be some of the causes for the break down:

- Neglecting basic conditions
- Inadequate skills
- Ignoring operating standards
- Deterioration unchecked
- Inherent design weakness
- Lack of user participation
- Changing requirements
- Unrealistic or unarticulated project goals
- Inaccurate estimates of needed resources

- Badly defined system requirements
- Poor reporting of the project's status
- Lack of resources
- Unmanaged risks

Need for software Maintenance:

- Software evolution and maintenance optimizes the software performance by reducing errors, eliminating useless development, and applying advanced development.
- Maintenance for repairing software faults: Changing a system to correct deficiencies for meeting its requirements.
- Maintenance for making software adaptable to a different operating environment: Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
- Maintenance to enhance the system's functionality: Modifying the system to satisfy new requirements.
- The elimination of malfunctions is a top concern in software maintenance management. Finding and fixing programming mistakes is part of this procedure.
- In order to adapt to the ever-changing nature of the market, solutions have had to undergo a process of feature and function enhancement. It improves system workflow by adjusting software platforms, work patterns, hardware updates, compilers, and other factors.

Optimal Maintenance and its Parameters:

In the field of software engineering, "optimal maintenance" is the practise of managing and improving software systems in the most efficient and effective way possible to ensure their continuous success. The goal is to maximise the software's value and lifespan by making strategic decisions about resource allocation, job prioritisation, and change implementation. Optimal maintenance planning tries to find financially optimized and balanced technical maintenance operations in component level to reach sufficient structural reliability, availability, serviceability, and durability for a certain period of time.

Its the procedure which is concerned with maintaining a system to maximise profit or minimise costCost functions rely on the reliability and maintainability characteristics of the system for determining the parameters to minimize

Following are the some of the common parameter for optimal maintenance :

- The cost of failure,
- The cost per time unit of "downtime",
- The cost (per time unit) of corrective maintenance,
- The cost per time unit of preventive maintenance an
- The cost of repairable system replacement

Optimal maintenance is a management strategy for software systems that seeks to strike a compromise between the pressure to constantly innovate and the limitations imposed by available resources. Getting the greatest results for the software and its stakeholders calls for meticulous planning, constant evaluation of priorities, and efficient use of resources.

Maintenance Cost:

When discussing software engineering, the term "maintenance cost" refers to the ongoing costs incurred to ensure that a software system continues to function as intended and is always up-to-date. These expenditures cover everything from staff time to hardware upgrades that keep the programme running smoothly. Organizations need to watch their maintenance spending closely to make sure their money is well spent. Long-term planning for the software's lifecycle is an important part of this process, as is prioritizing maintenance tasks based on criticality, influence on users, and business objectives.

Breaking an existing system is a very costly affair

- Maintenance costs are higher than development costs by a factor of 2 to 100.The costs occur due to both technical and non technical factors.
- An already deployed system is expensive to change. Maintenance costs increase over time with the evolution of the system

If the application of the program is defined and well understood, the system requirements may be definitive and maintenance due to changing needs minimized. Some of the maintenance cost factors are discussed below:

Team stability: Maintenance costs are reduced if the staff maintain the system is also stable.

Contractual responsibility: Generally the developers of a system have no contractual responsibility for maintenance hence there is lack of incentives to design for future change.

Bug Fixes and Error Correction: Identifying, diagnosing, and rectifying software defects or errors that may arise during regular usage.

Updates and Enhancements:

Implementing new features, functionality improvements, or enhancements based on technical requirements and changing technology trends.

Performance Optimization:

Improving the performance and efficiency of the software, which may involve code optimization, database tuning, or infrastructure upgrades.

Security Updates:

Identifying and addressing security vulnerabilities, applying patches, and implementing measures to safeguard against cyber threats.

Hardware and Infrastructure Costs:

Maintaining and upgrading the hardware and infrastructure on which the software runs, including servers, storage, and networking equipment.

Data Migration and Conversion:

Migrating and converting data when transitioning to new versions of the software or migrating to a different platform.

Testing and Quality Assurance:

Conducting technical testing activities, including regression testing, load testing, and security testing, to ensure software reliability.

User Support and Training:

Providing user support, including helpdesk services, documentation, and training materials to assist users in effectively utilizing the software.

Regulatory Compliance:

Ensuring that the software complies with industry-specific regulations, standards, and legal requirements.

License Renewals and Subscriptions:

Renewing software licenses, subscriptions, and service agreements with vendors or third-party providers.

Documentation and Knowledge Management:

Maintaining comprehensive documentation, knowledge bases, and training materials to support end-users and ongoing development efforts.

Change Management and Version Control (non-technical):

Implementing and maintaining non-technical change management processes related to organizational and procedural changes.

Retirement and Replacement Planning:

Planning for the eventual retirement or replacement of outdated or unsupported software with newer solutions.

Project Management and Coordination:

Overseeing and coordinating non-technical aspects of maintenance projects, including resource allocation, scheduling, and stakeholder communication.

Maintenance Problem: Maintenance is inevitable the system requirements are likely to change while the system is being developed because the environment is changing. Some of the common problem that arrases in maintenance are listed below:

- Someone else has written the program.
- Developer is not available.
- There is no proper documentation.
- While designing, future changes were not considered.
- Maintenance activity not highly regarded

Cost: It's one of the biggest challenges we face in the software maintenance process. Well, it can be a major expense for businesses with budget constraints and new in the market.

Time: Software maintenance is a time-consuming process. It requires a lot of time for improvement and enhancement of the software. Hence, organizations need to spare their time to allocate dedicated resources for this purpose. And that's somewhat difficult for them.

Complexity: Going for software maintenance without a clear roadmap can become complex as it consumes time. This becomes quite difficult to maintain and manage all the different components.

Fatigue: Due to the vast number of software product changes, managing them for organizations becomes less effective. As a result, you will find many errors that eventually accumulate.

3. REDEVELOPMENT VS. REENGINEERING

As we have seen, the legacy code maintenance has long list of problems associated with it. If the problems are severe enough and the legacy code is very important and functionality provided by the legacy application is very critical to the business, then the natural option seems to be re-developing the application. But we have to consider the following points before going ahead with re-development and might have to choose reengineering in such scenarios:

- **Critical corporate knowledge is contained within the legacy software**

  Legacy application of the organization holds lot of business critical details and represents the enterprise architecture of the corporation. Further, these business rules may not be all that well documented. Hence if we now start to entirely re-develop the application, the team might not have enough information behind the expected behavior and end up implementing a wrong system causing huge loss to the organization.

- **Legacy software is a valuable asset**

  There has been a huge investment on legacy applications since the development started from around 1950's. It is estimated that the investment of COBOL code, which was the major programming language during earlier days of software application runs into between $640 billion to $1.6 trillion. Around 80% of the coding of legacy application is done using COBOL. Discarding legacy applications means doing away with such huge valued asset and not considered as good option.

- **Reusable, reengineered software costs much less than redeveloped code**

  The advantage reengineering offers is that it aims at reusing the code which in a way ensures that investments made earlier on legacy code is paying off by some means. In other words, cost of reengineered code reusing the existing code base is much lesser than redevelopment.

With all these reasons reengineering seems as the viable alternate for redevelopment. Again the judgment on whether to go with redevelopment or reengineering depends entirely on organization's goals and strategies and decided considering above points.

4. BUSINESS PROCESS REENGINEERING (BPR)

With the advent of changing market dynamics a new business strategy is coming into practice – it is called as Business Process Reengineering (BPR). BPR is not an equivalent term to be used with software reengineering. Both are to be considered as separate entities. However in most of the cases the requirements of BPR will be the driving factors behind software reengineering, hence it would be useful to know the principle definition of BPR.

"Business Process Reengineering is 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." – Michael Hammer and James Champy

The business process reengineering mainly focusses on reorganizing the entities of the business model to meet the goal of increased performance and profitability to the organization. Traditionally the organizations are vertically structured with each unit working as individual entity in the organization. For example in a retail organization the marketing, inventory, procurement, HR etc. can be considered as individual verticals. The problem with vertical approach is that when new requirements come up, each unit has to work from top to bottom to realize the changes in its group and the similar development will be needed in other verticals of the same organization. This is waste of time and also increases cost to implement the change. The BPR suggests this wastage of time and money should be stopped through reorganizing the units so that they are cross functional to avoid the repentance of the work.

What is the role of software reengineering?

Software reengineering can help to achieve the goals of BPR. When the legacy systems were developed, naturally they incorporate the traditional vertical approach of business model. As stated earlier, software reengineering tools try to capture the design and implementation information from the legacy code base. Using these tools we can further group this information to logical chunks giving cohesive functional units of the application that can be redeployed after necessary enhancements.

Notice that in many ways, this sounds a lot like the process of translating process-oriented software to object-oriented software. Software should reflect a meta-model of the real world. In the past, organizations attempted to force their software to conform to a structure that did not match the real world as understood by the eventual users of the software. This often caused communication problems between software designers and users. Now, with object-oriented analysis and BPR, organizations are re-aligning their software (and organizational structure) so that they correspond to real-world objects (and processes).

SELF-ASSESSMENT QUESTIONS - 2

4. Legacy application of the organization holds lot of business critical details and represents the enterprise architecture of the corporation. (True / False)
5. Around 80% of the coding of legacy application is done using _____.
6. _____ is 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. (Pick right option)
   a) Software Engineering
   b) Business Process Reengineering
   c) Software Reengineering
   d) Business Process Outsourcing

5. SOFTWARE REENGINEERING PROCESS MODEL

Reengineering takes time. It costs significant amounts of money and it absorbs resources that might be otherwise occupied on immediate concerns. For all of these reasons, reengineering is not accomplished in a few months or even a few years. Reengineering of information systems is an activity that will absorb information technology resources for many years. That's why every organization needs a pragmatic strategy for software reengineering. A workable strategy is encompassed in a reengineering process model. We'll discuss the model later in this section, but first, some basic principles. Reengineering is a rebuilding activity, and we can better understand the reengineering of information systems if we consider an analogous activity, the rebuilding of a house.

Consider the following situation.

You have purchased a house in another state. You've never actually seen the property, but you acquired it at an amazingly low price, with the warning that it might have to be completely rebuilt. How would you proceed?

- Before you can start rebuilding, it would seem reasonable to inspect the house. To determine whether it is in need of rebuilding, you (or a professional inspector) would create a list of criteria so that your inspection would be systematic.
- Before you tear down and rebuild the entire house, be sure that the structure is weak. If the house is structurally sound, it may be possible to "remodel" without rebuilding (at much lower cost and in much less time).
- Before you start rebuilding, be sure you understand how the original was built. Take a peek behind the walls. Understand the wiring, the plumbing, and the structural internals. Even if you trash them all, the insight you'll gain will serve you well when you start construction.
- If you begin to rebuild, use only the most modern, long-lasting materials. This may cost a bit more now, but it will help you to avoid expensive and time-consuming maintenance later.
- If you decide to rebuild, be disciplined about it. Use practices that will result in high quality – today and in the future.

Although these principles focus on the rebuilding of a house, they apply equally well to the reengineering of computer-based systems and applications.

To implement these principles, we apply software reengineering process model that defines six activities, shown in figure 13.1. In some cases, these activities occur in a linear sequence, but this is not always the case. For example, it may be that reverse engineering (understanding the internal workings of a program) may have to occur before document restructuring can commence.

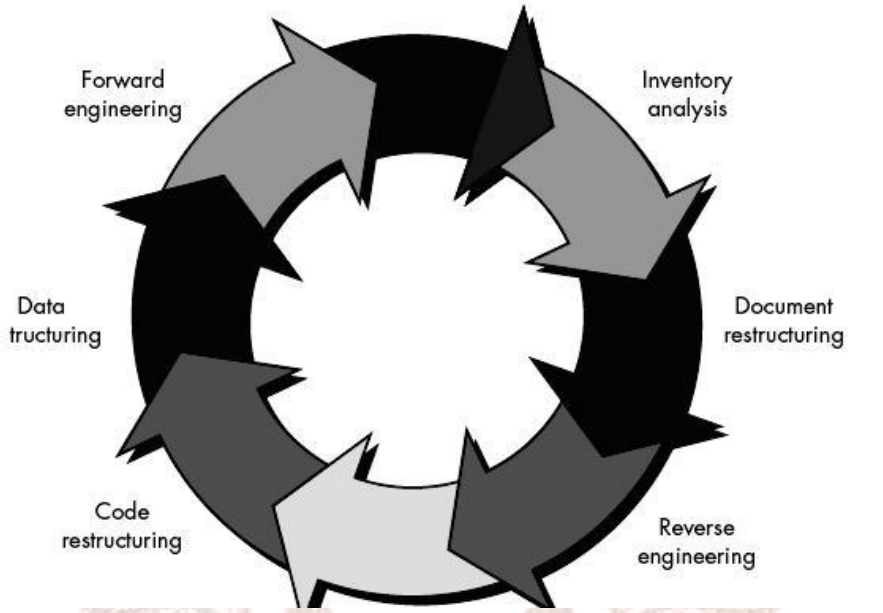

Fig. 13.1: Six Activities of Software Reengineering Process

The reengineering paradigm shown in the figure is a cyclical model. This means that each of the activities presented as a part of the paradigm may be revisited. For any particular cycle, the process can terminate after any one of these activities.

Inventory analysis: Every software organization should have an inventory of all applications. The inventory can be nothing more than a spreadsheet model containing information that provides a detailed description (e.g., size, age, business criticality) of every active application. By sorting this information according to business criticality, longevity, current maintainability, and other locally important criteria, candidates for reengineering appear. Resources can then be allocated to candidate applications for reengineering work. It

is important to note that the inventory should be revisited on a regular cycle. The status of applications (e.g., business criticality) can change as a function of time, and as a result, priorities for reengineering will shift.

Document Restructuring: Weak documentation is the trademark of many legacy systems. But what do we do about it? What are our options?

1) *Creating documentation is far too time consuming. If the system works, we'll live with what we have.* In some cases, this is the correct approach. It is not possible to re-create documentation for hundreds of computer programs. If a program is relatively static, is coming to the end of its useful life, and is unlikely to undergo significant change, let it be!
2) *Documentation must be updated, but we have limited resources. We'll use a "document when touched" approach.* It may not be necessary to fully re-document an application. Rather, those portions of the system that are currently undergoing change are fully documented. Over time, a collection of useful and relevant documentation will evolve.
3) *The system is business critical and must be fully re-documented.* Even in this case, an intelligent approach is to prepare documentation to an essential minimum. Each of these options is viable. A software organization must choose the one that is most appropriate for each case.

Reverse engineering: The term *reverse engineering* has its origins in the hardware world. A company disassembles a competitive hardware product in an effort to understand its competitor's design and manufacturing "secrets." These secrets could be easily understood if the competitor's design and manufacturing specifications were obtained. But these documents are proprietary and unavailable to the company doing the reverse engineering. In essence, successful reverse engineering derives one or more design and manufacturing specifications for a product by examining actual specimens of the product. Reverse engineering for software is quite similar. In most cases, however, the program to be reverse engineered is not a competitor's. Rather, it is the company's own work (often done many years earlier). The "secrets" to be understood are obscure because no specification was ever developed. Therefore, reverse engineering for software is the process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction than

source code. Reverse engineering is a process of design recovery. Reverse engineering tools extract data, architectural, and procedural design information from an existing program.

Code restructuring: The most common type of reengineering (actually, the use of the term *reengineering* is questionable in this case) is code restructuring. Some legacy systems have relatively solid program architecture, but individual modules were coded in a way that makes them difficult to understand, test, and maintain. In such cases, the code within the suspect modules can be restructured. To accomplish this activity, the source code is analyzed using a restructuring tool. Violations of structured programming constructs are noted and code is then restructured (this can be done automatically). The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced. Internal code documentation is updated.

Data restructuring: A program with weak data architecture will be difficult to adapt and enhance. In fact, for many applications, data architecture has more to do with the long-term viability of a program that the source code itself. Unlike code restructuring, which occurs at a relatively low level of abstraction, data structuring is a full-scale reengineering activity. In most cases, data restructuring begins with a reverse engineering activity. Current data architecture is dissected and necessary data models are defined. Data objects and attributes are identified, and existing data structures are reviewed for quality. When data structure is weak (e.g., flat files are currently implemented, when a relational approach would greatly simplify processing), the data are reengineered. Because data architecture has a strong influence on program architecture and the algorithms that populate it, changes to the data will invariably result in either architectural or code-level changes.

Forward engineering: In an ideal world, applications would be rebuilt using an automated "reengineering engine." The old program would be fed into the engine, analyzed, restructured, and then regenerated in a form that exhibited the best aspects of software quality. In the short term, it is unlikely that such an "engine" will appear, but CASE vendors have introduced tools that provide a limited subset of these capability-ties that addresses specific application domains (e.g., applications that are implemented using a specific database system). More important, these reengineering tools are becoming increasingly more sophisticated.

SELF-ASSESSMENT QUESTIONS - 3

7. Reengineering of information systems is an activity that will absorb information technology resources for a short time. (True / False)
8. There are _____ phases in the life-cycle of software reengineering process model.
9. _____ for software is the process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction than source code. (Pick right option)
   a) Software Engineering
   b) Business Process Reengineering
   c) Reverse engineering
   d) Business Process Outsourcing

Key steps in forward Engineering:

Here are the key steps in the forward engineering process:

- Requirements Gathering: Gathering and recording the software's needs, both functional and otherwise. Features, GUIs, performance metrics, and other requirements are all part of the scope.
- System Design: Making design documents that describe the software system's architecture, components, modules, and interactions at both a high and low level. Databases, user interfaces, and other crucial components are designed during this stage.
- Implementation: Coding the application's core functionality according to the design documents. Here, the software is built using programming languages and development environments.

- Testing: Putting the software through its paces by means of a battery of tests (unit testing, integration testing, system testing, etc.) to guarantee that it performs as expected and satisfies the requirements laid out in the specification document.
- Debugging and Fixing: Finding and fixing bugs, mistakes, and other problems that appear during testing to boost the software's quality.
- Integration: The process of integrating individual software elements into a unified whole.
- Documentation: Facilitating the software's adoption and use by producing detailed documentation such as user guides, technical specifications, and other useful items.
- Deployment: Getting the programme up and running on the system where it will be used by end users requires installation and configuration.
- Acceptance and User Training: Educating those who will be using or managing the programme to improve their proficiency with it. In this stage, you should also seek out consumer feedback and endorsement.
- Repair and Upkeep: Maintaining and improving the programme through regular upgrades and fixes to fix bugs and improve usability.
- When creating software, forward engineering is the standard method used because it has been around for a long time and has proven effective. To understand an existing software system without having access to its source code, one must resort to reverse engineering.

Economics of Re-engineering

Reengineering drains resources that can be used for other business purposes. Therefore, before an organization attempts to reengineer an existing application, it should perform a cost/benefit analysis. Business Process Reengineering (BPR) economics entails calculating the possible ROI for reworking and upgrading existing business processes by weighing the costs and benefits of doing so. Efficiency, productivity, and cost-effectiveness are the primary areas of emphasis.

A thorough analysis of the costs, benefits, and risks associated with reworking current procedures is at the heart of the economics of reengineering. To guarantee that the

reengineering project will improve the company's bottom line and overall performance, a comprehensive investigation is required.

- Cost of maintenance is calculate by finding out the cost of annual an maintenance over application lifetime
- Cost of reengineering is calculated by predicting the investment reduced by the cost of implementing the changes and engineering risks factors
- Cost benefit is calculated by subtracting the cost re-engineering and cost of maintenance.

6. TECHNICAL PROBLEMS OF REENGINEERING

Most reengineering tools work only on workstation and PC platforms. This may be because of the client server technology and advancements in the field of workstation because of their size and popularity. What this means to reengineering is that the legacy applications which run on Mainframe systems are now to be made compatible to run on the downsized workstations. This will be a big challenge on the interfacing side to bridge the gap between the input to the reengineering tool from the mainframe system and returning the code back to production legacy system from the tool.

Another issue the reengineering tools contend with is cases where the legacy system is developed using multiple languages or code is embedded with macros. As we mentioned earlier the dominant language for legacy applications is COBOL, and there are many reengineering tools available to work with COBOL code. But additionally if the legacy application contains a DBMS call, or a section or function developed using FORTRON, reengineering should be smart enough to extract the information accordingly understanding the functionality of this code section.

Usually a platform that allows an integration of tools is the best choice to handle this variety of input. This means that the tools can exchange software meta-data. Remember, the reengineering tools will need to interact with metrics tools, validation tools, a repository, documentation tools, etc. The DoD has created the I-CASE (Integrated Computer-Aided Software Engineering) project to allow suites of tools to exchange software meta-data. I-CASE was defined to allow an open architecture that encompasses development tools, maintenance tools, and reengineering tools.

In some cases the reengineering tools may not be available to transform the organization's legacy application. This may be because the user base of the language of legacy application might be so small that no proper support for the language is available to develop any tool around it. The same scenario is applied to software developed using proprietary technology. In such cases reengineering might not be the best option, and the organization will have to make one of the following three choices.

- Manually reengineer the code without using any tools or try to redevelop the code
- Develop proprietary tools to transform the application
- Consult the reengineering expert to help with the transformation

And finally, beware the problem of scalability. A small pilot project may successfully reengineer an organization's software but when applied to larger software applications (on the order of a million lines of code or more), some reengineering tools (or repository) start having problems. These problems include significantly slower tool response times (on the order of hours or days), insufficient memory, downloading congestion, and difficulty in re-integrating different software modules. Even the graphics will have problems displaying excessive design information. Our advice is to have the vendor(s) demonstrate their tool(s) using software of a magnitude similar to the software you need to reengineer.

SELF-ASSESSMENT QUESTIONS - 4

10. Reengineering tools are platform independent and can work in any platform. (True / False)
11. I-CASE stands for _____.

7. SUMMARY

Let's recapitulate important concepts discussed in this unit:

- Software organizations are continuously challenged by developments like requirements to eliminate duplication or redundancy in the system,
- Requirement to adhere to the standards, shortfall of skilled and trained resources, changing quality assurance trends, need of business process changes etc.
- While reengineering is not the solution to the above, it is a major enabling technology.
- A maintenance organization's short-term goal is to clear the growing backlog of maintenance demands.
- The long-term goal is to support change at the requirements level.
- As organizations begin to think in global terms, their primary challenge will be to use this technology under diverse social and political cultures.

Reasons for the failure of Reengineering Efforts:

In order to improve your chances of success, you need be familiar with the common causes of reengineering failure. Attempts at reengineering often fail for the following reasons:

By adopting a high-level structure for decision making early in the reengineering process, problems such as these can be more easily avoided.

- The organization inadvertently adopts a flawed or incomplete reengineering strategy
- Without well-defined goals, it's hard to tell if a reengineering project was successful. Uncertainty and failure are the results of setting objectives that are either too broad or too lofty.
- A flawed transition strategy. A flawed environment integration strategy and A flawed strategic process
- Lack of clear objectives and goals: Without clear and specific objectives, it's difficult to measure the success of a reengineering effort. Vague or overly ambitious goals can lead to confusion and eventual failure.

- Inadequate Planning and Preparation: Failure might result from diving headfirst into a reengineering project without first identifying risks, allocating resources, and setting a reasonable deadline.
- Poor Participation and Communication from Stakeholders: Resistance and lack of support can arise if key stakeholders aren't consulted or updated on the reengineering effort's goals, progress, and benefits. Absence of Management Buy-In and Sponsorship Reengineering initiatives may struggle to achieve the requisite resources, authority, and priority inside the organization if they do not receive significant support from top-level executives.
- Opposition to Progress: Existing procedures, systems, or workflows may be met with resistance from employees and other stakeholders. As a result of this opposition, the project may stagnate or even collapse
- Scope Expansion and Unrealistic Expectations: Extending the scope of the reengineering project beyond the initial plan or attempting to tackle too many problems at once might cause delays and deplete resources.
- Inadequately Specified Procedures and Prerequisites: Misinterpretations, delays, and mistakes in implementation are all possible outcomes of requirements and processes that are not crystal clear.
- Problems with Validation and Tests: Undiscovered bugs can cause system failures or performance concerns after implementation if testing is insufficient or inadequate.
- Inadequate Knowledge and Unmet Needs: Suboptimal solutions may result from a lack of knowledge or experience with the technology, processes, or industries involved in the reengineering endeavor.
- Organizational Culture Ignored: Alignment and integration issues with the reengineered processes may result from a failure to take into account and address the preexisting organizational culture, values, and conventions.

8. TERMINAL QUESTION

1. What are the common problems associated with software maintenance?
2. When to choose redevelopment and reengineering?
3. What do you mean by Business Process Reengineering? Explain.
4. What are the technical problems associated with reengineering?

9. ANSWERS

Self Assessment Questions

1. True
2. Reengineering
3. a) Maintenance
4. True
5. COBOL
6. b) Business Process Reengineering
7. False
8. Six
9. c) Reverse Engineering
10. False
11. Integrated Computer Aided Software Engineering

Terminal Questions

1. Software organizations are continuously challenged by developments like requirements to eliminate duplication or redundancy in the system, requirement to adhere to the standards, shortfall of skilled and trained resources, changing quality assurance trends, need of business process changes etc. (Refer Section 2 for detail)
2. Based on the situation you can completely redevelop the system or you can go for reengineering. (Refer Section 3)
3. Business Process Reengineering is 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. (Refer Section 4)
4. Most reengineering tools work only on workstation and PC platforms. This may be because of the client server technology and advancements in the field of workstation because of their size and popularity. (Refer Section 6)