Unit 13 Software Re-Engineering

Structure:

13.1 Introduction

In the previous unit, we have discussed various software testing techniques. Traditionally, software engineering techniques have attempted to improve the process of new software development. These efforts have produced structured analysis/design, object-oriented analysis/design, Computer-Aided Software Engineering (CASE), etc. But what does an organization do with its legacy (i.e., existing) software created prior to the adoption of these wonderful new methodologies? Legacy software still needs to be maintained even though its quality, performance, reliability, and maintainability are deteriorating.

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

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

13.2 Software Maintenance Problems

Reengineering tools can capture design information from otherwise indecipherable software (i.e. spaghetti code). These tools can supplement your legacy documentation to comply with DoD-STD-2167A, MIL-STD-498, or whatever documentation standard your organization uses. Unstructured software can be structured. And software/data can be ported to new languages, configurations, or platforms.

Software reengineering cannot exist in a vacuum. Reengineering must bridge to a defined target maintenance environment. It's important to note that any reengineering project must fit within the framework of an organization's strategic/tactical plan.

Other reasons to reengineer include:

- Allow legacy software to quickly adapt to changing requirements
- Comply to new organizational standards (e.g., migrate legacy software to Ada)
- Upgrade to newer technologies/platforms/paradigms (e.g., object-oriented)
- Extend the software's life expectancy
- Identify candidates for reuse
- Improve software maintainability
  - Increase productivity per maintenance programmer
  - Reduce reliance on programmers who have specialized in a given software system
  - Reduce maintenance errors and costs

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. Let's look at some statistics:

Software errors can be very expensive: In a recent study, the top 10 most expensive software errors were maintenance errors. In fact, the top 3 most expensive software errors involved a single line of source code and cost their respective organizations $1.6 billion, $900 million, and $245 million.

Software maintenance is very costly: A few statistics easily bears this out:

- Maintenance costs (including personnel and hardware/software usage fees) run as high as 50-80% of the software life cycle resources or $30 billion a year in the United States alone. Approximately 50% of maintenance costs are spent on just understanding what the software does.
- A 1991 Gartner Group study estimates that 90% of all software resources will be required by maintenance activities by 1995.
- The Department of Defense (DoD) spends approximately $24 billion on software each year. Maintenance accounts for about 70% of this budget. To cite one recent example, the software development costs for the F-16 jet fighter were $85 million. The projected software maintenance costs are $250 million.
- It was estimated that it will cost $75 billion to fix the year 2000 problem in the USA alone.

Maintenance personnel are getting scarce: Software maintenance typically faces the following problems:

- frequent failure rates
- Complex design (e.g. unstructured code, tightly coupled to hardware or other software, low cohesion, unknown development process, etc.)
- Unpredictable "ripple" effects
- Unreliable or missing documentation
- Obsolete hardware platforms
- Loss of experienced maintenance programmers or original developers
- Growing backlogs

All of the preceding problems are magnified when considered within an environment of shrinking budgets. But it is more economically feasible to address your organization's maintenance problems than it is to improve software development. Assume software maintenance accounts for 80% of a software system's lifecycle costs while new development accounts for 20%. If you double the efficiency of the development process (using CASE tools, etc.) then you've freed up 10% of your organization's software resources (half of the 20% formerly used for new development). But if you double the maintenance productivity, you free up 40% of your organization's software resources (half of the 80% formerly used for maintenance). Considering the DoD spends approximately $16.8 Billion a year on maintenance (roughly 70% of its entire software budget of $24 Billion) then we're talking about a $6.7 Billion savings in the DoD alone (40% of $16.8 Billion).

Self Assessment Questions

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. _______________ allows legacy software to quickly adapt to changing requirements.
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

13.3 Redevelopment vs. Reengineering

If the preceding problems become severe enough (and the legacy software is important enough) then redevelopment is the traditional solution. But this is not always a good idea for the following reasons:

- **Critical corporate knowledge is contained within the legacy software**
  Legacy software represents an enterprise model of your organization. If the software has been in use for a long time, then it has most likely survived by providing a critical service. In fact, some software engineers believe that most organizations have already built the majority of their useful software systems. The original users and developers may no longer be available to explain all the reasons behind the creation of, and subsequent modifications to, the software.
- **Legacy software is a valuable asset**
  According to some estimates, software development runs about $8-20 per LOC (Lines of Code). Since the 1950's, over 100 billion LOC have been written – 80% of which is COBOL. This represents an investment

in COBOL code alone of between $640 billion and $1.6 trillion. This asset is second only to oil reserves or the global marketplace.

- **Reusable, reengineered software costs much less than redeveloped code**
  Estimates of reusing reengineered software is about $2-$5 per LOC. As seen previously, newly developed code costs $8-20 per LOC.

Of course, redevelopment cannot be ruled out. Valid reasons exist for completely replacing legacy software. But for the preceding reasons, software reengineering should be considered a viable alternative.

13.4 Business Process Reengineering (BPR)

A new business strategy now challenges software organizations. It is called Business Process Reengineering (BPR). The reader should never confuse business process reengineering with software reengineering. But we have included this sub-section because software reengineering is often driven by the requirements 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

Traditionally, organizations are structured vertically. Business units may include manufacturing, marketing, administration, etc. But vertical structuring is often a barrier to quick adaptation to the demands of a rapidly changing world. When confronted with a new objective, client, strategy, etc., every business unit must be mobilized from the top down. Thus, administration gears up, marketing gears up, manufacturing gears up. Business process reengineering says this is wasteful of time and resources. The organization should be cross-functional so that integrated work teams focus on critical processes. This re-organization reaches into management so that instead of a vice-president in charge of marketing or R&D, there may be a vice-president in charge of the manufacturing and delivery of product XYZ or service ABC.

What is the role of software reengineering?

Legacy software was developed to support business functions within the traditionally vertical organization structure. Thus, organizations have software to support marketing, manufacturing, etc.

Software reengineering can capture the software design information. Using new tools and techniques, this design information can be broken up into functionally cohesive chunks. These chunks are then analyzed and regrouped around the newly identified key business process. This regrouping is termed re-aggregation.

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 communications 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

4. Legacy software represents an enterprise model of any organization. (True / False)
5. 80% of the codes written since 1950 were in _______________.
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

13.5 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 10.1. 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.

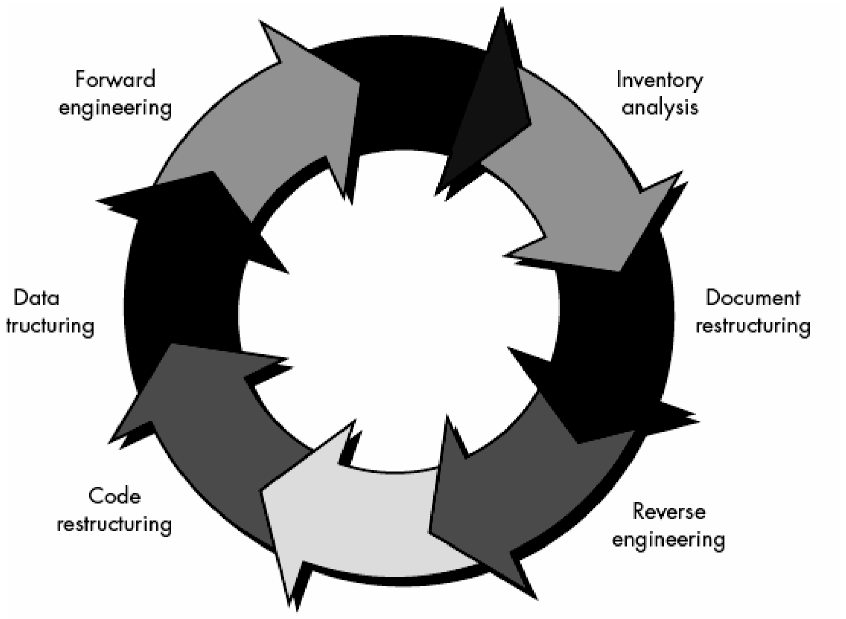

Fig. 10.1: Six Activities of Software Reengineering Process

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) T*he 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

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

13.6 Technical Problems of Reengineering

Most reengineering tools work only on workstation and PC platforms. This may be due to the trend of client-server, distributed systems processing or it may be due to the versatility and power of mid-sized workstations. Whatever the cause, mainframe-based organizations interested in reengineering face downsizing/downloading of their software to run on these platforms. Thus, an interface issue is introduced both for input to the tool and returning its output to the production platform.

Reengineering tools still have a serious problem handling software written in multiple languages or with embedded macros, DBMS calls, etc. A software system is more than just the dominant source code language. Let's look at a typical MIS application. The dominant language is usually COBOL and there are plenty of COBOL reengineering tools available. But what about all of your DBMS calls? Will the tool handle these or must they be stubbed out? Do you have any embedded calls to FORTRAN or Assembler routines? Is the system online and if so, will the tool handle your I/O screen calls? What about the design logic contained in the JCL? This, too, must be captured and represented. If you're a DoD organization, will you have to generate Ada code to conform to DoD standards?

Usually a platform that allows an integration of tools is the best choice to handle this variety of input. This means 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.

The technology and tools that fit your needs may simply not exist. If your legacy software is not written in a source code language with a sufficient user base, no vendor will deem it economically feasible to create a tool for that language. The same can be said regarding your proprietary DBMS or data file system. So you are left with one of three choices:

- Manually reengineer or redevelop your systems
- Develop your own proprietary tools
- Contract with a software reengineering service to reengineer your legacy software for you

Here is a cautionary note regarding this last option. Never turn your software systems over to a service company and assume the reengineering will be done to your complete satisfaction unless you stay intimately involved. Stay involves every step of the way and keeps the application's users involved as well.

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

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

13.7 Summary

Let's recapitulate important concepts discussed in this unit:

- Software organizations are clearly challenged by several major trends such as: Elimination of duplication/redundancy of systems, Conform to standards, Business reengineering, Quality assurance, Declining budgets, Declining trained personnel, Open systems, Concurrent processing 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.

13.8 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?

13.9 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. Conform to standards, Business reengineering, Quality assurance, declining budgets, declining trained personnel, Open systems, Concurrent processing etc. are some the problems associated with software maintenance. (Refer Section 13.2 for detail)
2. Based on the situation you can completely redevelop the system or you can go for reengineering. (Refer Section 13.3)

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 13.4)
4. Most reengineering tools work only on workstation and PC platforms. This may be due to the trend of client-server, distributed systems processing or it may be due to the versatility and power of mid-sized workstations. (Refer Section 13.6)