CHAPTER 2

Legacy Systems and Reengineering Issues

2.1 Introduction

A large portion of the software used in the industry today is legacy software. Legacy software systems are systems that were designed and developed many years ago. Too often a legacy system evolves into an unmaintainable system whose structure has a little to do with its original conception or the structure described in the system’s documentation. Despite these problems, organizations can have compelling reasons for keeping such a systems such as:

- Rich functionality that implements main business logic e.g. Payroll, Inventory, Financial Accounting etc.
- System do still meet basic needs of the organization e.g. system to handle customer’s accounts in banks.
- The costs of redesigning the system are prohibitive because it is large, monolithic, and/or complex.
- The system requires close to 100% availability, so it cannot be taken out of service, and the cost of designing a new system with similar availability level is high.
- The system works satisfactorily, and the owner sees no reason for changing it.

Tremendous changes are taking place in the business world today due to the frequent occurrence of new computer technologies. Modern software systems must conform to requirements, such as flexibility, adaptability, time to market, and continued business process reengineering. Driven by these requirements, the migration and integration of legacy systems towards new platforms and operating environments provides an effective strategy for organizations to maintain their competitive edge. Organizations,
which integrate new development with the existing legacy systems, have a higher success rate. Hence it commonly becomes necessary to reengineer such systems to move them to a new platform and integrate with other systems.

2.2 Legacy System Definitions

Legacy system/application/software is viewed differently from different perspectives of various professionals. Hence there are multiple definitions for legacy systems. Few of them are reproduced below, which bear direct meaning for the thesis.

- Hardware and software applications in which a company has already invested considerable time and money. Legacy systems typically perform critical operations in companies for many years even though they may no longer use state-of-the-art technology. Replacing legacy systems can be disruptive and therefore requires careful planning and appropriate migration support from the manufacturer.
  www.cyrdas.org/ci.glossary.html

- Legacy systems are those that have existed within an organization for many years in basically their original form. They are typically the core systems (or at least the core data) of any organization and are usually larger, mainframe-based systems. Examples include the state's payroll system, AFRS, and the driver's licensing application.
  www.dis.wa.gov/isb/core/system/glossary.htm

- Computer systems or application programs which are outdated and incompatible with other systems, but are too costly to replace or redesign. They are often large, intimidating, and difficult to modify.
  www.rigi.csc.uvic.ca/Pages/description/glossary.html

- Mainframe and minicomputer systems which pre-date LANs and Bridge/Router based WAN internetworking.
  www.syncresearch.com/support/glossary_g_m.htm
• The older, traditional mainframe-based business information systems of an organization.

• Legacy systems are the information resources currently available to the organization. They include existing mainframes, personal computers, serial terminals, networks, databases, operating systems, application programs and all other forms of hardware and software that a company may own. Typically they are, like most legacies, of great value to the organization, and for this reason, it is not viable to replace an entire system, when changing the programs that run on it. Unfortunately, most legacy systems cannot easily or economically be extended, modified or otherwise modified to fulfill new requirements.

• A legacy system is an antiquated computer system or application program which continues to be used because the user (typically an organization) does not want to replace or redesign it.

2.3 Gravity of Legacy System Problem

Is there enough legacy software in the world to justify investments in software renovation technology? It turns out that we are living on a software volcano: large numbers of new and old software systems control our lives. We admire the sheer bulk of the magnificent volcano, benefit from the fertile grounds surrounding it, yet at the same are suffering from frequent eruptions of lava, steam, and poisonous gas, uncertain what is going on within the volcano, and when the next large eruption will be.

The figures collected by Jones [2] provide insight in the size of the problem. He uses the function point (FP) as unit of measurement for software. It abstracts from specific programming languages and specific presentation styles of programs. The correlation between function points with the measurement in lines of code differs per programming language, and is summarized in Table 1.
Table 1: Relationship between Function Points (FP) and Lines of Code (LoC)

Evolving better ways of developing new software will not solve this problem. When an industry approaches 50 years of age—as is the case with computer science—it takes more workers to perform maintenance than to build new products. Based on current data, Table 2 shows extrapolations for the number of programmers working on new projects, enhancements and repairs. In the current decade, six out of ten programmers are working on enhancement and repair projects. The forecasts predict that by 2020 only one third of all programmers will be working on projects involving the construction of new software.

Table 2: Forecasts for numbers of programmers (worldwide) and distribution of their activities

<table>
<thead>
<tr>
<th>Year</th>
<th>New Projects</th>
<th>Enhancements</th>
<th>Repair</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1950</td>
<td>90</td>
<td>3</td>
<td>7</td>
<td>100</td>
</tr>
<tr>
<td>1960</td>
<td>8500</td>
<td>500</td>
<td>1000</td>
<td>10000</td>
</tr>
<tr>
<td>1970</td>
<td>65000</td>
<td>15000</td>
<td>20000</td>
<td>100000</td>
</tr>
<tr>
<td>1980</td>
<td>1200000</td>
<td>600000</td>
<td>200000</td>
<td>2000000</td>
</tr>
<tr>
<td>1990</td>
<td>3000000</td>
<td>3000000</td>
<td>1000000</td>
<td>7000000</td>
</tr>
<tr>
<td>2000</td>
<td>4000000</td>
<td>4500000</td>
<td>1500000</td>
<td>10000000</td>
</tr>
<tr>
<td>2010</td>
<td>5000000</td>
<td>7000000</td>
<td>2000000</td>
<td>14000000</td>
</tr>
<tr>
<td>2020</td>
<td>7000000</td>
<td>11000000</td>
<td>3000000</td>
<td>21000000</td>
</tr>
</tbody>
</table>
The above discussion clearly indicates scope and gravity of legacy system problem. Table 2 depicts forecasts for number of programmers, which will be required to maintain legacy software. Unless and until one is able to find affordable, scalable, and flexible solution to the problem, we are afraid the figures in the table may come into reality.

2.4 Software Renovation

A key aspect of software renovation is modernization: letting a legacy system, developed using the technology of decades ago, benefit from current advancements in programming languages. Chikofsky and Cross [3] have proposed a terminology for the field of re-engineering. The term reverse engineering has its origins in hardware technology and denotes the process of obtaining the specification of a complex hardware system. Today the notion of reverse engineering is also applied to software. While forward engineering goes from a high-level design to a low-level implementation, reverse engineering can be seen as the inverse process. It amounts to analyzing a software system to both identify the system's components and their interactions and to represent the system on a higher level of abstraction.

This higher level of abstraction can be achieved by filtering out irrelevant technical detail, or by combining legacy code elements in novel ways. Alternatively, it can be realized by recognizing instances of a library of higher level plans in the program code [4, 5, 6]. This latter technique is in particular applied to the problem of program comprehension, which aims at explaining pieces of source code to (maintenance) programmers. Techniques from the debugging and program analysis area, such as slicing [7], can be used for this purpose. The problem of explaining the overall architecture of a legacy system, indicating all the components and their interrelationships, is referred to as system understanding.

Adaptation of a system is the goal of system renovation. This can be done in one of two ways. The first is system restructuring, which amounts to transforming a system
from one representation to another at the same level of abstraction. An essential aspect of restructuring is that the semantic behavior of the original system and the new one remain the same; no modification of the functionality is involved. The alternative way is to perform the renovation via a reverse engineering step, which is called re-engineering: first a specification on a higher level of abstraction is constructed, then a transformation is applied on the design level, followed by a forward engineering step based on the improved design. A renovation factory is a collection of software tools that aim at the fully automatic renovation of legacy systems by organizing the renovation process as an assembly line of smaller, consecutive renovation steps.

One can distinguish methodology and technology for system renovation. The former deals with process and management aspects of renovation and typically identifies phases like system inventory, strategy determination, impact analysis, detailed planning, and conversion. The latter provides the necessary techniques to implement the steps prescribed by the methodology. Although methodology and technology form a symbiosis, we have concentrated on the technological aspects of system renovation.

2.5 Software Reengineering

Reengineering is the analysis of existing software system and modifying them to constitute a new form. The goal of reengineering is to understand the functionality and implementation of existing software and re-implement to improve system functionality, performance. System replacement cost is expensive. Hence reengineering is often a better choice. The four general reengineering objectives are:

- Improve Maintainability
- Migration
- Improve reliability
- Functional enhancement
As system grows, maintainability cost also increases. An objective of reengineering is to re-design the system to improve the functionality, performance of the system. The software's we use are getting outdated every now and then. Software must also accommodate the need to migrate to new hardware platform, operating system, or language. As maintenance and change introduces new bugs, software reliability should be enhanced for the redesigned software. Reengineering is not targeted for functionality enhancement, however it can be used as an opportunity for adding functionality for existing system.

2.6 The Reengineering Taxonomy

Around the world researchers are working on reverse engineering of legacy systems performing either individual research or participating in joint projects connected to it. The debate over the definitions within is far from being resolved. During 1992, several reverse engineering conferences brought up the issue of standard reengineering taxonomy. Though, standardization of reengineering taxonomy has not been evolved, it appears that most of the researchers agree on the definitions set by Chikofsky and Cross [3] and the Software Engineering Institute of Carnegie Mellon University [8]. In the present work, we have followed the definitions set by them.

The taxonomy

- **Restructuring**
  The Restructuring is a transformation from one form of a representation to another at the same relative abstraction level. The abstraction levels have been framed depending on the proximity to machine understanding of the representations. The new representation is meant to preserve the semantic and external behavior of the original representation. Restructuring the code leaves the system at the same abstraction level to fit a new paradigm such as structured system analysis or object-oriented analysis.

- **Reverse Engineering**
  The Reverse Engineering is the process of analyzing the subject system with two goals.
1. To identify the system's components and their interrelationships.
2. To create representations of the system in another for at a higher abstraction level.

Here, the higher abstraction level is understood within the context of the software system's life cycle. For example, the classical waterfall development lifecycle [9] calls for requirement specifications followed by design, code, test, implement, and maintain. Thus, if we start with code or database files, the reverse engineering process will extract design information, which is at a higher abstraction level in the life cycle. The requirement specifications are at the next higher abstraction level.

Reverse engineering involves the extraction of design elements but it does not involve generating new systems. Two of the sub processes of reverse engineering are design recovery and re-documentation.

♦ Design Recovery

In the task of design recovery, external knowledge is used to create a representation of the program's application domain and support the maintenance engineer in understanding the examined program more easily.

Design recovery recreates design abstractions from a combination of code, personal experience, and general knowledge about the problem and application domain [10]. The recovered design abstractions must include conventional software engineering representations such as formal specification, module breakdown, data abstraction, and data flow in a program description language. In short, design recovery must reproduce all of the information required for a person to fully understand what a program does, how it does it, why it does it, and so forth.

♦ Reverse Design

Reverse Design is synonym of design recovery.

♦ Re-documentation
While during the design recovery, external knowledge is used to develop various abstractions of the examined system, re-documentation focuses on the re-development of adaptations of existing representations of the system. The resulting semantically equivalent representations show the system from different view intended for human audience. For example, the representations like control flow graphs, data flow analysis tables and structure charts are the re-documentation of the design information.

- **Forward Engineering**
  Forward Engineering is the traditional process of moving higher abstraction level and logical implementation independent designs to the lower abstraction level and physical implementation of the a system. Here, we define the software development process that uses the output of reverse engineering process as forward engineering. This implies that some reverse engineering must have occurred prior to the forward engineering activity. So it is the set of engineering activities that consumes the products and artifacts derived from legacy software and new requirements to produce new target system with changed functionality and different implementation techniques in another platform.

- **Reengineering**
  The Reengineering is the examination of the subject system to reconstitute it in a new form and subsequent implementation of the new form. Reengineering offers an approach to migrate legacy system towards an evolvable system. The process encompasses a combination of sub processes such as reverse engineering, restructuring, re-documentation, and forward engineering. The new form of the subject system could be structured code from “spaghetti” code, design information in graphical form from the input code or translation of a source code from one language to another while preserving the system’s functionality.

Reengineering covers the examination and alteration of a legacy system in order to rebuild it according to modern software reengineering methods and techniques used in a forward engineering process. Support for such a reengineering task can be gained in providing means for both understanding and capturing knowledge.
about the application and its domain and redeveloping the system on the basis of changed requirements.

Reengineering = Reverse Engineering + δ + Forward Engineering

Where δ denote the change in functionality and implementation technique (two dimensions) in general. Thus, reengineering is the process of creating an abstract description of a system, reason about a change at higher abstraction level and then re-implement the system [11].

- Software Maintenance

The Software Maintenance is the modification of a software product after delivery, to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment. The need for changes in software are mainly motivated by users complain about usability features. Software reengineering can be used to modify these software products and must be preceded by a well-elaborated planning [12].

2.7 Issues and Challenges Facing Legacy Systems

Maintaining and upgrading legacy systems is one of the most difficult challenges enterprises face today. Constant technological change often weakens the business value of legacy systems, which have been developed over the years through huge investments. Enterprises struggle with the problem of modernizing these systems while keeping their functionality intact. Despite their obsolescence, legacy systems continue to provide a competitive advantage through supporting unique business processes and containing invaluable knowledge and historical data.

Despite the availability of more cost-effective technology, about 80% of IT systems are running on legacy platforms. International Data Corp. estimates that 200 billion lines of legacy code are still in use today on more than 10,000 large mainframe sites. The difficulty in accessing legacy applications is reflected in a December 2001 study
by the Hurwitz Group that found only 10% of enterprises have fully integrated their most mission-critical business processes.

Driving the need for change is the cost versus the business value of legacy systems, which according to some industry polls represent as much as 85-90% of an IT budget for operation and maintenance. Monolithic legacy architectures are antitheses to modern distributed and layered architectures. Legacy systems execute business policies and decisions that are hardwired by rigid, predefined process flows, making integration with customer relationship management (CRM) software and Internet-based business applications torturous and sometimes impossible. In addition, IT departments find it increasingly difficult to hire developers qualified to work on applications written in languages no longer found in modern technologies. One approach to dealing with spiraling maintenance costs, manpower shortages and frequent breakdowns for legacy code is to “migrate” the code into a new platform and/or programming language [13].

Several options exist for modernizing legacy systems, defined as any monolithic information system that’s too difficult and expensive to modify to meet new and constantly changing business requirements. Techniques range from quick fixes such as screen scraping and legacy wrapping to permanent, but more complex, solutions such as automated migration or replacing the system with a packaged product. Debate on legacy modernization can be traced more than a decade, when reengineering experts argued whether it was best to migrate a large, mission-critical information system piecemeal or all at once.

Rewriting a legacy system from scratch can create a functionally equivalent information system based on modern software techniques and hardware. But the high risk of failure associated with any large software project lessens the chances of success. Researchers from the pioneering 1991 DARWIN project at the University of California, Berkeley, listed several factors working against the so-called “Cold Turkey” approach:
• Management rarely approves a major expenditure if the only result is lower maintenance costs, instead of additional business functionality.
• Development of such massive systems takes years, so unintended business processes will have to be added to keep pace with the changing business climate, increasing the risk of failure.
• Documentation for the old system is frequently inadequate.
• Like most large projects, the development process will take longer than planned, testing management's patience.
• And finally, there's a tendency for large projects to end up costing much more than anticipated.

DARWIN advocated the incremental approach, popularly referred to as "Chicken Little," because it split a large project into manageable pieces. An organization could focus on reaching specific milestones throughout the long-term project, and management could see progress as each piece was deployed on the target system. Industry experts challenged this model several years later, saying the need for legacy and target systems to inter-operate via data gateways during the migration process added complexity to an already complex process. In addition, gateways were a significant technical challenge.

Many migration projects failed because of the lack of mature automated migration tools to ease the complexity and technical challenges. That started to change in the mid-1990s with the availability of tools from companies such as Anubex, ArtinSoft, FreeSoft, and Relativity Technologies. These tools not only convert legacy code into modern languages, but in doing so, also provide access to an array of commercially available components that provide sophisticated functionality and reduce development costs. They help break up a legacy system's business knowledge into components accessible through modern industry-standard protocols, a component being a collection of objects that perform specific business services and have clearly defined application-programming interfaces (APIs).
Choosing A Modernization Approach

The Internet is often the driving force behind legacy modernization today. The Web can save an organization time and money by delivering to customers and partners business processes and information locked within a legacy system. The approach used in accessing back-office functionality will depend on how much of the system needs to be Internet-enabled. Businesses often face two contradictory constraints: they have to deal with heterogeneous platforms and at the same time meet the quality requirement [14].

Screen scrapers, often called "frontware," is an option when the intent is to deliver Web access on the current legacy platform. The non-intrusive tools add a graphical user interface to character-based mainframe and minicomputer applications. Screen scrapers run in the personal computer, which is used as a terminal to the mainframe or mini via 3270 or 5250 emulation. Popular screen scrapers include Star:Flashpoint, Mozart, and ESL. This technique provides Internet access to legacy applications without making any changes to the underlying platform. Because they're non-intrusive, screen scrapers can be deployed in days and sometimes hours. However, scalability can be an issue because most legacy systems cannot handle nearly as many users as modern Internet-based platforms. In the meantime, reengineering has subsided and is now in the process of being replaced by new technique aimed towards the reuse of the existing software in modern distributed architectures [15]. This is because the ability of a new technology is very important for its economic success [16].

Legacy wrapping is a second non-intrusive approach. The technique builds callable APIs around legacy transactions, providing an integration point with other systems. Wrapping does not provide a way to fundamentally change the hardwired structure of the legacy system, but it is often used as an integration method with Enterprise Application Integration (EAI) frameworks.
EAI moves away from rigid application-to-application connectivity to more loosely connected message- or event-based approaches. The middleware also includes data translation and transformation, rules- and content-based routing, and connectors (often called adapters) to packaged applications. Vendors generally offer one of three system-wide integration architectures: hub-and-spoke, publish and subscribe, or business process automation. XML-based EAI tools are considered the state-of-the-art of loosely coupled modern architectures.

EAI vendors advocate wrapping as a way to tap legacy data while avoiding the misery of trying to modify the underlying platform. This approach also enables integration vendors to focus on the communications and connectivity aspects of their solutions, while avoiding the complexity of legacy systems. Like screen scraping, wrapping techniques are applicable in situations where there's no need to change business functionality in the existing platform.

Another option is replacing an older information system with modern, packaged software and hardware from any one of a variety of ERP vendors, including Lawson Software, Manugistics, PeopleSoft, Oracle, and SAP. This approach makes sense when the code quality of the original system is so poor that it can't be reused. However, deploying a modern ERP system is not a universal remedy. An organization either has to customize the software or conform to its business processes. The first option is necessary if the original system was custom-made and provided a critical business advantage. Over the last couple of years, major ERP vendors have added tools to help adapt their applications to a customer's specific needs. However, customization still carries enormous risks that the system won't be able to duplicate a unique set of business processes. In addition, a packaged system requires retraining of end users whose productivity will slow as they adjust to a new way of doing their jobs. IT staff also will need training on the new system. Finally, ERP applications carry hefty licensing fees that remain throughout the life of the software.
In view of the above discussion, it is clear that most practical, affordable, flexible, and scalable approach for bringing legacy systems into today's technological framework is Enterprise Application Integration. This approach uses strengths of Legacy applications, which are satisfying business local needs, however allows them to go in distributed framework and interact with other legacy/new applications, by adding new required functionality.