Chapter 3
3.1 The Software Maintenance Process

To implement a software maintenance process requires that two key activities be fully mastered regarding Software Engineering:

3.1.1 Software Configurations Management; and
3.1.2 Change management.

These two well-known activities shall not be detailed here. However, it is important to stress another activity which is relatively little used: measurement. Indeed it is not possible to control a process without metrics. Apart from elementary indicators such as days*man-day, or number of Lines of Code modified per person, it is important to measure:

3.1.2.1 The technical quality of each component;
3.1.2.2 The economical value of each component; and
3.1.2.3 The evolution of metrics over time.

This should be weighted with the context of the application, and it should be possible to aggregate these measures vertically per groups of programs and per application, and horizontally according to features such as portability, reliability, ease of maintenance...Once the domains of non-quality are identified, a decision model should be available in order to choose the corrective actions.

3.2 Software Redocumentation

Documentation has always been the weak link in the set of deliverables related to software projects. This is confirmed by a study by IEEE which shows that 95% of software documentation is obsolete and or not complete. The study shows that in 89% of the cases, developers are obliged to use the source code as the main source for documentation. It follows that the greater part of the maintenance time is spent
looking for information, reading it and trying to understand it rather than modifying the code.

A really useful documentation is more than just comment lines explaining a line of code or a group of instructions. It includes all the information required for the maintenance of the application, notably:

3.2.1 How the application is organized (who is the owner, what is the functional or structural decomposition);
3.2.2 Which components are involved (JCL, programs, files and data bases, transactions, etc...);
3.2.3 What are the relations within a component and between components; and
3.2.4 The Data Dictionary.

The wealth of information is not the only criteria of a good documentation. This documentation should also be up to date, available to all, and easy to use, which implies:

3.2.5.1 Processes for automatic update and shared access;
3.2.5.2 Adaptiveness to the user's cognitive model;
3.2.5.3 Easy navigation within a component and between components;
3.2.5.4 A familiar usage metaphor; and
3.2.5.5 A dynamic capacity for impact analysis intra- and inter-components

Since application knowledge resides in the source codes and with the experts, a good redocumentation system should include an automated engine for analyzing the sources, and an infrastructure to capture the experts' knowledge.
Today, it is possible with the technology to find and automatically exploit all the knowledge contained in the source codes. In addition to this knowledge recovery advantage, one should add that the source codes are the most valuable source since they contain the most up to date information. As an example, the automated redocumentation of an application in the insurance industry comprising about 5 million LoC, generates around 500 000 HTML pages of exhaustive documentation. This plainly shows that:

3.2.6.1 Without an automated process, such a large volume of documents cannot be either generated nor kept updated within a reasonable budget and period of time; and
3.2.6.2 Without hypertext navigation, such an amount of information cannot be exploited.

It is not possible to completely automate the capture of the experts' knowledge, called semantic assignment. However it is facilitated on one hand by a man-machine interface consisting of a documentation tool with data-entry fields, and on the other hand by the technology of object-data bases which allows an automatic storage and management of this knowledge. Indeed, a comment is only one of the attributes of an object in data-processing.

3.3 Renovation

The first goal of renovation is to improve the maintainability of code in order to reduce maintenance costs. Beyond this, renovation has an impact on the strategy for the evolution of applications. When an application becomes too complex to maintain, management faces two options:

3.3.1 Re-write the application; and
3.3.2 Replace the application by another or by a software package.

In some cases these options cannot be avoided, but are high-risk, because they are of the big bang type, and are costly. The third way, Renovation, which is now possible thanks to current technologies, leads to a quick reduction of maintenance costs, to an extension of the lifetime of the application, and therefore to the decrease of the global cost. Furthermore, renovation is a low-risk solution because it allows for reverse actions, can be divided in activities of reasonable size, and can be automated in great part.

Renovation covers Redocumentation, Restructuration, Restoration and Reengineering techniques. Redocumentation was discussed hereabove. Restructuration includes a number of techniques which are very automated such as:

3.3.3.1 The removal of dead codes and of unused variables;
3.3.3.2 The reduction of the number of GOTO;
3.3.3.3 The removal of backward GOTO;
3.3.3.4 The restructuration of data;
3.3.3.5 The deletion of unnecessary files;

Figure 3.1. Economics of Software Reengineering (Harry M. Sneed)
3.3.3.6 The reduction of Dominance Trees (for example, reduction of nested IF);

3.3.3.7 The alignment of the precision of numerical data; and

3.3.3.8 The standardization of code and of names.

In a later stage, it is possible to tackle operations such as:

3.3.4.1 Restoring the meaning of functions; and

3.3.4.2 Re-engineering

These operations require more human interventions, but are made easier with the tools described here above, and by other automated techniques such as slicing which allows the grouping of the Lines of Code according to the type of processes (display, access to data, etc...) and to assign them to functions (concept of Function Mining).

A Renovation Project should start with an initial diagnosis. It rests on measuring the quality of the source code and of the economical value of the components. It allows choosing: the sub-set of components with the highest economical value in the application, the most adequate renovation techniques for this subset. The technical and economical metrics of components and of applications which have been previously mentioned play a key role in the Renovation process. Not only do metrics allow targeting the renovation actions, but it allows monitoring how quality evolves over a period of time, and therefore to correct any deviation through specific actions.

3.4 Technologies and Architectures

Redocumentation and Renovation rest on various technologies:

3.4.1 Analysis of Source Code;
3.4.2 Manipulation of Source Code; and
3.4.3 Information storage and diffusion.

For analysis and transformations, the tools which operate directly on the Source Code have generally limited capabilities. The most advanced tools use techniques which proceed from work done on compilers. Indeed, a compiler is the only automate which can interpret correctly and exhaustively the grammar and the syntax of a source code. To that end, it builds up an abstract representation of the source code in the form of a tree structure adapted to the construction of control flows and of data flows, and to the propagation of modifications.

Storing in form of objects is in conformity to the nature of programmatic objects which are very interdependent. It allows the enrichment of information and to interrogate it with a formal language. Regarding diffusion and visualization of information, the Internet technology - based on centralized servers for the creation of HTML and XML pages - and workstation-servers provided with navigators is very adapted to the documentation of applications. It offers major advantages:

3.4.4.1 The concept of a central server allows to control the integrity of documentation, its synchronization with that of software configurations, and the accesses;
3.4.4.2 The navigators alleviate the installation and the maintenance of the proprietary code on each workstation;
3.4.4.3 Intuitive navigation thanks to hypertext and hypergraph interactions, while XML makes possible to exploit the full content of the collected information; and
3.4.4.4 The Web metaphor reduces the training time to a few hours at the most.
An architecture based on client/server models allows the creation of documentation servers which have no impact on the mainframes nor on the workstations. Moreover, the servers can be spread over several computers in order to reach the desired scalability. It is possible to ask oneself why the solutions which are presented here are not more widely used. There may be several reasons to that:

3.4.5.1 Very few companies master the compiler technology;
3.4.5.2 Furthermore it is also necessary to master the mass production process required for the treatment of application systems of several millions LoC;
3.4.5.3 Few people have the know-how to productize such technologies;
3.4.5.4 Until the relatively recent past, the required processing capacities and memory sizes were an obstacle to solutions which were economically acceptable;
3.4.5.5 The Internet technology which forms the basis for the diffusion of information is also a relatively recent tool for maintenance teams;
3.4.5.6 The traditional suppliers of redocumentation software tools are handicapped by the old concepts: workstations oriented (instead of
servers), proprietary presentation interfaces, low performance of the database implementation when millions of lines of codes are to be.

Surprisingly, it is the software houses which came last into the software maintenance market who have been able to capitalize on all these new technologies and which offer well architectured, evolutive, and economical solutions.

3.5 **Measurable Benefits**

The solutions which have been recommended in this section have been implemented in real cases that have provided valuable information:

3.5.1 Compared to a traditional documentation approach, using an automatic hypertext redocumentation and navigation tool can reduce research time from 5 hours to 1h30;

3.5.2 The improvement of the maintenance and of the documentation processes, and the restructuration of the application codes can be allowed to process 9% more of requests in 18% less of man-days.

From the qualitative point of view, one should stress the need to provide maintenance teams with efficient tools having a high technological content in order to boost their motivations. As for management, using these tools allows them to regain control of the maintenance process and put them in a position to engage into a real quality policy.

3.6 **CONCLUSION**

It seems obvious that the solutions presented here have a promising future as they help addressing many challenges:
3.6.1 The human resources shortage, most software development engineers being engaged in software maintenance tasks;
3.6.2 The continuously growing number of new applications systems, and their increasing complexities;
3.6.3 The variety of computer languages and of components used; and
3.6.4 The pressure on costs and quality.

These solutions provide a significant leverage effect on software quality and on Information Systems budgets. They are the first tools for the industrialization of the software asset management.

3.7 Legacy Systems

3.7.1 Background

A legacy system is a system approaching the end of its useful life, in danger of preventing a group from maintaining its operational objective. Legacy systems may be "massive, long-term business investment[s]", crucial to an organization [2]. They must adapt to the shifting business and technological requirements of an organization to remain operationally ready [5]. For example, if new government legislation mandates all cars must run on electricity by 2015, the useful life of an automobile may be cut short as this date approaches. Because the car satisfies critical transportation needs, for instance servicing the commute for a job 40 miles away, an additional investment may be required to deal with the impending obsolescence of the car. Some options include buying a new car, converting the current car so that it works with electricity, or utilizing public transportation. Regardless, the driver must find a way to address the changing government requirements.
Preventing a legacy system from failing or not performing as desired is a challenging task, being delegated to more decision makers as systems age [2]. Because these systems affect company revenue streams and fulfill critical needs, their obsolescence cannot be ignored without negatively impacting other organizational functions [5]. Researching the various methods utilized for life extension and how they can be applied to systems with particular problems will help system engineers and decision makers decide best how to approach system obsolescence and keep systems functioning for years more.

Legacy systems are also important to study because they are often directly linked to a corporation’s core competency, performing key tasks the organization cannot do without. A pizza parlor may be using an authentic wood-burning oven to bake pizza crust. If the oven should break or require service, the bakery will be unable to make pizza crusts, curtailing profit as their core product is halted. If the oven becomes troublesome when the pizza parlor first opens for business, a solid base of customers whom are used to the ‘authentic’ taste of the pizza crust would be small and the repair or replacement of the oven would go unnoticed. However, as time goes on, it becomes more difficult to halt systems actions for service without affecting the business adversely and risking an interruption of the organizations’ function. The organization becomes more dependent on the legacy system as its importance to business functions grows, increasing the risks associated with interrupting the system for maintenance and service along with the risk of what will happen if legacy system issues are not addressed properly [8]. Back to the pizza parlor example above, if a new oven is bought in after the pizza parlor has built up a solid customer base, it may be unable to successfully duplicate the taste and texture of the authentic oven. Customers may not come back as the pizza does not have the authentic taste they have grown accustomed to.
It is important to study life extension of legacy systems because as a system ages it becomes host to all types of problems. Budgetary constraints and functionary requirements for the system may rise, while the reliability of third party vendors and suppliers decreases [8]. New regulatory changes made by outside organizations may need to be applied to the system [10]. Also, vendors may no longer supply the materials or parts needed to repair or maintain elements of the system [11]. Systems dealing with impending obsolescence must find a way to remain responsive and continue to provide valuable products and services to their users [12].

Many companies try to postpone the decline of their systems into legacy territory by concentrating on maintenance and support of the system. Maintenance is part of every system’s life cycle and many organizations assume that regular, scheduled maintenance is enough to keep a system going [12]. However, the techniques used to maintain a system have many concerns that drive the search for reliable, efficient life extension techniques. Continued maintenance and support may not be robust enough to add functionality or incorporate new methodologies successfully. Constant modification and maintenance is thought to contribute to the degradation of the system [13]. “Constant patching makes the system’s reliability questionable” . For instance, a car with a flat tire may have the tire repaired once so that it is safe to drive on. A car that is being driven with a tire that has been patched for a second time may not be as reliable.

It also becomes increasingly difficult to find employees with the skills to maintain legacy systems as the system or the employees’ age advances [16]. One reason may be the skills required to maintain the system are no longer taught or passed on, reducing the dwindling pool of people who do have the knowledge needed to perform what may or may not be highly specialized procedures. Also, those that do
know what was needed to upkeep a system may have retired or moved around in the organization so that there is no one immediately available who can service the system [18]. The employees who are left and have the skill set desired to work on an "older" system may be reluctant to volunteer for the job. Maintenance is often seen as a less desirable job function because people associate job advancement with staying abreast of new skills [9]. Development tasks, associated with learning the latest and greatest information about a system, tend to be rewarded more than maintenance careers, which few employees aspire to [9]. The difficulty associated with finding someone to maintain a system has been a contributing factor in the search to define what other types of life extension techniques can be used for a particular legacy system.

Another issue influencing the study of legacy systems is the costs associated with the life cycle extension of systems. When determining if a system is becoming burdensome to an organization, "the total cost of a system over its intended life cycle" must be taken into account [12]. The maintenance and support of a legacy system can consume an enormous amount of its life cycle budget, particularly towards the end of the life cycle when the system is under increasing pressure to keep up with new technologies, upgrades, and replacement of components. As you can see from Figure 3.3 below, many systems incur substantial changes in cost during two key phases of their life cycle; the first key phase is when the system is starting up and nearing its implementation and the second key phase is towards the end of the system's life (Dilts, 2004). The high cost during the start-up/implementation phase can be attributed to the gathering of materials, testing of the system, and actual implementation costs associated with getting the product up and running. The high costs incurred as the system nears the end of it's life are often attributed to the increased maintenance required to keep the system functioning and performing its needed tasks (Dilts, 2004).
Although some legacy systems do not have maintenance cost as the determining factor of their legacy status and take into account other issues such as physical degradation and age; recent estimates have concluded most companies spend from 70%-75% of a system’s total lifecycle costs on support measures designed to keep the system operational [21]. Extending the life of a system may help to curb these support measures and reduce the heavy increase in money spent on the system towards the end if its life as components are upgraded and repaired for further use.
As Figure 3.3 and Figure 3.4, above, show, one effective way to reduce life cycle cost is to reduce the maintenance cost tied to the system. Figure 3.4 illustrates the relative costs of maintenance compared with other phases of the life cycle using a bar graph format. Figure 3.3 shows that the cost of maintaining a system is much greater than any of the costs associated with the development, design, implementation, and testing phases of a system's life cycle. Maintenance costs are rising so much that the percentage of Fortune 1000 companies information systems' budgets dedicated to maintenance has gone "from 40% in the early 1970's to 90% in the early 1990's" [22]. The cost of ensuring a system is in no danger of interfering with the organizations' revenue stream and remains operational has begun to consume a substantial amount of total life cycle costs, both financial and time wise [27]. Organizations are very interested in other techniques that could be used to treat legacy systems and decrease the maintenance costs associated with keeping them going. "Management must be convinced that the organization is really going to achieve a significant benefit in reduced cost" to utilize a particular life extension method over another [28]. The model hopes to identify those particular methods that may reduce maintenance costs for an organization.

3.7.2 Legacy System Life Cycle

A system can become a legacy system while in any of the different life cycle phases, including introduction, growth, maturity, decline, phase-out, and obsolescence (Figure 3.5). The introduction stage signifies the beginning of the life cycle and is marked by high production and design costs, low usage, and frequent modifications to the system [25]. The 21 Hummer SUV for example was very expensive and designed for military use only during this stage of its life, undergoing frequent modifications to fulfill mission needs. A system could also be
classified as a legacy system in this stage if it has a long design time and has not been introduced yet, meaning new requirements of functionality may be required of it as it is being introduced [33]. The next stage is the growth stage, which takes place when the product has been accepted into the mainstream market. It is here that economies of scale are improved allowing usage to increase and the cost of keeping up the system to decline [43]. The Hummer is now in this stage, as it is being sold through commercial automobile dealers and the price and functional requirements have been reduced to fit mainstream needs. The maturity stage lasts for the longest amount of time, as shown on the life cycle curve below. It signifies that the product has sustained heavy use and is probably being offered at its lowest cost [43]. Some automobiles make this stage as the ubiquitous Honda Civic has, and some have not, as classic Corvette owners could attest to.

![Life Cycle of a System](image)

Figure 3.5. Life Cycle of a System

The decline stage is marked by increased maintenance and part replacement, a sure sign of impending obsolescence [43]. Many owners of older vehicle find themselves with an automobile in this stage as they try to preserve its functionality. If no method is chosen to address obsolete components and surging maintenance responsibilities, the system will enter the phase-out stage followed by the obsolescence phase. The phase-out stage describes a system that may not be used
as much as it previously had been because it is requiring frequent maintenance and declining support by manufacturers and third party vendors [43]. The obsolescence stage signals the system is no longer receiving manufacturer support and the technology responsible for the system is no longer being implemented in the same form and fashion [43]. A model -T ford, the first mass produced car, would now be in the obsolete stage as cars have made great strides in technology and no longer use the same technology as the model-T. Once the system reaches the obsolescence stage, a final decision must be made to extend its life or ignore it.

The model incorrectly assumes a system can only approach obsolescence during the phase-out and obsolescence stage of the life cycle. Legacy systems can be deemed obsolete before they enter these final stages due to the long design and testing phases associated with their complexity [44]. Because of the advances in technology or changes in materials that can spur a system’s march to obsolescence during any phase of the life cycle, life extension can be performed concurrently with any phase of system development to ward off a speedy demise towards obsolescence [43]. The life extension model developed by the researchers examines actions taken to keep the system performing at the levels needed to be viable to the organization during any phase of the life cycle because the literature review has shown there has been little research as to how to actually extend a product’s life and how to determine when a system is slated for extension.

The previous sections have defined what a legacy system is and explained the characteristics associated with these systems. The life cycle of a system has also been defined along with the various phases a system can pass through during its life cycle. The idea that a system can approach obsolescence and become a legacy system during any phase of the life cycle has also been presented, challenging
conventional thought that system designers must only begin to treat aging systems when they near the end of their life cycle.

3.7.3 Legacy System Assessment:

Determining whether legacy systems are worth keeping requires an overall assessment of the system. There is an assessment method that examines a legacy system from its technical, business and organizational perspectives. Assessment characteristics can be selected and values can be assigned to them by users through assessment of these perspectives. By interpreting the results obtained from assessment we can use a method to tailor to the needs of particular evolution projects and organizations. It is not prescriptive of particular tools and techniques, and can be instantiated to offer a cost/risk trade-off. Quick estimates can be derived from performing the method at a high level. The risk of producing an inaccurate assessment can be reduced by further iterations of the method, performed at more detailed levels.

"A legacy system may evolve in a number of ways, depending on factors such as its technical condition, its business value, and the characteristics of organizations involved in maintaining and operating the system." Reaching a decision about how best to evolve a legacy system cannot be made spontaneously; rather it should be based on an assessment of all relevant system attributes.

System assessment is used to gain an understanding of a legacy system, which is fundamental to any system evolution exercise. System assessment should be an initial activity for evolution projects. The system assessment method is composed of a number of activities, illustrated in Figure 3.6. This method is designed to be iterative and may be performed at a number of levels of detail to offer a cost/risk
trade-off. In the interests of applicability to a range of organizations, this method is not prescriptive of techniques and tools used to perform method activities.

Figure 3.6. Assessment Method Activities

3.7.4 Assessment activities

We can gain a sufficient depth of understanding of the legacy system from technical, business, and organizational perspectives from which a decision can be made for an appropriate evolution strategy.

The following factors that force the legacy system for evolution

3.7.4.1 Business process redefinition and reorganisation, due to rapidly changing legal requirements;
3.7.4.2 Redefinition of corporate strategy, e.g. from a data processing model to a multichannel service oriented model;
3.7.4.3 Lack of existence of a corporate information model;
3.7.4.4 Lack of a well defined architecture; and
3.7.4.5 No proper database system, phasing out of a line of hardware and system software.
3.8 Instantiation

An instantiation activity is necessary to tailor the assessment method to the needs of the organization and project. The assessment activities shown in Figure 3.6 may be performed in parallel, and in some cases, particular activities can be omitted. This flexibility is necessary to provide a user-driven method. It involves defining two parameters:

3.8.1 Assessment Technique: Either expert opinion or quantitative metrics may be used. This decision is based on availability of individuals to play expert roles, and maturity of the organization performing the assessment;

3.8.2 Assessment Level: The level of detail at which to carry out the assessment. A decision can be made by examining the varying degrees of granularity.

In order to perform system evolution of legacy systems we need to choose an appropriate strategy for its development. For implementing this below strategic options can be followed:

3.8.3.1 When it is found that the system does not make any effective contribution to the business the system can be completely decommissioned;

3.8.3.2 If the system is still required, we can use the same system by making minor changes with regular maintenance;

3.8.3.3 Legacy systems involve many changes over year and hence the system quality decreases, in such case we can use reengineering;
3.8.3.4 As technology changes take place and availability of the COTS increases we can replace some or all parts of the system.

3.9 Conclusion
As the software progresses, many new software design methodologies are developed, improving software reusability and maintainability, and decreasing development and maintenance time. But most companies have legacy systems that are out of date and costly to maintain. These systems cannot just be replaced with new systems; they contain corporate information and implied decisions that would be lost. They also are an investment, and were too costly to develop and evolve; just to discard. For these purposes, re-engineering becomes a useful tool to convert old, obsolete systems to more efficient, streamlined systems.