Software Maintenance - A Management Perspective (Issues, Tools, Techniques, and Trends)

Phaneendra Nath Vellanky

DISSERTATION.COM

Boca Raton

Software Maintenance - A Management Perspective
(Issues, Tools, Techniques, and Trends)

Dissertation.com
Boca Raton, Florida
USA • 2007

ISBN: 1-58112- 376-0
13-ISBN: 978-1-58112-376-0

Software Maintenance - A Management Perspective (Issues, Tools, Techniques, and Trends)

Doctoral Dissertation

By

Phaneendra Nath V V

Submitted to

Pacific Western University, Los Angeles, California, USA

Dated April 1996

ABSTRACT

Computer systems play an important role in our society. Software drives those systems. Massive investments of time and resources are made in developing and implementing these systems. Maintenance is inevitable. It is hard and costly. Considerable resources are required to keep the systems active and dependable. We cannot maintain software unless maintainability characters are built into the products and processes. There is an urgent need to reinforce software development practices based on quality and reliability principles. Though maintenance is a mini development lifecycle, it has its own problems. Maintenance issues need corresponding tools and techniques to address them. Software professionals are key players in maintenance. While development is an art and science, maintenance is a craft. We need to develop maintenance personnel to master this craft. Technology impact is very high in systems world today. We can no longer conduct business in the way we did before. That calls for reengineering systems and software. Even reengineered software needs maintenance, soon after its implementation. We have to take business knowledge, procedures, and data into the newly reengineered world. Software maintenance people can play an important role in this migration process. Software technology is moving into global and distributed networking environments. Client/server systems and object-orientation are on their way. Massively parallel processing systems and networking resources are changing database services into corporate data warehouses. Software engineering environments, rapid application development tools are changing the way we used to develop and maintain software. Software maintenance is moving from code maintenance to design maintenance, even onto specification maintenance. Modifications today are made at specification level, regenating the software components, testing and integrating them with the system. Eventually software maintenance has to manage the evolution and evolutionary characteristics of software systems. Software professionals have to maintain not only the software, but the *momentum of change* in systems and software.

In this study, we observe various issues, tools and techniques, and the emerging trends in software technology with particular reference to maintenance. We are not searching for specific solutions. We are identifying issues and finding ways to manage them, live with them, and control their negative impact.

Software Maintenance - A Management Perspective

(Issues, Tools, Techniques, and Trends)

Table of Contents

Section 1

Computer Software and Maintenance

Computers have attained distinguished social acceptance with the successful transformation of many activities and facilities in our society into highly efficient and reliable functions, systems and procedures. Computers and software drive most of the critical functions in our society today, with not only business and welfare functions, but the defense and survival of nations depending on them. Software systems are undoubtedly valuable assets to any organization, and consequently they need care and attention. Information system management everywhere is facing stringent budgets and rising expectations. That makes it difficult to throwaway the systems and applications built over years of efforts and large investments. It is wiser to retain and maintain useful systems, and envelop them into evolving new systems and technologies. Anything worth making is worth maintaining. Computer software is no exception to this axiom.

1. Computer Software and Maintenance

1.1. Computer Software and Society

Computing is becoming ubiquitous and more powerfully embedded in our work environments and in everyday objects, like automobiles and many electric gadgets. Beyond doubt, modern society is increasingly dependent on computers and software that runs these systems. Significant advances in computing and communication technology made possible hitherto unthinkable ways of using them. The volume of software is increasing. Every successful implementation of an application increased user confidence in the computer systems and resulted in further demands. The result is voluminous production of software in the last three decades, in billions of lines of code. With every next release or version that includes additional functions and new features, software gets fatter. Users ask for hardware upgrades about two months after getting new software. Then when they get new hardware, they want software with more features. Fat software thus feeds a vicious cycle of upgrading and new features. In the US, Federal government continues to develop more than 90% of its software for specific custom requirements. Reengineering exercises in corporations are generating new and significantly different software products and services. This in turn brought more computing power within everybody's reach at a reasonable cost. In the computing industry, hardware performance is doubling in speed every other year and reliability achieving very high degree. The software cost and investment, on the other hand, kept on increasing. Software always dominates hardware as a constraint on the successful use of information technology. Computer hardware sale today is driven by the software it runs. While hardware production and performance in the last 25 years improved by several factors, software still is produced in the 1960's laborious 'hand-crafted' manner. Thus years of human effort and intelligence, and corresponding time and cost, went in to make systems and software work. Producing software in quantities all these years did not teach us much with software quality and reliability. Best estimates put software productivity improving at a rate of 4% to 7% annually - a negligible figure compared with hardware performance. Peter Neumann questions whether our ability to develop dependable system is improving at all [CACM, Vol 36 no 2, Feb 1993, pp 146]. There is a continual increase in frequency of horror stories reported and successes somewhat few, as can be seen in newspapers today. New problems with software continue to rise and old problems reoccur. In today's rapidly changing environment, it is no longer possible or sufficient to solve problems in isolation. Gone are the days when one product and doing one specific task at a time was adequate to meet user needs. In today's complex computing environment of distributed databases, networks and computing power, software maintenance should be quick, precise and effective over a wide area of operation. Software maintenance is therefore a continuous challenge for years to come. The progress in providing dependable products and reliable software is still to match user needs, application backlogs, rising expectations of what technology can do.

1.1.1. Change is natural - systems and expectations change constantly

Change is natural in our society and it is no longer an option in the corporate world. Humans hate change, yet the change comes on its own. Corporations across the world are undergoing fundamental changes these days. There are many key factors driving corporate transformation today. The demand for reduced cycle time in any business is increasing. Rapid development of information technology brought information resources within the end user reach. Forced by mergers and globalization of operations, many industries are facing the need for greater scale of operations. The power of people over systems is growing. Relationships between and among individuals, different segments of society, nations, government bodies, and business organizations are in constant flux as new alliances are changing the old 'rules of the game.' The direction and success of corporations greatly depend on the people running them. The essential part of business change management is mass brain surgery - it can be bloodless but never painless. Corporations worldwide must therefore, restructure, revitalize, and reframe themselves. Most information systems are designed to serve the very basic needs of an organization, thus resorting 'getting basics done first' approach. Markets are changing too fast for such

a modest approach. In today's information era, knowledge (information) is the resource of the highest value forming the very nucleus of progress, replacing traditional raw components like capital, labor and materials. Creative and competitive use of information systems and technology is essential in today's business world. "Change is in the wind, and it would be unwise to ignore its implications on software engineering" says Professor Sorel Reisman of California State University in Fullerton. Managers are realizing that a new way of doing business takes much more than good information systems, resulting in a radical transformation of the entire organization. Hammer and Champy [1993] advocate reengineering of corporations and systems starting over from scratch. "Everyone involved in a process looks inward toward their department and upward toward their boss, but no one looks outward toward the customer" argue Hammer and Champy. *Work is best organized around results, not tasks such as sales or production.* That is the basic principle behind every reengineering exercise. The existing software systems cannot transform any organization radically. On the contrary bad or inappropriate systems can impede change. Thus reengineering corporation naturally results in reengineering information systems and software. Information systems being critical arms of the business, undergo continuous transformations along with the changing business environments. Changes in organizational structures, business process reengineering, integration of legacy applications with new technologies, and redevelopment of systems within the new framework are not only important, but are critical and vital to stay in business. Managing corporations today is tougher than before. It is hard to raise prices in the competitive world, technological dependencies and need to accelerated changes is critical for the very survival, and globalization is inescapable. Achieving growth means taking more risks. Business world is discovering that the company with the most information has an edge. Company with most software has the strength, advantage and responsibility of managing the choatic situations. Customers and staff will benefit from more choices, more options, and more services. In the long run technology is only as good as the skills of the people who use it. Unless the systems can effectively adapt to the needs and styles of the organization, the stability and competitive position of the organization can be at stake. Transforming radical changes in organizational design into corresponding information systems and procedures is a challenge.

The scenario of computer systems today is really awe inspiring. Giant mainframes gave way to versatile mini-computers. Personal computers gained proficiency and acceptance beyond imagination. Computers today are scalable with varying capacities of handling audio, video, graphic and text combinations, and transporting these frames across vast networks. Most systems today comprise of a coherent collection of computer hardware, software, communications, and possibly other special-purpose hardware (as in process-control, real-time, and on-line systems). These systems allow us to collect data as it is generated (using POS, ATMs, and other devices) and to present information in the form and format, where it is needed for decision-making. Several heterogeneous computer systems work together in this process - some modern, some inherited from previous acquisitions, with diverse technologies and capabilities. These systems process a large, yet undetermined number of space-based elements continuously communicating with the next work through ground and satellite links. Data became a precious commodity that needs care and efficient handling. Corporations are depending on information systems for the very survival. Today, we cannot imagine an airline system, factory, hospital, police force, bank, or any other organization without an information system. Reliability expectations are very high as computers take on critical applications. Hardware or software malfunctions are is no longer tolerated, as failures can result in damage to life or property. Once the system is operational (an airline system, or on-line banking system), it must continue to be operational throughout its life time. Both users and management expect computer systems to continue normal work without any disruption of service, even when undergoing any software updates, trouble-shooting, and other maintenance operations. National security and domestic tranquillity have become highly dependent on our computerized systems. Air-plane crashes, aborted space launches, ghost trains, runaway missiles, self-paying teller machines, interrupted power, elevator, traffic, services in the operation theater of a hospital - are only a few examples of what can happen when a computer system does not work properly or correctly. In this study, we consider subsystems and networks of systems,

both as systems. Software comprises the core intelligence of those systems, that needs to be built and maintained all through its operational life.

1.1.2. Systems evolution is a continuous process, and maintenance is inevitable

The quality, maintenance and management of information systems, and the software that drives these systems can make a difference. Maintaining a software system, especially a large scale system, is usually a huge and complex task. Software systems are business assets today. Computer applications are increasingly recognized as collections of vital core business knowledge. This, together with information technology's increasing proactive role in business tactics, puts software system in a key position. Considerable amounts of human effort, intelligence, time and money go into the production of software. Replacing an operational software is not easy, we need to reinvest on resources and wait till the replaced software reaches acceptable level and stability. Moreover, it may not be a wise move to categorically discard existing systems, and bring in new ones periodically. It is necessary to prolong lifespan, adaptability, and usability of software, specially in view of the gap between demand and supply of information systems, productivity and quality problems, development time, and maturation period. The ever changing hardware, software implementations, methods and innovative applications of computers make software products natural targets of modification. Changes to software are inevitable for a number of reasons. The requirements of the original application can change as a result of a change in the external world or the identification of shortcomings in the requirements as originally realized. Software can change as a result of change in the hardware configuration (again for a number of reasons). Information systems are not be built in a go, they evolve with time as they become progressively useful and acceptable, as they respond to the changes of human needs, technology and application environments. Computer programming and systems analysis activities are of intellectual type and are subject to individual capabilities as well as group dynamics. It is often impossible to gather all user expectations, conceive in advance all available alternatives and options, define or overcome design limitations, traverse all logical paths and include conditions. There can be a snag in any phase of software development - requirement specification, design, programming and implementation. All these activities are predominantly human with varying degrees of expertise, imagination, creativity and attention to details. Complexities often lead to improper decision-making and difficulties in enumerating and understanding the activities. Building large systems and enterprise-wide systems, and systems demanding global communications and connectivities face many challenges. "Complexity of software activities," says Brooks [1990] "makes overview hard; consequently it is difficult to find and control all loose ends; thus impeding integrity." Not every organization develops software, but maintenance remains their concern too. To buy the latest and greatest software or system may be a smart move and offers an easy way. Every software product, whether in-house developed or vendor supplied, is a likely target of change in its operational life and needs maintenance until a successive product takes its place. Even a new software product developed using proven techniques and methods, needs maintenance. The software that does not generate maintenance requirements is probably either not useful or not used. If there are no problems with software, there is trouble. Many properties of software are unique. Software does not decay or deteriorate with time, but its functions and performance characteristics deviate from user expectations as these expectations and associated technology keep changing over a period. Continued modifications to software product over a long period can render the product unmaintainable. It gets progressively worse with the increase of patches applied, with ill-structured code, weak control over its code, documentation or distribution. A state-of-the-art enterprise information system is the backbone of any corporate organization today. Hundreds, if not thousands, of users count on its reliability, availability, and service. Software support is vital to maintain it's reliability, availability, and service level.

1.2. The dominant role of maintenance in software lifecycle

Software errors abound in the world of computing. Sophisticated computer software ranks high on the list of the most complex systems ever created by humankind. Software cost has increased steadily

during the last two to three decades. This steady increase is a result of increased complexity and diversity of software systems. Statistical studies highlight the dominant role of software maintenance within software management and technology. Barry Boehm [1981] projected that by 1985 the cost of producing a computer system would be 90% software and 10% hardware. He also predicted dominant role of software maintenance - about half the software effort and cost likely to go into to maintenance, rather than development.

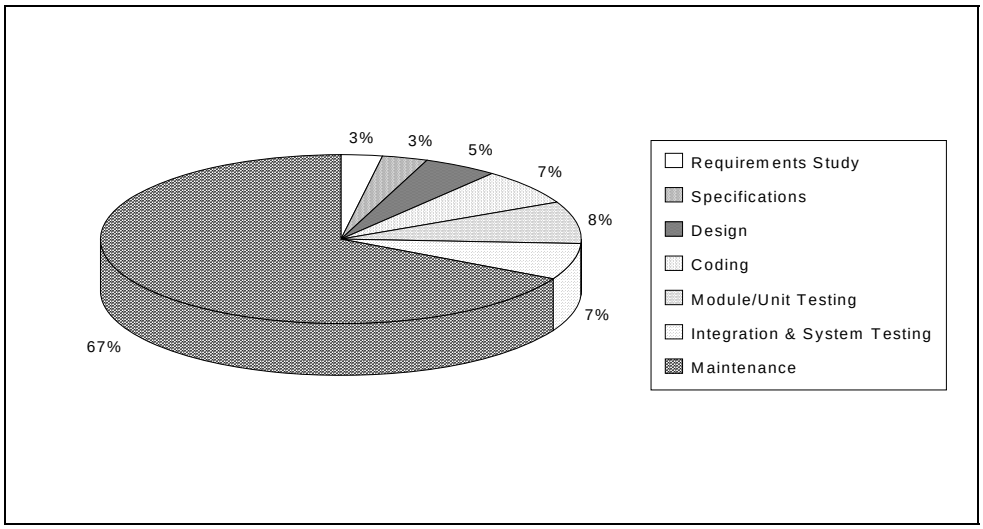

Fig 1: Dominant Role of Software Maintenance [Zelkowitz 1978]

Boehm [1973] estimated one U. S. Air Force system that costed $30 per instruction to develop, but $4000 per instruction to maintain over its life cycle. These facts may seem illogical for inexperienced readers. Today, software maintenance represents 60%-70% of the total cost of software that runs into the tens of billions of dollars, proving Boehm's predictions of 70's and 80's beyond doubt. We can no longer underestimate maintenance. Our ability to support and maintain millions of lines of code developed in multiple languages, intended to run on complex networked systems, is likely to determine the success and future of software systems. Software maintenance is the assurance (and insurance) that software systems, thereby information systems, are kept useful, acceptable, effective and in line with the current requirements and expectations. We have been producing software for decades. Software engineering, in practice, seems to remain loosely related collection of more-or-less good ideas, techniques, methodologies, aided by sometimes helpful tools and program development environments. Most of them work well for small programs, but large systems and network still cause panic. Software still demands significant human intelligence and knowledge at all levels of handling, and maintenance of such human knowledge, intelligence, and expertise is the responsibility of software maintenance. Software maintenance is likely to remain in focus, as there is no alternative to software to drive today's robust computer applications.

Software maintenance is expensive. Estimates put software maintenance costs at over US $30 billion throughout the world, and over $10 billion in the US alone [Martin & McClure 1983, Melymuka 1991]. Most companies are allocating 50% of IS budgets for maintenance; while most programmers (estimated one million programmers) spend 50% to 80% of their time on maintenance [Parikh & Zvegintzov 1983]. In another study, Zelkowitz states that only 33% of a software lifecycle involves new product development, the remaining 67% being maintenance [Zelkowitz 1978]. The estimates in 1992 indicate investment on software is 7 times that of hardware [IDC 1992 survey] which explains why software is in the focus today. For every one dollar a company spends on hardware, it is bound to spend at least

ten dollars on software [Geoff Morris chairman of portability pressure group X/Open in *Computing*, 12 May 1988].

"Software is hard to acquire and support ... ," says Barry Boehm, an authority on software economics, "of every 10 projects I have that are in trouble, seven are in trouble because of software." Boehm was also the director of the Software and Intelligent Systems Technology office of the Defense Advanced Research Projects Agency, in, USA. According to him, the software cost at US Department of Defense would rise from US $ 24 billion in 1992 to US $ 52 billion in the next 15 years. Hardware costs amount for only 15% of project costs, while 70% to 90% of costs amount for software, particularly the maintenance and support of the system. Similar high investment figures dominate software in other key business, industrial and service sectors. Studies have shown that software cost for various systems may exceed 80% of the total system cost [Somerville 1985, Pressman 1992]. The costs of software development are immense, although up-to-date figures are not easily obtainable. Lehman [1980] suggested software costs in 1977 were more than $50 billions in USA alone. This represents more than 3% of American gross national product (GNP) for that year. Boehm [1986] quoted the software cost in 1985 to be about $70 billions in the United States and over $140 billions worldwide. By 1990, this US figure would hit $125 million, predicted Boehm [1986]. Now, it is much higher than those estimates. Datamation [June 15, 1993] estimates software and services revenue worldwide at US $ 300 billion in the year 1992 alone. This can grow significantly with users demanding and vendors injecting new life and change process at all levels in information systems industry. While total software costs have risen rapidly, the ratio of development to maintenance costs has remained relatively constant. This is the result of the growing dependency of many organizations on software, the extended life of software, and the increasing complexity and size of many applications.

There are 15,000,000,000 lines of computer software code currently installed in North America alone and possibly as much more in Europe and rest of the world. A typical software system of 1,000,000 lines of code needs 20 people to support and continually add new functions and integrate subsystems. Each software professional is responsible for an organizational asset with a replacement cost of $500,000 that operates on $600,000 worth of the organization's business. Installed software is a high tech, high stake business, says Nicholas Zvegintzov [1991]. The average Fortune 100 company maintains 35 million lines of code and adds 3 to 5 million lines a year just in enhancements, updates and other maintenance. That does not count new development. This indicates that maintenance alone results in doubling the size of software inventories in every seven year period. The programming teams charged with the responsibility of maintaining this software face on enormous task. Understanding installed software (after the delivery) is another area of concern to software management and the costs involved is considerably high. Estimates indicate those costs at 30-35 percent of total life-cycle cost [Melymuka 1991]. Tom DeMarco [1995] addresses the issue why does software cost so much. Maintaining software on sophisticated and ever expanding computer networks is yet another challenge. In 1991, Andrew Grove, president of Intel Corporation noted, "We would be spending twice as much as the US defense budget on network maintenance." The recognition that software is a capital asset to any corporation or government, is important and constitutes the main driving force for software maintenance research and practice. "Hewlett-Packard currently has between 40 and 50 million lines of code under maintenance; 60 to 80 percent of R&D personnel are involved in maintenance activities; 40 to 60 percent of the cost of production is now maintenance expense," says Don Coleman [IEEE Aug. 1994]. This however, is not a new phenomenon. "Applications development was commonly supposed to be the main problem in data processing, but in the last 25 years, maintenance has taken up some 75 percent of our effort, not development," said Harlan Mills. The technical reason for this high level of maintenance is that it has turned out to be more difficult to develop good systems (good by correctness and capability) than commonly supposed. Information system management everywhere is facing stringent budgets and rising expectations. That makes it difficult to throwaway the systems and applications built over years of efforts and large investments. It is wiser to retain and maintain useful systems, and envelop them into evolving new systems and

technologies. Maintenance is an inescapable part of the process of software development. Radical new methods are required to achieve high quality software. Emerging software methodologies, engineering environments, prototyping tools may be an answer to this [Harlan Mills].

1.3. Computing technology is complex, dynamic and competitive

Information systems today thrive on the dynamics of an environment where nothing is certain but change is imminent. The progress we achieved in computer and communication technology in the last few years is unbelievable. Hardware lifecycle today is three-to-five years in reality. Products and services today changed beyond recognition; they are far different from what we had seen before. The world is moving on to Cyberspace, Information superhighway, National Information Infrastructure, and World Wide Web on Internet. We are already in Information society, with heavy emphasis on information. Organizations are eager to implement technology to attain competitive advantages, some even providing technology directions. The demands on software are high as it plays the intermediate role of meeting user requirements and expectations. The world of computer software is complex, dynamic and even confusing. Computer software drives a variety of hardware systems - super-computers, mainframes, mini's, micros, special purpose machines, personal computers, laptops, client/server machines, network processors, and emerging network computers. High volume data handling devices, redundant storage devices, fail-safe systems are in vogue. Multi-media applications and global communications are not only possible, but are irresistible. Operating systems software is no longer proprietary. New industry standards are evolving with wide user acceptance. Yet, there are differences as every other system offers something more - Unix and its many variants, OSF, MACH, NextStep, Solaris, OpenVMS, OS2, Windows NT, Apple's System 7, and AUX. There are many products waiting to come. These are likely to coexist with already installed products like IBM's MVS, OS/2, DEC's VMS, HP's MPE, Microsoft DOS and Windows, Mac OS. Numbers vary significantly regarding computer platforms and operating systems. There are a few million IBM sites of varying platforms and a few million more of its competitors; there are 4 million Unix sites in USA and many more Unix variants. Estimates indicate world-wide Unix usage at 30 million. We can see new trends setting in with many companies downsizing operations from mainframes to minis, moving on to client/server technologies. Operating systems software is assuming multiple personalities like chameleon. Subordinate operating systems, with such personalities can make and run an application that is not native to the hardware. Microsoft Windows applications can run with Solaris on a Sun workstation. Solaris is mimicking Windows (Wabi - Windows Application Binary Interface) by substituting its GUI functions to appear to the end user in its usual format. Computer networks, network software and protocols are many with acronyms - IBM SNA, LAN Server, APPC, Banyan SPP/IPC Vines, Novell NetWare SPX/iPX, DEC's Pathways DECNET, Unix TCP/IP, internet gateways, DMS, DAL, DCA LAN, SNMP, SMTP, CMIP, Ethernet, Microsoft's LAN Manager, ATE. There is extensive technology packed into every one of these; software engineer should understand each specific function and maintain corresponding prerequisite (protocol) to achieve communication. Sky is the limit for networks. IDC survey of 1992 reveals that there are 130 million PC's working in USA, 37 million of them on LANs. Internet WANs serve 10 million users a day. As on Jan 1995, there are about 50 million users on Internet with 300 million connections, about 7000 WEB sites and 6 million FTP servers. Novell claims that its networking software is working on 535,000 hosts with another network getting on to the network every 30 minutes.

The world is moving towards information superhighways, which could send data, pictures and sounds that enhance communications at speeds never imaged before to serve complex multimedia applications. "We are going to go - long-distance, broadband gigabit worldwide networks, and fiber optics capable of transmitting 100 gigabits per second" says James Martin [CW, June 22, 1992]. Already we depend heavily on on-line services. Business Week's [February 7, 1994] estimates show 1993 figures for on-line personal computing services at 1.3 million subscribers for Prodigy, 1.4 million for CompuServe, and 510,000 for America On-line service. Corporate networks, Defense networks are far and wide, with numerous services and nodes. These figures are not available for public. Workstations

are many and they getting complex - from character-mode (VT100 like) to block-mode (3270 like) terminals to intelligent terminals to PCs emulating terminals. Graphic and non-graphic user interfaces are many running MAC OS, Windows, X-windows Motif, HP NewWave, OpenView, OpenLook, NextStep, etcetera. According to Datapro survey [1993], there are 400 million users using block/character mode terminals; 8 million users using Apple Mac GUI's, and 15 million using Microsoft Windows GUI, in USA alone. While the number of block/character mode terminals is not growing fast, the number of GUI based terminals and PC's are increasing fast. Database management systems are many - IBM's DB2, Oracle, Ingres, Informix, Progress, Powerhouse, Gupta SQL/Windows, SQLbase, Sybase, dBase, Foxbase, Paradox, InterBase etcetera. Every one of these products comes in different flavors to serve different user environments. Over 56 versions of Oracle exist on Unix platforms alone. There are many programming aids and tools - editors, debuggers, test aids, debuggers, software development methodologies and environments. So are the languages - Assemblers, Fortran, Cobol, PL/I, Pascal, C, and other special purpose (Simula, GPSS, Simscript, RPG, SPSS) and 4GL (Ingres 4GL, SQL). Many development methodologies are in practice. We have many software utilities, transaction processing monitors (CICS, TUXEDO, various other TP monitors) and other software in use today. Many general purpose applications are emerging from new technologies - object-oriented, imaging, knowledge-based expert systems, and artificial intelligence. Software is required to deal with a variety of storage media, file systems - VSAM, ISAM, direct, relative, database files; formats - character-sets for languages (Latin, Japanese, French, Hebrew, Arabic), color and graphics (TIFF, BMP, PCX, PICT, JFIF/JPEG, CGM, Oracle Graphics), sound (AIFF-c), objects, images, video (Mac Quickvideo, Microsoft Video for Windows (Windows AVI), Silicon Graphics Indego) and emerging Secure HyperText Transmission Protocol (HTTP). Every specific device and environment needs corresponding software drivers. Also they have to work in a synchronized fashion. Multimedia applications are catching up fast. Most software engineering and operating environments are integrating database, transaction processing and graphic presentation systems forming into further complex entities. Today's real problem, according to an advertisement by Plexes FloWare, can be hypothetically described as something revolving around 2000 workstations, 32 applications, 21 locations, 16 platforms, multiple databases and a feisty board demanding results. Even with emerging open systems standards, the architectural diversity can make open systems support a nightmare.

Anderson Consulting, Chicago projects a worst-case scenario using simple mathematical formula:

4 (Platforms) * 3 (GUIs) * 4 (DBMSs) * 4 (Comm_Providers) * 3 (Productivity_Environments) *

4 (UNIX_Versions) * 3 (OLTP_Systems) = **6,912 Combinations.**

Many applications and systems software products operate interacting with each other in such a complex scenario. Software is global commodity; it is bought and used throughout the world. Many individuals and organizations from all over the world are likely to be involved in software production. Many more users and maintenance personnel would be involved in its operational life. That requires us to address many languages, business practices, cultures and corresponding considerations. Creating or retrofitting software for another country requires preparation and planning with attention to myriad technical details. We need to pay particular attention to issues like language translation, interfaces with many implementations of varied character representations, standards and other conventions. We inherited billions of lines of application and other software code, written all these years to work in specific environment. We need to carry on these applications into new and radically different operating environments.

Mobile offices, home offices, and information access from any where in the world, are becoming way of life. Multi-media applications are emerging fast. Database technology is moving on from relational and object-oriented databases to corporate data warehouses, parallel servers and multi-media formats and services. Technology is today moving closer to work process. Working environments are facing radical changes with the emergence of CDPD (Cellular Digital Packet Data) and wireless access. Traditional computer architecture is moving fast in the new directions of scalable client/server systems,

symmetric multiprocessors, and massive parallel processors. Languages like Java allow us to transport software objects across Internet and Corporate Intranet to any workstation. Changing environments and virtual offices can directly affect the software built around these products. That means maintenance.

1.4. What we know about software maintenance?

Most of the computer systems development till recently, revolved around hardware. Users often bought the available hardware that they could afford, and made software to work on the available hardware to deliver application results. This was the case till late 1980s, only the 1990's started showing a clear shift of emphasis to software. Much of software engineering is still in the early stages. "*No one understands the software industry*," to quote Steve Jobs, Chairman, Next Inc. Maintenance of software is still less understood, if not misunderstood subject. However, software is in operation for many years and some general observations can be made on its operational life and maintenance. Maintenance is an essential element in the life of a software system. Software is on probation when it is released as the testing done initially is under artificially frozen goals with dummy data. Only when the software is operational in user environment, it is put to productive use, and only then it is tested against real life activities. Maintenance begins at this stage. Maintenance is a process with its own rules and techniques. The inescapable challenge for the maintenance professional to face is that the system exists already, with users watching over his shoulder. Usually systems and software developers learn user requirements, and synthesize solutions in their style. The maintainers listen to the user's request (and the agony of malfunctions) from user's view of the system (there lays the catch). They analyze the system, plan and implement changes, and bring changed version of the system on line with a smooth transition. The softness of software is an opportunity for the system synthesist, but is a pitfall for the maintainer. The maintainer's primary skill, like that of surgeons, is not only in making desirable change but also in avoiding undesirable ones. The maintainer has to match the resources of change to the apparent size of the change, not to the size of the system as a whole. The maintainers must somehow find their way through the complexities of the system to the part or component to be modified. Such reductions are simple in retrospect but are a major challenge when they rise.

Every aspect of software maintenance is challenging, particularly when the product or process is not fit to face it. Some challenges are -

1. Grasping and retaining the intellectual control of the system;
2. Understanding and documenting changes to the system;
3. Protecting system integrity during and after maintenance;
4. Protecting knowledge from decay or deterioration due to staff turnover, forgetting;
5. Improper functional or implementation change;
6. Controlling the implementation of continuing stream of change requests;
7. Minimizing the structural decay of systems due to adding functions not originally in the design.

Statistical studies in early 1980's [Parikh 1983] indicated that half of all programming resources are spent for software maintenance. More than half of these resources are used to add new functions and capabilities, and more than half of the programmer's task is understanding the system.

In reality, all software has bugs. Software is released for use, not when it is known to be correct, but when the rate of discovering new errors slows down to one that management considers acceptable. Users learn to expect errors and are often told how to avoid the bugs until the program is improved [Parnas 1985]. All software professionals know that a program has bugs when it is delivered to the

customer. They warn customers about the bugs by printing mind-numbering legal disclaimers on the front page of user manual. *The Wall Street Journal* [Jan 18, 1995] in its headline article talks of software that has frequent glitches. Disclaimers that are publicized say effectively, "we don't really know if this stuff works, and you probably won't be able to figure it out either - but if you actually *do* discover that it doesn't work, don't come whining to us about it. We make no guarantees, warranties, or promises about this stuff." Strange but true, we are still to know for sure why and how software works and more than that why and how it does not work.

Software maintenance includes a chain of activities - understanding and documenting existing systems, extending existing functions, adding new functions, finding and correcting bugs, answering questions for users and operations staff. Software maintenance also involves training new systems staff, rewriting, restructuring, converting and purging software, managing the software of an operational system, and many other activities that go into running a successful software system [Parikh and Zvegintzov 1983]. All changes made during software maintenance, by correcting, inserting, deleting, extending, and enhancing features, would result in the deviation from the originally designed and accepted baseline system. With such deviations, the maintenance professionals should prove and ensure user acceptance at every stage and every cycle of maintenance. Enhancements, by definition involve adding new features, or extending the existing system features. They are also considered part of software maintenance, 'because they are done on operational software.' Some major enhancements have characteristics of development, as the work requires analysis, design, coding, testing and so on; and is like a mini-development lifecycle. However, there is an important difference. Maintenance work is done within the framework of existing software. It is bound by its predefined structures. All new enhancements are to be integrated into that existing structure without disturbing the previously programmed functions. Additionally, the corresponding documentation, user procedures, and training material should be modified to accommodate and reflect the software modifications done. Software maintenance needs careful planning, control and implementation, as it involves working with live systems with many limitations and constraints, while users are waiting and watching. Maintenance professionals need to be responsive and reactive.

Software maintenance activities are grouped into three classes -corrective, adaptive and perfective. These widely used terms were coined by E. B. Swanson [1976] and Parikh calls it 'Swanson's Classification of Software Maintenance'.

- *Corrective maintenance* is correcting programs and systems that abend or do not meet the original requirements or specifications. These changes are needed to correct the actual errors (induced or *residual* bugs) in a system. It accounts for 20% of all the software maintenance efforts and consists of activities normally considered to be error correction required to keep the software operational. The key causes for corrective maintenance are design errors, logic errors, and coding errors. Design errors are generally the result of incorrect, incomplete, or unclear specifications of requirements, or descriptions of the system components, or change requests, or misunderstanding. Logic errors are the result of faulty logic flow, invalid tests and conclusions, incorrect implementation of the design specifications, or unusual combinations of data, which were not tested thoroughly. Coding errors are generally caused by the computer programmer and are the result of either incorrect implementation or the detailed logic design, or the incorrect use of the source code logic.
- *Adaptive maintenance* is adapting to the external changes, such as changes to the operating system, hardware environment, government regulations, and business rules. Changes to software are made to adapt to the changes in the environment in which software must operate. It accounts about 20% of all software maintenance efforts. The environmental changes are normally beyond the control of the software professional. Some of them are rules, laws, and regulations that affect the system, hardware configurations (new terminals,

printers, communications), data formats and file structures, system software (operating systems, compilers, utilities and their updates and bug fixtures).

- *Perfective maintenance*, as its name implies, is aimed at perfecting the software. It includes the improvements - changes initiated by users who ask for new features, and add-on functions. These are essentially modifications because of changing internal requirements, or an attempt to augment or fine tune the software. Other activities designed to make the code easier to understand, such as restructuring or updating documentation are also considered to be perfective. Optimization of code to make it run faster or use storage more efficiently or to retrieve records from a database more efficiently, is also considered as perfective maintenance. Estimates indicate that perfective maintenance comprises approximately 60% of all software maintenance effort.

Along with the above classification, a new area of software maintenance, *Preventive maintenance*, is emerging fast. This includes maintenance activities performed to prevent problems before they occur. The virus detection and prevention software, security maintenance, data integrity maintenance, and many operating systems services (that monitor network break-downs, data access violations, recovery from failures) come under this classification. Preventive maintenance may involve drills to ensure that practices are in place. Some preventive maintenance operations may have to be initiated, while some are gradually built into the systems and procedures to operate automatically. Some preventive maintenance tasks may have to be built as adoptive, or perfective maintenance add-ons, if they are not designed in the original software. Recent studies by Nosek and Palvia [1990], Deklava [1992] and Magne Jorgensen [1995] indicate slight shift in distribution of efforts by maintenance category, with different pictures projected by maintenance staff and software managers. These can be viewed indicative of trends, as much depends on the size and territory coverage of systems, applications and users. Eventually we move on to *Evolutionary maintenence,* that we facilitate system and software evolution. Change by any name is a change. We can expect significant shift of maintenance efforts, from perfective and corrective maintenance, to adaptive maintenance soon. Proven tools and methods are helping us to achieve reliable software systems faster than before, and often with user participation. At the same time, users are willingly on race to catch up with fast emerging technologies. We need to constantly migrate old systems into new technologies, through adaptive maintenance or reengineering them entirely.

Software does not wear out as hard systems do. Software maintenance differs from the maintenance of physical systems. Physical systems involve repairing deteriorated equipment or parts, whereas software does not deteriorate. Maintainers of physical systems are usually less skilled and need fewer resources (by way of facilities and tools) than builders of physical systems, whereas maintenance of software demands more skills, resources and facilities. The deterioration of software, if any, is due to the haphazard "maintenance" work without systematic approach or patch-work. Software professionals used to handle bugs by patching the code or fixing a disk sector (or zapping a string of characters), without taking time to update corresponding documentation. A quick-fix approach may seem to be work initially, but it can create serious problem in future maintenance. What we need is the right environment (resources, techniques, tools, and training) to deal with almost any kind of software and software problems, so that software does not deteriorate as a result of poorly-thought-out maintenance practices. The work of maintenance includes enhancements for adding new features not originally designed in the system. This is not 'maintaining'in the conventional sense (that is, keeping a system working properly), but involves enhancements (additional development work) to fit into the framework of existing software. The systems in this case, may not meet the original specifications, or not tested completely, and hence may require some 'corrective'maintenance. There is also a need for continuously improving software while maintaining it, and when appropriate, to rewrite or redevelop it partially or completely (reengineering).

Rochkind [1975] provides a partial explanation for the reasons behind maintenance. "Computer programs are always changing; There are bugs to fix, enhancements to add, and optimizations to make; There is not only the current version to change, but also earlier version (which is still in use and need's support) and next version (in testing which almost runs)". He also adds, "Besides the problems whose solutions required the changes first, we cannot forget that changes themselves can create additional problems." Software professionals need information to efficiently manage large software systems. The particular interests to maintenance include - obtaining documentation and system information, preparing impact analysis and change specifications, program and systems analysis, program debugging, editing, testing, and integration tools, configuration management tools, system renewal and reengineering. These are all the activities software maintenance professionals are familiar. Yet we find the information is either not sufficiently and reliably available, or we do not know how to use it.

The following are some more facts that software professionals learnt over the period. Though not presented in any logical order, each of these is an important observation.

- System maintenance begins immediately after the system has been installed. Change requests start flowing in almost immediately after release.
- Software maintenance team have no choice or say over the selection of operating system environment, database software or language selection. These are done in the development stage of the product. Maintenance professionals are simply expected to carry on further tasks.
- The maintenance of the program would be most dependent on its clarity and completeness.
- The characteristics of software such as structure and complexities, module interdependencies, coding, and documentation - would have direct impact on software maintenance. If these are done well, software maintenance becomes easy. Otherwise maintenance has to face the consequences.
- The most frequent problem met in software maintenance is inadequacy, unavailability or total lack of vital information. The know-how and details of software are supposed to be retained in documentation. Such documentation may be unavailable, incomplete, or mismatching with the piece of software in question. Documentation therefore needs special attention in all phases of software development.
- In software maintenance corrections, adaptations and system changes are important, but it is refinements that take the largest amount of time.
- The effort needed to fix a bug is three times in the test lab than by the original programmer. Ten times the effort is needed to fix a bug in the field than in the test lab. It is easier to find and fix bugs at one site than at many.
- The post-installation or implementation meetings are important. This meeting should include a review to determine if objectives have been met. These meetings are better conducted approximately 3 months after software is placed into production. This is usually long enough for any problem to begin to appear.
- Prototyping will generally quicken software development and if properly followed the approach can reduce software maintenance.
- Adaptive maintenance will be undertaken whenever the needs of users' change.
- Radical changes in systems and procedures often require a reengineering project. They cannot be handled through maintenance of existing software.

- The elimination of duplicate or unnecessary steps in a program by reducing the number of instructions required is primary reason for instruction modification. Should maintenance attempt such perfective maintenance measures? It is a big question.
- If changes are useful (desirable), but not essential, they should be deferred until after implementation.
- Natural deterioration of a system over time, known as Entropy, is possible. Application software suffers from it. Use of the same program for several years would tend to increase maintenance, especially when subjected to many changes.
- Project implementation is generally complete when the users sign off their acceptance. Software maintenance kicks off almost immediately after the sign off, and thus requires adequate preparation well in advance.
- Maintenance before actual breakdown is preventive maintenance. Configuration, version and distribution control are important steps in preventive maintenance.
- Detection and correction of operational data (input) errors are the responsibility of (Data control) operations staff. Backup and restore operations are the responsibility of Operational Controls. These are NOT part of software maintenance. Software maintenance personnel are involved to some extent with Disaster Recovery operations.
- The major cause of program maintenance is due to users requesting program enhancements.
- An attempt to keep software up to date with changes in the environment is called Enhancements. Software enhancements are important segment of maintenance.
- The relation between maintenance and system development is that maintenance is a miniature of system development, involving practically all phases of systems development. Software maintenance professionals should therefore, have access and prificiency in all software development activities, tools and techniques.
- Application backlog is the most important factor concerning user relations with software maintenance staff.
- The use of prototypes is likely to reduce application maintenance requirements.
- Maintenance of application software is the responsibility of application programmer. Application program maintenance is usually done by in house software staff. A request to enhance an existing system is usually prepared by the user and not by software personnel.
- Most of software maintenance effort goes to enhancement of applications software.
- The use of high level programming languages, database software, can decrease application software maintenance.
- A decision to revise a system would generally be a result of system analysis or review with adequate support from management.

Software maintenance issues revolve around the entire computer systems environment. These include dynamic changes in technologies - computer hardware, communication systems, operating software. Application domains are subject to change with methods and procedures changing. User expectations keep changing. There are human limitations in planning and controlling various logical steps, disparities in software skills and competence. These can be worsened with lack of unified and scientific approach, absence of tools and techniques, geographically wide-spread areas of operation. Customer relations play important role. Improper handling of change requests, problem escalation, or situation

can result in a crisis, further damaging relations. Lack of proper understanding among teams and team members can aggravate situation and keep management in dark. Lack of attention from management in allocating resources, funds can cripple maintenance process. Maintenance needs adequate resources, preparation, and attention. It is a process that requires continuous review and improvement.

Software products no doubt, are attaining increasing acceptability, but are getting complex with interdependencies and interoperabilities. Layered systems architecture is in practice, with scalable client/server systems and high-speed networks. Standards are increasingly favored. Many software products are converging with uniform application of technology solutions. Technology is moving fast, users are keen to implement latest products and services. Every segment of software technology - processing, database, networking - is getting complex day by day. Every software layer demands specialist knowledge, and their integration demands both technical and practical experiences. Software maintenance needs special skills and extensive intelligence and control of the associated layers for penetration, interpretation, and repair. It is difficult to isolate problems in today's complex networked computing systems operating with many software components and products working together. The future of present systems is the concern of software maintenance [Nicholas Zvegintzov]. Software maintenance is deceptively green field. Although frequent surveys have shown software maintenance to be high priority item, its growth is frustratingly slow. This naturally leads to the conclusion that we *cannot yet* deliver what the user wants. A large program must undergo continual change, or become progressively less useful. The problem with continual change is that software becomes increasingly more complex, and correspondingly the cost of change even higher. Eventually, the cost of change becomes greater than the cost of replacement. That is what is triggering off many reengineering projects. Zvegintzov suggested that the process used by therapists on human beings to encourage them to change themselves could also be applicable to computer systems [Fourth European Software Maintenance workshop at University of Durham UK]. The states of change in therapy are - status quo, foreign element, chaos, integration, new status quo. This leads to the conclusion that software is just as unsatisfactorily open-ended as everything else in human life. The trouble with software is its changing shape. After software is installed, it begins a life of its own, it begins to pay its way, and it begins to change its shape - like Ameoba.

1.4.1. Software maintenance: The Journey is Long

The typical life span of a software product is 1 to 3 years in development and 5 to 15 years in use, the period during which it requires software maintenance. The road to software perfection is long as it takes time for software to stabilize and mature. Riddle observes that it takes the magic number eighteen plus or minus three years for (software) technology to mature from its initial idea (concept) to deployment (commissioning) to its wide-spread use (distribution) [Riddle 1984]. Continued advances in programming technologies permit us today to build more sophisticated systems at less cost. The dichotomy of this situation is that the more useful a system, the longer is its life, and the more it will cost to maintain it's lifetime. Thus the cost and total software lifetime are rising, though the cost per year may decrease for some software products. Trillions of dollars have been invested in legacy systems. Most systems professionals see emerging technologies as a way to leverage that investment into the future. Software product delivered to the user at the end of development cycle is only the initial version of the system. The usability and operational life of the software product depend heavily on its maintainability. Maintainability, like all high-level quality attributes, needs to be built into the product. Not all software products possess such attributes built into the product, and as a result, software maintenance has become an issue of major concern to all. Even a high quality software product needs maintenance to adapt to the changing user environments and expectations, to maintain the built-in quality attributes during the transitional phases. Software maintenance has long been regarded as a reactive practice, its primary objective being to maintain existing computer-based applications in service. This attitude is changing, as renovation, encapsulation, merging, information extraction and incorporation of the very fundamentals of systems into reworked business processes. They offer an unprecedented software maintenance challenge.