Introduction to Software Engineering

1

The nature and complexity of software have changed significantly in the last 30 years. In the 1970s, applications ran on a single processor, produced alphanumeric output, and received their input from a linear source. Today's applications are far more complex; typically have graphical user interface and client-server architecture. They frequently run on two or more processors, under different operating systems, and on geographically distributed machines.

Rarely, in history has a field of endeavor evolved as rapidly as software development. The struggle to stay, abreast of new technology, deal with accumulated development backlogs, and cope with people issues has become a treadmill race, as software groups work as hard as they can, just to stay in place. The initial concept of one "guru", indispensable to a project and hostage to its continued maintenance has changed. The Software Engineering Institute (SEI) and group of "gurus" advise us to improve our development process. Improvement means "ready to change". Not every member of an organization feels the need to change. It is too easy to dismiss process improvement efforts as just the latest management fad. Therein lie the seeds of conflict, as some members of a team embrace new ways of working, while others mutter "over my dead body" [WIEG94].

Therefore, there is an urgent need to adopt software engineering concepts, strategies, practices to avoid conflict, and to improve the software development process in order to deliver good quality maintainable software in time and within budget.

1.1 THE EVOLVING ROLE OF SOFTWARE

The software has seen many changes since its inception. After all, it has evolved over the period of time against all odds and adverse circumstances. Computer industry has also progressed at a break-neck speed through the computer revolution, and recently, the network revolution triggered and/or accelerated by the explosive spread of the internet and most recently the web. Computer industry has been delivering exponential improvement in price-performance, but the problems with software have not been decreasing. Software still come late, exceed budget and are full of residual faults. As per the latest IBM report, "31% of the projects get cancelled before they are completed, 53% over-run their cost estimates by an average of 189% and for every 100 projects, there are 94 restarts" [IBMG2K].

1.1.1 Some Software Failures

A major problem of software industry is its inability to develop bug free software. If software developers are asked to certify that the developed software is bug free, no software would have

ever been released. Hence, "software crisis" has become a fixture of everyday life. Many well published failures have had not only major economic impact but also become the cause of death of many human beings. Some of the failures are discussed below :

(*i*) The Y2K problem was the most crucial problem of last century. It was simply the ignorance about the adequacy or otherwise of using only last two digits of the year. The 4-digit date format, like 1964, was shortened to 2-digit format, like 64. The developers could not visualise the problem of year 2000. Millions of rupees have been spent to handle this practically non-existent problem.

(*ii*) The "star wars" program of USA produced "Patriot missile" and was used first time in Gulf war. Patriot missiles were used as a defence for Iraqi Scud missiles. The Patriot missiles failed several times to hit Scud missiles, including one that killed 28 U.S. soldiers in Dhahran, Saudi Arabia. A review team was constituted to find the reason and result was software bug. A small timing error in the system's clock accumulated to the point that after 14 hours, the tracking system was no longer accurate. In the Dhahran attack, the system had been operating for more than 100 hours.

(*iii*) In 1996, a US consumer group embarked on an 18-month, $1 million project to replace its customer database. The new system was delivered on time but did not work as promised, handling routine transactions smoothly but tripping over more complex ones. Within three weeks the database was shutdown, transactions were processed by hand and a new team was brought in to rebuild the system. Possible reasons for such a failure may be that the design team was over optimistic in agreeing to requirements and developers became fixated on deadlines, allowing errors to be ignored.

(*iv*) "One little bug, one big crash" of Ariane-5 space rocket, developed at a cost of $7000 M over a 10 year period. The space rocket was destroyed after 39 seconds of its launch, at an altitude of two and a half miles alongwith its payload of four expensive and uninsured scientific satellites. The reason was very simple. When the guidance system's own computer tried to convert one piece of data—the sideways velocity of the rocket—from a 64-bit format to a 16-bit format; the number was too big, and an overflow error resulted after 36.7 seconds. When the guidance system shutdown, it passed control to an identical, redundant unit, which was there to provide backup in case of just such a failure. Unfortunately, the second unit had failed in the identical manner a few milliseconds before. In this case, the developers had decided that this particular velocity figure would never be large enough to cause trouble—after all, it never had been before.

(*v*) Financial software is an essential part of any company's I.T. infrastructure. However, many companies have experienced failures in their accounting system due to faults in the software itself. The failures ranged from producing the wrong information to the complete system crashing. There is a widespread dissatisfaction over the quality of the financial software. Even if a system only gives incorrect information, this may have an adverse impact on confidence.

(*vi*) Charles C. Mann shared his views about windows XP through his article in technology review [MANN02] as :

> "Microsoft released windows XP on october 25, 2001 and on the same day, which may be a record, the company posted 18 megabytes of patches on its website for bug fixes, compatibility updates, and enhancements. Two patches were intended to fix important

security holes. (One of them did ; the other patch did not work). Microsoft advised (still advises) users to back up critical files before installing patches".

This situation is quite embarrasing and clearly explains the sad state of affairs of present software companies. The developers were either too rushed or too careless to fix obvious defects.

We may keep on discussing the history of software failures which have played with human safety and caused the projects failures in the past.

1.1.2 No Silver Bullet

We have discussed some of the software failures. Is there any light ? Or same scenario would continue for many years. When automobile engineers discuss the cars in the market, they do not say that cars today are no better than they were ten or fifteen years ago. The same is true for aeronautical engineers ; no body says that Boeing or Airbus does not make better quality planes as compared to their previous decade planes. Civil engineers also do not show their anxieties over the quality of today's structures over the structures of last decade. Everyone feels that things are improving day by day. But software, also, seems different. Many software engineers believe that software quality is NOT improving. If anything they say, "it is getting worse".

As we all know, the hardware cost continues to decline drastically. However, there are desperate cries for a silver bullet-something to make software costs drop as rapidly as computer hardware costs do. But as we look to the horizon of a decade, we see no silver bullet. There is no single development, either in technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity.

Inventions in electronic design through transistors and large scale integration has significantly affected the cost, performance and reliability of the computer hardware. No other technology, since civilization began, has seen six orders of magnitude in performance-price gain in 30 years. The progress in software technology is not that rosy due to certain difficulties with this technology. Some of the difficulties are complexity, changeability and invisibility.

The hard part of building software is the specification, design and testing of this conceptual construct, not the labour of representing it and testing the correctness of representation. We still make syntax errors, to be sure, but they are trivial as compared to the conceptual errors (logic errors) in most systems. That is why, building software is always hard and there is inherently no silver bullet.

Many people (especially CASE tool vendors) believe that CASE (Computer Aided Software Engineering) tools represent the so-called silver bullet that would rescue the software industry from the software crisis. Many companies have used these tools and spent large sums of money, but results were highly unsatisfactory, we learnt the hard way that there is no such thing as a silver bullet [BROO87].

The software is evolving day by day and its impact is also increasing on every facet of human life. We cannot imagine a day without using cell phones, logging on to the internet, sending e-mails, watching television and so on. All these activities are dependent on software and software bugs exist nearly everywhere. The blame for these bugs goes to software companies that rush products to market without adequately testing them. It belongs to software

developers who could not understand the importance of detecting and removing faults before the customer experiences them as failures. It belongs to the legal system that has given a free pass to software developers on bug related damages. It also belongs to universities and other higher educational institutions that stress on programming over software engineering principles and practices.

1.2 WHAT IS SOFTWARE ENGINEERING?

Software has become critical to advancement in almost all areas of human endeavour. The art of programming only is no longer sufficient to construct large programs. There are serious problems in the cost, timeliness, maintenance and quality of many software products.

Software engineering has the objective of solving these problems by producing good quality, maintainable software, on time, within budget. To achieve this objective, we have to focus in a disciplined manner on both the quality of the product and on the process used to develop the product.

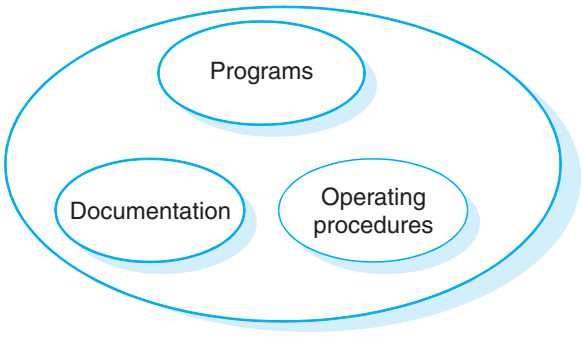
1.2.1 Definition

At the first conference on software engineering in 1968, Fritz Bauer [FRIT68] defined software engineering as *"The establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines"*. Stephen Schach [SCHA90] defined the same as *"A discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements"*.

Both the definitions are popular and acceptable to majority. However, due to increase in cost of maintaining software, objective is now shifting to produce quality software that is maintainable, delivered on time, within budget, and also satisfies its requirements.

1.2.2 Program Versus Software

Software is more than programs. It consists of programs, documentation of any facet of the program and the procedures used to setup and operate the software system. The components of the software systems are shown in Fig. 1.1.



Software = Program + Documentation + Operating Procedures

Fig. 1.1: Components of software

Any program is a subset of software and it becomes software only if documentation and operating procedure manuals are prepared. Program is a combination of source code and object code. Documentation consists of different types of manuals as shown in Fig. 1.2.

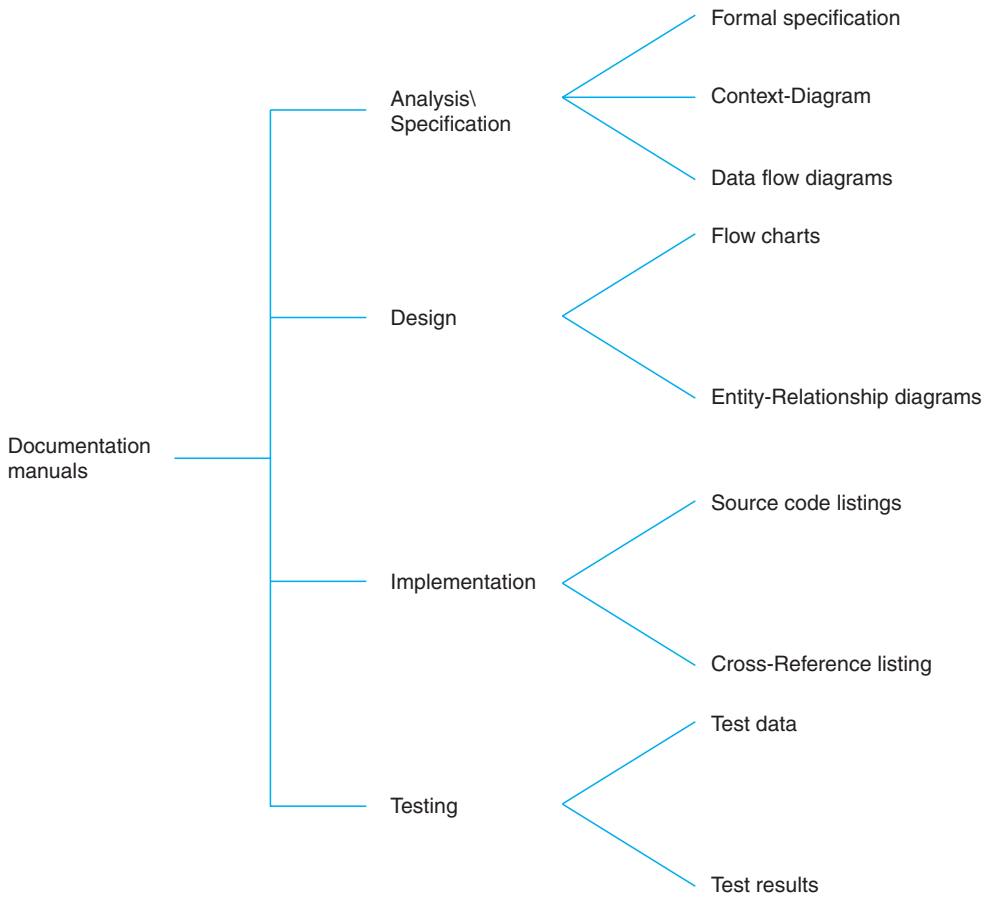


Fig. 1.2: List of documentation manuals

Operating procedures consist of instructions to setup and use the software system and instructions on how to react to system failure. List of operating procedure manuals/documents is given in Fig. 1.3.

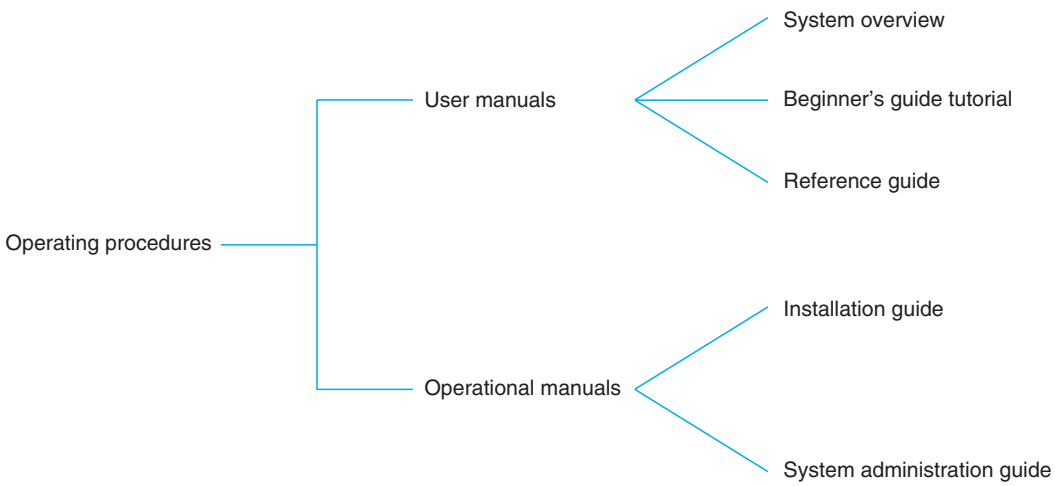


Fig. 1.3: List of operating procedure manuals

1.2.3 Software Process

The software process is the way in which we produce software. This differs from organization to organization. Surviving in the increasingly competitive software business requires more than hiring smart, knowledgeable developers and buying the latest development tools. We also need to use effective software development processes, so that developers can systematically use the best technical and managerial practices to successfully complete their projects. Many software organizations are looking at software process improvement as a way to improve the quality, productivity, predictability of their software development, and maintenance efforts [WIEG96].

It seems straight forward, and the literature has a number of success stories of companies that substantially improved their software development and project management capabilities. However, many other organizations do not manage to achieve significant and lasting improvements in the way they conduct their projects. Here we discuss few reasons why is it difficult to improve software process [HUMP89, WIEG99] ?

1. **Not enough time:** Unrealistic schedules leave insufficient time to do the essential project work. No software groups are sitting around with plenty of spare time to devote to exploring what is wrong with their current development processes and what they should be doing differently. Customers and senior managers are demanding more software, of higher quality in minimum possible time. Therefore, there is always a shortage of time. One consequence is that software organizations may deliver release 1.0 on time, but then they have to ship release 1.01 almost immediately thereafter to fix the recently discovered bugs.

2. **Lack of knowledge:** A second obstacle to widespread process improvement is that many software developers do not seem to be familiar with industry best practices. Normally, software developers do not spend much time reading the literature to find out about the best-known ways of software development. Developers may buy books on Java, Visual Basic or ORACLE, but do not look for anything about process, testing or quality on their bookshelves.

The industry awareness of process improvement frameworks such as the capability maturity model and ISO 9001 for software (discussed in Chapter 7) have grown in recent years, but effective and sensible application still is not that common. Many recognized best practices available in literature simply are not in widespread use in the software development world.

3. **Wrong motivations:** Some organizations launch process improvement initiatives for the wrong reasons. May be an external entity, such as a contractor, demanded that the development organization should achieve CMM level X by date Y. Or perhaps a senior manager learned just enough about the CMM and directed his organization to climb on the CMM bandwagon.

The basic motivation for software process improvement should be to make some of the current difficulties we experience on our projects to go away. Developers are rarely motivated by seemingly arbitrary goals of achieving a higher maturity level or an external certification (ISO 9000) just because someone has decreed it. However, most people should be motivated by the prospect of meeting their commitments, improving customer satisfaction, and delivering excellent products that meet customer expectations. The developers have resisted many process improvement initiatives when they were directed to do "the CMM thing", without a clear explanation of the reasons why improvement was needed and the benefits the team expected to achieve.

4. **Insufficient commitment:** Many times, the software process improvement fails, despite best of intentions, due to lack of true commitment. It starts with a process assessment but fails to follow through with actual changes. Management sets no expectations from the development community around process improvement; they devote insufficient resources, write no improvement plan, develop no roadmap, and pilot no new processes.

The investment we make in process improvement will not have an impact on current productivity; because the time we spend developing better ways to work tomorrow is not available for today's assignment. It can be tempting to abandon the effort when skeptics see the energy they want to be devoted to immediate demands being siphoned off in the hope of a better future (Fig. 1.4). Software organizations should not give up, but should take motivation from the very real, long-term benefits that many companies (including Motorola, Hewlett-Packard, Boeing, Microsoft etc.) have enjoyed from sustained software process improvement initiatives. Improvements will take place over time and organizations should not expect and promise miracles [WIEG2K] and should always remember the learning curve.

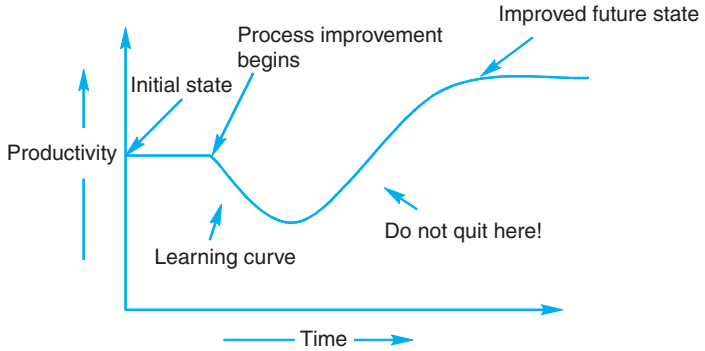


Fig. 1.4: The process improvement learning curve

1.2.4 Software Characteristics

The software has a very special characteristic *e.g.*, "it does not wear out". Its behaviour and nature is quite different than other products of human life. A comparison with one such case, *i.e.,* constructing a bridge vis-a-vis writing a program is given in Table 1.1. Both activities require different processes and have different characteristics.

Table 1.1: A comparison of constructing a bridge and writing a program

Sr. No.	*Constructing a bridge*	*Writing a program*
1.	The problem is well understood.	Only some parts of the problem are understood, others are not.
2.	There are many existing bridges.	Every program is different and designed for special applications.
3.	The requirements for a bridge typically do not change much during construction.	Requirements typically change during all phases of development.
4.	The strength and stability of a bridge can be calculated with reasonable precision.	Not possible to calculate correctness of a program with existing methods.
5.	When a bridge collapses, there is a detailed investigation and report.	When a program fails, the reasons are often unavailable or even deliberately concealed.
6.	Engineers have been constructing bridges for thousands of years.	Developers have been writing programs for 50 years or so.
7.	Materials (wood, stone, iron, steel) and techniques (making joints in wood, carving stone, casting iron) change slowly.	Hardware and software changes rapidly.

Some of the important characteristics are discussed below:

(*i*) **Software does not wear out:** There is a well-known "bath tub curve" in reliability studies for hardware products. The curve is given in Fig. 1.5. The shape of the curve is like "bath tub"; and is known as bath tub curve.

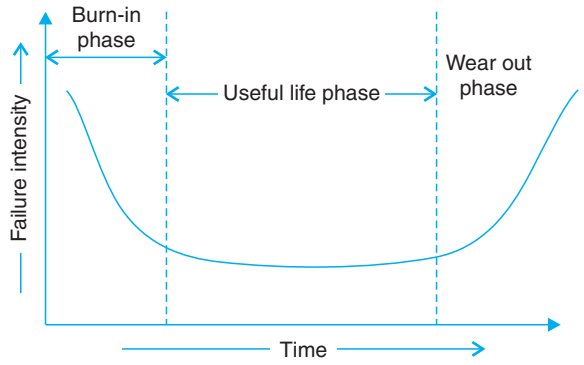


Fig. 1.5: Bath tub curve

There are three phases for the life of a hardware product. Initial phase is burn-in phase, where failure intensity is high. It is expected to test the product in the industry before delivery. Due to testing and fixing faults, failure intensity will come down initially and may stabilise after certain time. The second phase is the useful life phase where failure intensity is approximately constant and is called useful life of a product. After few years, again failure intensity will increase due to wearing out of components. This phase is called wear out phase. We do not have this phase for the software as it does not wear out. The curve for software is given in Fig. 1.6.

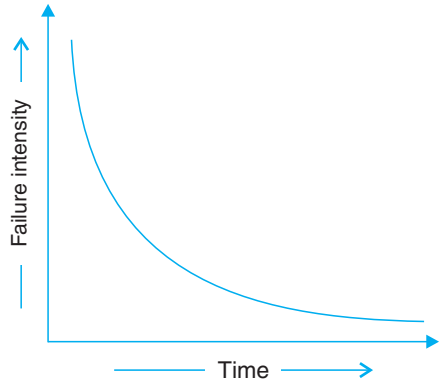


Fig. 1.6: Software curve

Important point is software becomes reliable overtime instead of wearing out. It becomes obsolete, if the environment for which it was developed, changes. Hence software may be retired due to environmental changes, new requirements, new expectations, etc.

(*ii*) **Software is not manufactured:** The life of a software is from concept exploration to the retirement of the software product. It is one time development effort and continuous maintenance effort in order to keep it operational. However, making 1000 copies is not an issue and it does not involve any cost. In case of hardware product, every product costs us due to raw material and other processing expenses. We do not have assembly line in software development. Hence it is not manufactured in the classical sense.

(*iii*) **Reusability of components:** If we have to manufacture a TV, we may purchase picture tube from one vendor, cabinet from another, design card from third and other electronic components from fourth vendor. We will assemble every part and test the product thoroughly to produce a good quality TV. We may be required to manufacture only a few components or no component at all. We purchase every unit and component from the market and produce the finished product. We may have standard quality guidelines and effective processes to produce a good quality product.

In software, every project is a new project. We start from the scratch and design every unit of the software product. Huge effort is required to develop a software which further increases the cost of the software product. However, effort has been made to design standard components that may be used in new projects. Software reusability has introduced another area and is known as component based software engineering.

Hence developers can concentrate on truly innovative elements of design, that is, the parts of the design that represent something new. As explained earlier, in the hardware world, component reuse is a natural part of the engineering process. In software, there is only a humble beginning like graphical user interfaces are built using reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms.

(*iv*) **Software is flexible:** We all feel that software is flexible. A program can be developed to do almost anything. Sometimes, this characteristic may be the best and may help us to accomodate any kind of change. However, most of the times, this "almost anything" characteristic has made software development difficult to plan, monitor and control. This unpredictability is the basis of what has been referred to for the past 30 years as the "Software Crisis".

1.3 THE CHANGING NATURE OF SOFTWARE

Software has become integral part of most of the fields of human life. We name a field and we find the usage of software in that field. Software applications are grouped in to eight areas for convenience as shown in Fig. 1.7.

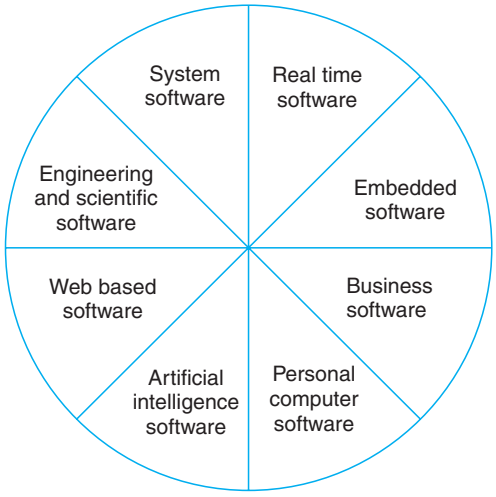


Fig. 1.7: Software applications

(*i*) **System software:** Infrastructure software come under this category like compilers, operating systems, editors, drivers, etc. Basically system software is a collection of programs to provide service to other programs.

(*ii*) **Real time software:** These software are used to monitor, control and analyze real world events as they occur. An example may be software required for weather forcasting. Such software will gather and process the status of temperature, humidity and other environmental parameters to forcast the weather.

(*iii*) **Embedded software:** This type of software is placed in "Read-Only-Memory (ROM)" of the product and control the various functions of the product. The product could be an aircraft, automobile, security system, signalling system, control unit of power plants, etc. The embedded software handles hardware components and is also termed as intelligent software.

(*iv*) **Business software:** This is the largest application area. The software designed to process business applications is called business software. Business software could be payroll, file monitoring system, employee management, account management. It may also be a data warehousing tool which helps us to take decisions based on available data. Management information system, enterprise resource planning (ERP) and such other software are popular examples of business software.

(*v*) **Personal computer software:** The software used in personal computers are covered in this category. Examples are word processors, computer graphics, multimedia and animating

tools, database management, computer games etc. This is a very upcoming area and many big organisations are concentrating their effort here due to large customer base.

(*vi*) **Artificial intelligence software:** Artificial Intelligence software makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis [PRESOI]. Examples are expert systems, artificial neural network, signal processing software etc.

(*vii*) **Web based software:** The software related to web applications come under this category. Examples are CGI, HTML, Java, Perl, DHTML etc.

(*viii*) **Engineering and scientific software:** Scientific and engineering application software are grouped in this category. Huge computing is normally required to process data. Examples are CAD/CAM package, SPSS, MATLAB, Engineering Pro, Circuit analyzers etc.

The expectations from software are increasing in modern civilisation. Software of any of the above groups, has a specialised role to play. Customers and development organisations are desiring more features which may not be always possible to provide. Another trend has emerged to provide source code to the customers and organisations so that they can make modifications for their needs. This trend is particularly visible in infrastructure software like data bases, operating systems, compilers etc. Software where source codes are available, are known as open source. Organisations can develop software applications around such source codes. Some of the examples of "open source software" are LINUX, MySQL, PHP, open office, Apache web server etc. Open source software has risen to great prominence. We may say that these are the programs whose licenses give users the freedom to run the program for any purpose, to study and modify the program, and to redistribute copies of either the original or modified program without paying royalties to original developers. Whether open source software are better than proprietary software ? Answer is not easy. Both schools of thought are in the market. However, popularity of many open source software give confidence to every user. They may also help us to develop small business applications at low cost.

1.4 SOFTWARE MYTHS

There are number of myths associated with software development community. Some of them really affect the way, in which software development should take place. In this section, we list few myths, and discuss their applicability to standard software development [PIER99, LEVE95].

1. **Software is easy to change:** It is true that source code files are easy to edit, but that is quite different than saying that software is easy to change. This is deceptive precisely because source code is so easy to alter. But making changes without introducing errors is extremely difficult, particularly in organizations with poor process maturity. Every change requires that the complete system be re-verified. If we do not take proper care, this will be an extremely tedious and expensive process.

2. **Computers provide greater reliability than the devices they replace:** It is true that software does not fail in the traditional sense. There are no limits to how many times a given piece of code can be executed before it "wears out". In any event, the simple expression of this myth is that our general ledgers are still not perfectly accurate, even though they have

been computerized. Back in the days of manual accounting systems, human error was a fact of life. Now, we have software error as well.

3. **Testing software or 'proving' software correct can remove all the errors:** Testing can only show the presence of errors. It cannot show the absence of errors. Our aim is to design effective test cases in order to find maximum possible errors. The more we test, the more we are confident about our design.

4. **Reusing software increases safety:** This myth is particularly troubling because of the false sense of security that code re-use can create. Code re-use is a very powerful tool that can yield dramatic improvement in development efficiency, but it still requires analysis to determine its suitability and testing to determine if it works.

5. **Software can work right the first time:** If we go to an aeronautical engineer, and ask him to build a jet fighter craft, he will quote us a price. If we demand that it is to be put in production without building a prototype, he will laugh and may refuse the job. Yet, software engineers are often asked to do precisely this sort of work, and they often accept the job.

6. **Software can be designed thoroughly enough to avoid most integration problems:** There is an old saying among software designers: "Too bad, there is no complier for specifications": This points out the fundamental difficulty with detailed specifications. They always have inconsistencies, and there is no computer tool to perform consistency checks on these. Therefore, special care is required to understand the specifications, and if there is an ambiguity, that should be resolved before proceeding for design.

7. **Software with more features is better software:** This is, of course, almost the opposite of the truth. The best, most enduring programs are those which do one thing well.

8. **Addition of more software engineers will make up the delay:** This is not true in most of the cases. By the process of adding more software engineers during the project, we may further delay the project. This does not serve any purpose here, although this may be true for any civil engineering work.

9. **Aim is to develop working programs:** The aim has been shifted from developing working programs to good quality, maintainable programs. Maintaining software has become a very critical and crucial area for software engineering community.

This list is endless. These myths, poor quality of software, increasing cost and delay in the delivery of the software have been the driving forces behind the emergence of software engineering as a discipline. In addition, following are the contributing factors:

- Change in ratio of hardware to software costs
- Increasing importance of maintenance
- Advances in software techniques
- Increased demand for software
- Demand for larger and more complex software systems.

1.5 SOME TERMINOLOGIES

Some terminologies are discussed in this section which are frequently used in the field of Software Engineering.

1.5.1 Deliverables and Milestones

Different deliverables are generated during software development. The examples are source code, user manuals, operating procedure manuals etc.

The milestones are the events that are used to ascertain the status of the project. Finalisation of specification is a milestone. Completion of design documentation is another milestone. The milestones are essential for project planning and management.

1.5.2 Product and Process

Product: What is delivered to the customer, is called a product. It may include source code, specification document, manuals, documentation etc. Basically, it is nothing but a set of deliverables only.

Process: Process is the way in which we produce software. It is the collection of activities that leads to (a part of) a product. An efficient process is required to produce good quality products.

If the process is weak, the end product will undoubtedly suffer, but an obsessive over-reliance on process is also dangerous.

1.5.3 Measures, Metrics and Measurement

The terms measures, metrics and measurement are often used interchangeably. It is interesting to understand the difference amongst these. A measure provides a quantitative indication of the extent, dimension, size, capacity, efficiency, productivity or reliability of some attributes of a product or process.

Measurement is the act of evaluating a measure. A metric is a quantitative measure of the degree to which a system, component, or process possesses a given attribute. Pressman [PRESO2] explained this very effectively with an example as given below:

"When a single data point has been collected (*e.g.,* the number of errors uncovered in the review of a single module), a measure has been established. Measurement occurs as the result of the collection of one or more data points (*e.g.,* a number of module reviews are investigated to collect measures of the number of errors in each module). A software metric relates the individual measures in some way (*e.g.,* the average number of errors found per review)."

Hence we collect measures and develop metrics to improve the software engineering practices.

1.5.4 Software Process and Product Metrics

Software metrics are used to quantitatively characterise different aspects of software process or software products. Process metrics quantify the attributes of software development process and environment; whereas product metrics are measures for the software product. Examples of process metrics include productivity, quality, failure rate, efficiency etc. Examples of product metrics are size, reliability, complexity, functionality etc.

1.5.5 Productivity and Effort

Productivity is defined as the rate of output, or production per unit of effort, *i.e.,* the output achieved with regard to the time taken but irrespective of the cost incurred. Hence, there are two issues for deciding the unit of measure

(*i*) quantity of output

(*ii*) period of time.

In software, one of the measure for quantity of output is lines of code (LOC) produced. Time is measured in days or months.

Hence most appropriate unit of effort is Person Months (PMs), meaning thereby number of persons involved for specified months. So, productivity may be measured as LOC/PM (lines of code produced/person month).

1.5.6 Module and Software Components

There are many definitions of the term module. They range from "a module is a FORTRAN subroutine" to "a module is an Ada Package", to "Procedures and functions of PASCAL and C", to "C++ Java classes" to "Java packages" to "a module is a work assignment for an individual developer". All these definitions are correct. The term subprogram is also used sometimes in place of module.

There are many definitions of software components. A general definition given by Alan W. Brown [BROW2K] is:

"An independently deliverable piece of functionality providing access to its services through interfaces".

Another definition from unified modeling language (UML) [OMG2K] is:

"A component represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces".

Hence, a reusable module is an independent and deliverable software part that encapsulates a functional specification and implementation for reuse by a third party.

However, a reusable component is an independent, deployable, and replaceable software unit that is reusable by a third party based on unit's specification, implementation, and well defined contracted interfaces.

1.5.7 Generic and Customised Software Products

The software products are divided in two categories:

(*i*) Generic products

(*ii*) Customised products.

Generic products are developed for anonymous customers. The target is generally the entire world and many copies are expected to be sold. Infrastructure software like operating systems, compilers, analysers, word processors, CASE tools etc. are covered in this category.

The customised products are developed for particular customers. The specific product is designed and developed as per customer requirements. Most of the development projects (say about 80%) come under this category.

1.6 ROLE OF MANAGEMENT IN SOFTWARE DEVELOPMENT

The management of software development is heavily dependent on four factors: People, Product, Process, and Project. Order of dependency is as shown in Fig. 1.8.

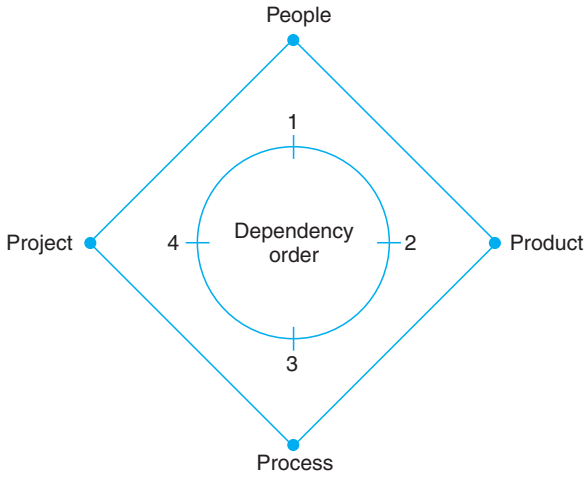


Fig. 1.8: Factors of management dependency (from People to Project)

Software development is a people centric activity. Hence, success of the project is on the shoulders of the people who are involved in the development.

1.6.1 The People

Software development requires good managers. The managers, who can understand the psychology of people and provide good leadership. A good manager cannot ensure the success of the project, but can increase the probability of success. The areas to be given priority are: proper selection, training, compensation, career development, work culture etc.

Managers face challenges. It requires mental toughness to endure inner pain. We need to plan for the best, be prepared for the worst, expect surprises, but continue to move forward anyway. Charles Maurice once rightly said "I am more afraid of an army of one hundred sheep led by a lion than an army of one hundred lions led by a sheep".

Hence, manager selection is most crucial and critical. After having a good manager, project is in safe hands. It is the responsibility of a manager to manage, motivate, encourage, guide and control the people of his/her team.

1.6.2 The Product

What do we want to deliver to the customer? Obviously, a product; a solution to his/her problems.

Hence, objectives and scope of work should be defined clearly to understand the requirements. Alternate solutions should be discussed. It may help the managers to select a "best" approach within constraints imposed by delivery deadlines, budgetary restrictions, personnel availability, technical interfaces etc. Without well defined requirements, it may be

impossible to define reasonable estimates of the cost, development time and schedule for the project.

1.6.3 The Process

The process is the way in which we produce software. It provides the framework from which a comprehensive plan for software development can be established. If the process is weak, the end product will undoubtedly suffer. There are many life cycle models and process improvements models. Depending on the type of project, a suitable model is to be selected. Now-a-days CMM (Capability Maturity Model) has become almost a standard for process framework. The process priority is after people and product, however, it plays very critical role for the success of the project. A small number of framework activities are applicable to all software projects, regardless of their size and complexity. A number of different task sets, tasks, milestones, work products, and quality assurance points, enable the framework activities to be adopted to the characteristics of the project and the requirements of the project team.

1.6.4 The Project

A proper planning is required to monitor the status of development and to control the complexity. Most of the projects are coming late with cost overruns of more than 100%. In order to manage a successful project, we must understand what can go wrong and how to do it right. We should define concrete requirements (although very difficult) and freeze these requirements. Changes should not be incorporated to avoid software surprises. Software surprises are always risky and we should minimise them. We should have a planning mechanism to give warning before the occurrence of any surprise.

All four factors (People, Product, Process and Project) are important for the success of the project. Their relative importance helps us to organise development activities in more scientific and professional way.

Software Engineering has become very important discipline of study, practice and research. All are working hard to minimise the problems and to meet the objective of developing good quality maintainable software that is delivered on time, within budget, and also satisfies the requirements. With all cries and dissatisfaction, discipline is improving and maturing day by day. New solutions are being provided in the niche areas and encouraging results are being observed. We do feel that with in couple of years, situation is bound to improve and software engineering shall be a stable and mature discipline.

REFERENCES

[BOEH89] Boehm B., "*Risk Management*", IEEE Computer Society Press, 1989.

[BROO87] Brooks F.P., "*No Silver Bullet : Essence and Accidents of Software Engineering*", IEEE Computer, 10—19, April, 1987.

[BROW2K] Brown A.W., "*Large Scale, Component based Development*", Englewood cliffs, NJ, PH, 2000.

[FRIT68] Bauer, Fritz et al., "*Software Engineering: A Report on a Conference Sponsored by NATO Science Committee*", NATO, 1968.

[HUMP89] Humphrey W.S., "*Managing the Software Process*", Addison-Wesley Pub. Co., Reading, Massachusetts, USA, 1989.

[IBMG2K] IBM Global Services India Pvt. Ltd., Golden Tower, Airport Road, Bangalore, Letter of Bindu Subramani, Segment Manager—Corporate Training, March 9, 2000.

[LEVE95] Leveson N.G., "*Software, System Safety and Computers*", Addison Wesley, 1995.

[MANN02] Charles C.Mann, "*Why Software is so Bad*", Technology Review, www. technologyreview.com. 2002.

[OMG2K] OMG Unified Modelling Language Specification, Version 1.4, Object Mangement Group, 2000.

[PRES02] Pressman R., "*Software Engineering*", McGraw Hill, 2002.

[PIER99] Piersal K., "*Amusing Software Myths*", www. bejeeber.org/Software-myths.html, 1999.

[SCHA90] Schach, Stephen, "*Software Engineering*", Vanderbilt University, Aksen Association, 1990.

[WIEG2K] Wiegers K.E., "*Stop Promising Miracles*" Software Development Magazine, February, 2000.

[WIEG94] Wiegers K.E., "*Creating a Software Engineering Culture*", Software Development Magazine, July, 1994.

[WIEG96] Wiegers K.E., "*Software Process Improvement: Ten Traps to Avoid*", Software Development Magazine, May, 1996.

[WIEG99] Wiegers K.E., "*Why is Process Improvement So Hard*", Software Development Magazine, February, 1999.

MULTIPLE CHOICE QUESTIONS

Note : Select most appropriate answer of the following questions.

1.1. Software is

(*a*) superset of programs (*b*) subset of programs

(*c*) set of programs (*d*) none of the above.

1.2. Which is NOT the part of operating procedure manuals?

(*a*) user manuals (*b*) operational manuals

(*c*) documentation manuals (*d*) installation manuals.

1.3. Which is NOT a software characteristic?

(*a*) software does not wear out (*b*) software is flexible

(*c*) software is not manufactured (*d*) software is always correct.

1.4. Product is

(*a*) deliverables (*b*) user expectations

(*c*) organisation's effort in development (*d*) none of the above.

1.5. To produce a good quality product, process should be

(*a*) complex (*b*) efficient

(*c*) rigorous (*d*) none of the above.

1.6. Which is not a product metric?

(*a*) size (*b*) reliability

(*c*) productivity (*d*) functionality.

1.7. Which is not a process metric?
(*a*) productivity (*b*) functionality
(*c*) quality (*d*) efficiency.

1.8. Effort is measured in terms of:
(*a*) person-months (*b*) rupees
(*c*) persons (*d*) months.

1.9. UML stands for
(*a*) uniform modeling language (*b*) unified modeling language
(*c*) unit modeling language (*d*) universal modeling language.

1.10. An independently deliverable piece of functionality providing access to its services through interfaces is called
(*a*) software measurement (*b*) software composition
(*c*) software measure (*d*) software component.

1.11. Infrastructure software are covered under
(*a*) generic products (*b*) customised products
(*c*) generic and customised products (*d*) none of the above.

1.12. Management of software development is dependent on
(*a*) people (*b*) product
(*c*) process (*d*) all of the above.

1.13. During software development, which factor is most crucial?
(*a*) people (*b*) product
(*c*) process (*d*) project.

1.14. Program is
(*a*) subset of software (*b*) super set of software
(*c*) software (*d*) none of the above.

1.15. Milestones are used to
(*a*) know the cost of the project (*b*) know the status of the project
(*c*) know user expectations (*d*) none of the above.

1.16. The term module used during design phase refers to
(*a*) function (*b*) procedure
(*c*) sub program (*d*) all of the above.

1.17. Software consists of
(*a*) set of instructions + operating system
(*b*) programs + documentation + operating procedures
(c) programs + hardware manuals (*d*) set of programs.

1.18. Software engineering approach is used to achieve:
(*a*) better performance of hardware (*b*) error free software
(*c*) reusable software (*d*) quality software product.

1.19. Concepts of software engineering are applicable to
(*a*) fortran language only (*b*) pascal language only
(*c*) 'C' language only (*d*) all of the above.

1.20. CASE Tool is

(*a*) computer Aided Software Engineering (*b*) component Aided Software Engineering

(*c*) constructive Aided Software Engineering (*d*) computer Analysis Software Engineering.

EXERCISE

1.1. Why is the primary goal of software development now shifting from producing good quality software to good quality maintainable software?

1.2. List the reasons for the "software crisis"? Why are CASE tools not normally able to control it?

1.3. "The software crisis is aggravated by the progress in hardware technology?" Explain with examples.

1.4. What is software crisis? Was Y2K a software crisis?

1.5. What is the significance of software crisis in reference to software engineering discipline.

1.6. How are software myths affecting software process? Explain with the help of examples.

1.7. State the difference between program and software. Why have documents and documentation become very important?

1.8. What is software engineering? Is it an art, craft or a science? Discuss.

1.9. What is the aim of software engineering? What does the discipline of software engineering discuss?

1.10. Define the term "Software Engineering". Explain the major differences between software engineering and other traditional engineering disciplines.

1.11. What is software process? Why is it difficult to improve it?

1.12. Describe the characteristics of software contrasting it with the characteristics of hardware.

1.13. Write down the major characteristics of a software. Illustrate with a diagram that the software does not wear out.

1.14. What are the components of a software? Discuss how a software differs from a program.

1.15. Discuss major areas of the applications of the software.

1.16. Is software a product or process? Justify your answer with examples.

1.17. Differentiate between the followings

(*i*) Deliverables and milestones (*ii*) Product and process

(*iii*) Measures, metrics and measurement

1.18. What is software metric? How is it different from software measurement?

1.19. Discuss software process and product metrics with the help of examples.

1.20. What is productivity? How is it related to effort? What is the unit of effort?

1.21. Differentiate between module and software component.

1.22. Distinguish between generic and customised software products. Which one has larger share of market and why?

1.23. Describe the role of management in software development with the help of examples.

1.24. What are various factors of management dependency in software development? Discuss each factor in detail.

1.25. What is more important: Product or process? Justify your answer.