

## Chapter 4

### Current Technology

#### 4.1 Overview

The evolution of small mini PC has started earlier itself. Many companies and research scholars involved in similar kind of project more than 2 years. However, no one has come to an end without problems.

Most of the companies are facing problems with the input devices and recommending standard input device like mouse and keyboard for Carputer, which is not an ideal solution to operate while driving a car. On the other hand they are facing problems while integrating the Carputer in to the car dashboard since each and every company designing their own PC unit hardware.

#### 4.2 Xenarc Technologies.

Xenarc Technologies a leading LCD Monitor manufacturing company in USA currently involved in the project called “*Mini P3 Aluminum Car PC*”.



Figure 9: “7 inch LCD Touch Screen.

The company used “7 inch LCD touch screen” for the project in order to replace the standard input devices like mouse and keyboard shown in Figure 9. However the touch screen option is not so good enough to operate while driving a car also to select small menus since the screen is only 7 inches. Also LCD screen freezes in abnormal cold weather condition. The below Figure 10 shows the view of the Mini P3 Aluminium Car PC [2].

### 4.2.1 Mini P3 Aluminium Car PC

Vehicle Personal Computer opens up the world of in-vehicle entertainment. Couple this multimedia PC with the portable high-resolution VGA monitors and other computer peripherals, makes the system capable for GPS Navigation, MP3 playback, in-car theatre, in-car gaming, and in-car office.

This offers superior 2D/3D graphic performance and Soft DVD MPEG-2 playback with hardware motion compensation which allows its user to experience life like audio and video. Use this unit as a music and movie jukebox by downloading songs and movies into its built-in hard drive is possible. Since this is an open-architecture, customer can use their creativity to create customized in-car entertainment system.

Compact and light - with a footprint of 5.75" x 9.84" x 1.65" and weighing in at 2 lbs, this mini Pentium 3 based computer system has aluminium casing and takes 12V DC, making it perfect for in-vehicle computing use. Customers can install their choice of a CD-ROM, DVD-ROM, CD-RW, or DVD-ROM/CD-RW combo drive. The draw back of this specified product is there is no input devices mentioned along with the product [2].



Figure 10: *Mini P3 Aluminium Car PC*



Figure 11: *Mini P3 Aluminium Car PC*

The updated version of the above mentioned product is “*XENARC CP-1000™ In-Dash 1-DIN Car PC*” shown in Figure 10. Also in the updated latest version the company recommends to use external input devices like mouse and keyboard in CAR [2].

#### 4.3 Peter Bridger a one man company

Peter Bridger has a one man company in United Kingdom involved in the Carputer project for about more than 2 years. He experienced so many problems while implementing a computer in a CAR. His experience gives motivation to think on various sides to overcome the problems in the master thesis. His first try to keep the mother board of the computer in the dash board case as shown in Figure 12. The monitor was placed near the gearbox just below the music system provided by the car manufacturer. The setup is shown in Figure 12. A full view of the LCD monitor located near the gearbox is shown below in Figure 14 [3].



Figure 12: Motherboard in dashboard case.



Figure 13: “7 inch” Touch Screen Display near the Gear Box



Figure 14: Full view of the LCD monitor located near the gearbox

#### **4.3.1 Experience**

One of the main uses of Carputer is for GPS Navigation route planning. He made a little round up of his experiences with different route planning software. He got a GPS receiver along with software Route 66 shown in Figure 13. It is an average piece of software that talks to the GPS unit and can plan routes from the maps available.

Info Map is a powerful software shown in Figure 15 can able to update the map with the GPS data every second and even leaves a little trail of where you have been. As for route planning it is pretty good.

AutoRoute is Microsoft product shown in Figure 17 offering a very easy use and a very good all rounder for GPS Navigation. For gaining information about near by landmarks, cash points and other such places of interests this software plays a vital role to the best of customer's satisfaction [3].



Figure 15: GPS software Route 66.



Figure 16 : GPS software Infomap



Figure 17: GPS software Auto Route.

### **4.3.2 Realisation**

Fixing a computer into a car is not a daily procedure; he didn't have any instructions to follow or beaten path to take when working on the project. This meant in reflection, some choices he made were not the best, while others worked out ok. This sections deal the positive and negative sides of the project [3].

#### **Carputer Display**

Computer monitors are not cheap and are the most expensive part of any computer system. Trying to find a small and cheap display for this project wasn't easy, and was one of the biggest sticking points in the early days. He finally found a good priced monitor at Merconnet, which was quite big at 5.6" compared to others he found it was fairly reasonably priced. Admittedly it is not perfect, but for the price he needs to do with it. Mostly the LCD touch screen monitors are bit expensive.

#### **Voice Command**

Programming his own voice command software in Visual Basic was pretty straight forward and enabled his product to play any song at will. Without diverting the drivers attention from the road is an added advantage and he planned to improve the software for the next version.

## **GPS**

One of the things that he wanted from the start, but only came together towards the end is the GPS Navigation system. Finally he could watch the map zoom into his current position, and the navigation map software works perfectly updating every second.

## **DVD Playback**

He did not have major worries about the DVD playback process with 800MHz Pentium 3 processor. He picked up a cheap DVD drive which came with a copy of Power DVD that solves the DVD Playback worries.

## **Keyboard / Mouse Interface**

Both the standard input device mouse and keyboard are very small but very functional, perfect for the task in hand. The keyboard was a bit pricey, but it is a solid piece of kit. The mouse was also very usable and durable.

## **Size of Components**

The whole idea really started after seeing the size of the mini-ITX Eden motherboard and he had been lucky enough to find a whole host of other components that have compromised the project to be very small.

## **Windows 2000 Operating System**

His main home computer is an Athlon 2000 XP + powered rig, nice and up to date, but he was using W2K on it rather than XP. Because he felt 2K is slicker and cleaner.

When he first tested the Carputer's Eden motherboard with 64MB stick of RAM in it. Windows 2000 ran perfectly on it, starting up very fast. However when he upgraded it to Windows XP, it slowed to a snails pace. The Windows 2000 installation had been stripped down all non essential services so that the system starts very quickly. It was very stable and runs whatever great stuff software he installed on it.

## **What Went Wrong**

A big project like this could not have been a smooth ride and of course it was not. Things did not always turn out as he thought they would. So what exactly went wrong?

## **No Touch Screen**

One of the ways he hoped to keep this vision was by having a touch screen interface, so that it will replace the standard input devices like mouse and keyboard. Sadly touch-screen devices are expensive and causes unusual problem in cold weather conditions. Finally he found such a device, a 7" SVGA touch screen monitor for about \$499 price which was not certainly cheap.

## **Monitor Size and Quality**

The LCD small screen showed a good look appearance because of its 5.6" monitor size however it is very small and not flexible for the customer's usage. Also it was not very clear since it takes a TV video signal rather than a standard VGA monitor signal. By using large icons and font plus some high contrast colours the situation can be improved. Although at the end of the day, many applications can be a real strain to read and use. By using custom designed interfaces the problems can be rectified.

## **Power Efficiency**

The Carputer takes the 12DC from the car battery, uses an inverter to transform this into 240AC which then makes its way into the Carputers PSU that then transforms this into the 12DC, 5DC, etc... With respect to the computer needs. Now the project concentrates to use the power directly without power loss.

## **Only a Few Voice Commands**

Voice recognition is very useful and powerful, however it is only written to playback music. He would like to extend the system in the next version so that he can control every part of the Carputer with voice, so the customer never needs to take eyes off from the road.

## **What Could Be Improved?**

The above sections described the good and the bad, now what can be improved on and used in Carputer version 2?

### **Larger and Clearer Screen**

Much of what the Carputer offers is visual, from GPS maps to DVD films. So for this reason a good monitor is a must, whether this will be achieved from merely using a large 7"/10" screen or a monitor using a VGA input, has to be considered to concentrate the key things to improve the next version of the project.

### **Better Performance**

Surely the C3 processor will never compete with a P4 chip, but considering every drop of power he was always eager to make more power, more performance of the system. Even though the system has Quake3, Elite Force and the Sims running on the Carputer, he was interested always for further enhancements.

The new range of mini-ITX boards from VIA look great, with a 933MHz C3 chip, enhanced graphics and hardware assisted DVD decoding. Probably the next version will be released with the above mentioned configuration.

### **Wireless LAN**

Currently software can be loaded onto the Carputer like normal PC operations just by popping a CD or DVD. For better data transfer a wireless LAN would be an ideal solution, it needs two USB wireless modules and things would be made much easier.

### **Auto Starting**

The next enhancement of the Carputer is to start the system automatically during car start-up. No more burdens to turn it on, it'll always be there, ready for GPS, games, DVD or anything the customer needs.

## **FM Sound Transmitter**

In the current system, the soundcard of the Carputer goes through a CD-Tape adapter into the cars stereo system, not exactly comparable to an optical audio connection. So in version 2 the system will be using a mini FM radio transmitter, so the customer can just tune the car radio into the Carputers broadcast.

## **Enhanced GPS**

As it stands the GPS does a great job, but it could do better? The system should plan a route and should instruct the customer when to turn left, right or whatever. The customer should put total faith in the GPS system to tell exactly where to go, just by talking.

## **4.4 Overall Impression**

Lot of companies and research scholars from UK, USA and Canada are deeply involved in the same project for more than years, however with the input device each company facing different problems by implementing standard or customized input devices, also facing problems to fix it in car dash board.

# Chapter 5

## Software Development Process

### 5.1 Introduction

A *software development process* is a process used to develop computer software, which refers to one or more computer programs and data held in the storage of a computer for some purpose. Software program performs the function of the program it implements, either by directly providing instructions to the computer hardware or by serving as input to another piece of software. Data software exists solely for its eventual use by other program software. It may be an ad hoc process, devised by the team for one *project* but the term often refers to a standardised, documented *methodology* which has been used before on similar projects or one which is used habitually within an organisation [20].

### 5.2 Software Engineering

#### 5.2.1 Introduction

Software Engineering is the profession concerned with creating and maintaining software applications by applying computer science, project management and other technologies and practices.

Software Engineering applications are used in wide range of activities, from industry to entertainment. Software applications improve user productivity and quality of life. Application software examples are office suites, video games, and the World Wide Web. System software examples are embedded systems and operating systems.

In 1967, a NATO study group coined the phrase "*software engineering*" as a response to their belief that the current software crisis could be solved by adapting existing engineering practices to software development. This crisis was characterized by the consistent development of low quality software which exceeded cost limits and development deadlines. Twenty years later, the software crisis was still thriving. Consider the following analysis done by the Government Accounting Office (GAO) in 1979 on the state of management information systems software development [20].

Out of the 163 contractors and 113 government personnel surveyed,

- 60% of their contracts had schedule overruns,
- 50% of their contracts had cost overruns,
- 45% of the software could not be used,
- 29% of the software was never delivered, and
- 19% of the software had to be reworked to be used.

Even today the software crisis is a significant problem that software engineering must address. Dr. John Dalbey of California Polytechnic State University organized the following information about the current state of the software crisis.

### **5.2.2 Software Chronic Crisis Recognition**

In the September 1994 issue of Scientific American, W. Wayt Gibbs chronicled the following example of the modern software crisis in his article *"Software's Chronic Crisis"*.

"Denver's new international airport was to be the pride of the Rockies, a wonder of modern engineering. Twice the size of Manhattan, 10 times the breadth of Heathrow, the airport is big enough to land three jets simultaneously in bad weather. Even more impressive than its girth is the airport's subterranean baggage-handling system. Tearing like intelligent coal-mine cars along 21 miles of steel track, 4,000 independent "telecars" route and deliver luggage between the counters, gates and claim areas of 20 different airlines. A central nervous system of some 100 computers networked to one another and to 5,000 electric eyes, 400 radio receivers and 56 bar-code scanners orchestrates the safe and timely arrival of every valise and ski bag.

At least that is the plan. For nine months, this Gulliver has been held captive by Lilliputians-errors in the software that controls its automated baggage system. Scheduled for takeoff by last Halloween, the airport's grand opening was postponed until December to allow BAE Automated Systems time to flush the gremlins out of its \$193-million system. December yielded to March. March slipped to May. In June the airport's planners, their bond rating demoted to junk and their budget haemorrhaging red ink at the rate of \$1.1 million a day in interest and operating costs, conceded that they could not predict when the baggage system would stabilize enough for the airport to open" [21].

Eventually the Denver International Airport (DIA) did open, but the advanced baggage system was only partially functioning. The four delayed openings of the airport caused many residents to speculate that DIA really stood for "Do It Again," "Doesn't Include Airlines," or "Done In April". In order to finally open the terminal, the city invested \$51 million to install a conventional baggage system as a work around to the high-tech system. Ironically, the conventional system was completed four weeks ahead of schedule and \$3.4 million under budget [21 & 23].

*The obvious question is: why was the high-tech system so difficult to implement?*

According to Fred Brooks a scientific software analyst, part of the answer to this question is that software is inherently complex. Unlike other engineering disciplines, software systems lack repeated elements. While a building may be constructed of thousands of bricks, a software product will combine pieces with the same functionality into a single subroutine. This means that software is composed of thousands of unique parts rather than repeated parts. Software systems also have very large numbers of operational states which makes exhaustive testing impossible. A bridge, on the other hand, is also a large and complex structure, but only a handful of extreme states (i.e. inclement weather, heavy traffic, earthquakes) need to be tested to insure the reliability of the bridge. In addition to inherent *complexity*, Brooks also mentions some other essential qualities of software such as *changeability* and *invisibility* that contribute to the software crisis. Changeability refers to the fact that all software eventually gets changed. Clients may want to add new functionality or developers may want to port the program to a new hardware platform. While it is unthinkable that a civil engineer would be asked to move a bridge to a new location, software engineers are regularly expected to perform major modifications on existing software. Invisibility refers to the fact that software is not a physical entity. Because of this, it is difficult for the human mind to use some of its most powerful conceptual tools in the development of software.

With these difficulties in mind, the need for effective software engineering becomes even more urgent. Software systems are playing an increasingly common role in our everyday lives. Failure to develop reliable software systems can cost more than just money and time. Consider the following examples cited by Michael Lyu in the introduction to the *Handbook of Software Reliability Engineering*:

"Unfortunately, software can also kill people. The massive Therac-25 radiation therapy machine had enjoyed a perfect safety record until software errors in its sophisticated control systems malfunctioned and claimed several patients' lives in 1985 and 1986. On October 26, 1992, the Computer Aided Dispatch system of the London Ambulance Service broke down right after its installation, paralyzing the capability of the world's largest ambulance service to handle 5000 daily requests in carrying patients in emergency situations. In the recent aviation industry, although the real causes for several airliner crashes in the past few years remained mysteries, experts pointed out that software control could be the chief suspect in some of these incidences due to its inappropriate response to the pilots' desperate inquiries during abnormal flight conditions"

In further sections, we will examine some of the fundamental concepts of software engineering such as processes and meta processes, life cycle of software development, two major paradigms of developing software and Software Quality Characteristics [21].

### **5.2.3 What is a project?**

A project is a temporary endeavour undertaken to create a unique product or service. Temporary means that the project has an end date. Unique means that the project's end result is different than the results of other functions of the organization. It can also comprise an ambitious plan to define and constrain a future by limiting it to set goals and parameters. The planning, execution and monitoring of major projects sometimes involves setting up a special temporary organization, consisting of a project team and one or more work teams. A project usually needs resources.

### **5.2.4 Project lifecycle**

In software engineering, a project lifecycle models how a project is planned, controlled and monitored from its inception to its completion. In the earliest stages and in the last stages of the project lifecycle, Software architecture, requirements and system definition is an issue:

- What is the market for the system/software and its competition?
- What is the platform of the system/software?
- How much time is available to the project?
- How many skill-sets are needed?

Over the years, a number of different models have been developed, beginning with the oldest and simplest being the "*Waterfall Model*" [21].

However, as software has become larger and more complex, this method of development has been found to be counter productive, especially when large teams are involved. Models that are iterative have evolved including *Prototyping*, *Evolutionary Prototyping*, *Incremental Development*, *spiral model*, *V modal*, and *Chaos model*. The use of these models for the most part confined to the overall management of the project. However projects are now considered better controlled if the model best suited to the individual aspects of the project. For example, a project may be managed using the Incremental Development model, but during each increment the Documentation is created managed using the Waterfall Model and the Code development managed using the 'V' model.

The level of formality and complexity of the lifecycle for each project is constrained by any number of factors, including *budgetary constraints*, *experience*, *size and complexity* of the project and development team.

Some experienced and highly respected project leaders and programmers consider rigid application of lifecycle plans to be a theory that does not work well in practice, the very highly regarded project leader of the Linux kernel made the following statement on the Linux kernel mailing list:

*"No major software project that has been successful in a general marketplace has ever gone through those nice lifecycles".*

*ISO 12207* is a standard that was developed to describe the method of *selecting, implementing and monitoring a lifecycle for a project*.

### **5.2.5 Project Methodology**

In software engineering and project management, the term *methodology* is a codified set of recommended practices, sometimes accompanied by training materials, formal educational programs, worksheets and diagramming tools. Software engineering methods span many disciplines, including project management, analysis, specification, design, coding, testing, and quality assurance. All of the methods guiding this field are collations of all these disciplines [21].

### 5.2.6 Organisation

An organization is a formal group of people with one or more shared goals. This topic is a broad one. According to management science, most human organizations fall roughly into five types:-

- Pyramids or hierarchies
- Committees or juries
- Matrix organisations
- Ecologies
- Composite organisations

Some managers, who are held accountable for software development, may seek to find the commonalities in the efforts of their organizations. If those managers are process-oriented, rather than people-oriented, task-oriented, profit-oriented, project-oriented, etc. Then they may seek methodologies or other *proxies* which can serve as templates for the software development process.

Of course, it is entirely rational for other managers who are not process-oriented to use a documented *software development process* or *methodology*. In such a case one might say that the methodology is used by them as a "*proxy*" for the necessary set of process-oriented skills required in any software engineering project [21].

## 5.3 Processes and meta-processes

A growing body of software development organisations implement process methodologies. Many of them are in the defence industry, which in the U.S. requires a "Rating" based on 'Process models' to obtain contracts [21].

*Rating* is a means of classifying things in different categories. Some of them are:-

- ELO rating system
- Suitability-to-audience ratings
- Motion picture rating systems
- MPAA film rating system
- Film rating systems
- TV Parental Guidelines

- V – chip ratings
- Entertainment Software Rating Board
- Fuel performance ratings
- Octane rating
- Cetane rating
- PR rating
- Nielsen Ratings
- Naval rating

### **5.3.1 Capability Maturity Model**

The *Capability Maturity Model (CMM)* grades organizations on how well they create software according to how they define and execute their processes.

The Capability Maturity Model of the Software Engineering Institute at Carnegie Mellon University describes the maturity of software development organisations on a scale of 1 to 5. It has been used extensively for avionics software and for government projects since it was created in the mid 1980's. The Software Engineering Institute has subsequently released a revised version known as the Capability Maturity Model Integration (CMMI) [21].

### **5.3.2 ISO 9000**

ISO 9000 describes standards for formally organizing processes with documentation. It specifies requirements for a Quality Management System overseeing the production of a product or service. It is not a standard for ensuring a product or service is of quality; rather, it attests to the process of production, and how it will be managed and reviewed.

It was originally created by the British Standard Institute as *BS 5750*. The standard is now maintained by ISO (the International Organization for Standardization) and administered by accreditation and certification bodies [21].

### **5.3.3 ISO 15504**

ISO 15504 is a “framework for the assessment of software processes” developed by the International Organization for Standardization (ISO). It is also known as SPICE (Software Process Improvement and Capability dEtermination).

The software process life cycle is also gaining wide usage. This standard is aimed at setting out a clear model for process comparison. SPICE is used much like CMM and CMMI. It models processes to manage, control, guide and monitor software development. This model is then used to measure what a development organization or project team actually does during software development. This information is analyzed to identify weaknesses and drive improvement. It also identifies strengths that can be continued or integrated into common practice for that organization or team [21].

### **5.3.4 Six Sigma**

Six Sigma is a quality management program to achieve “six sigma” levels of quality. It was pioneered by Motorola in the mid – 1980s and has spread to many other manufacturing companies. It continues to spread to service companies as well. In 2000, Fort Wayne, Indiana became the first city to implement the program in a city government.

It aims to have the total number of failures in quality, or customer satisfaction, occur beyond the sixth sigma of likelihood in a normal distribution of customers. Here sigma stands for a step of one standard deviation; designing processes with tolerances of at least six standard deviations will, on reasonable assumptions, yield fewer than 3.4 defects in one million.

Generally, it is a project management methodology that uses data and statistical analysis to measure and improve a company's operational performance. It works by identifying and eliminating "*defects*" in manufacturing and service-related processes. The maximum permissible defects are 3.4 per million opportunities. However Six Sigma is manufacturing-oriented, not software development-oriented and needs further researches to even apply to software development [21].

Defects here refer to: -

In Software Engineering, the non – conformance of software to its requirements, often but incorrectly, called bug.

### **5.3.5 Agile software development**

In Software Engineering, agile software development or agile methods refers to low-overhead methodologies that accept that software is difficult to control. They minimize risk by ensuring that software engineers focus on smaller units of work.

One way in which agile software development is generally distinguished from “heavier” more process-centric methodologies, for example the *waterfall model*, is by its emphasis on values and principles, rather than on processes.

Agile Software methodologies such as “*Extreme Programming*” and “*Lean Software Development*” are full blown methods that take an incremental or evolutionary approach to software development [21].

#### **Extreme Programming:**

It is a method in or approach to software engineering, formulated by Kent Beck, Ward Cunningham, and Ron Jeffries. Kent Beck wrote the first book on the topic, *Extreme Programming Explained*, published in 1999. It is the most popular of several agile software development methodologies. The generalisation of extreme programming to other types of projects is extreme project management [21].

#### **Lean Software Development:**

Lean Manufacturing is a business initiative to reduce waste in manufactured products. The basic idea is to reduce the cost systematically, throughout the product and production process, by means of a series of engineering reviews. The crucial insight is that most costs are assigned when a product is designed. Often an engineer will specify familiar, safe materials and processes rather than inexpensive, efficient ones [21].

This reduces project risk, that is, the cost to the engineer, while increasing financial risks, and decreasing profits. Good organizations develop and review checklists to review product designs.

## 5.4 Software Lifecycle Models

A common mistake people make concerning software is assuming that the majority of software development is programming. When they think of programming, their minds conjure up the image of a late-night hacker pounding out code on an old computer in the basement of a musty apartment. While this is certainly one approach to programming, it is hardly the norm and definitely not the way the majority of current software is developed. In fact, programming is only a fraction of the software development process. Today, many other steps are involved in the successful development and deployment of computer software. Taken together, all these steps are referred to as the software lifecycle. Often these steps are described by models called *software life cycle models*. In the next two sections we will examine two of these models: the waterfall model and the spiral model. First, however, we need to describe the basic processes that make up the software lifecycle [20 & 21].

Most models of the software lifecycle include the following six processes: *requirements engineering, design, programming, integration, delivery, and maintenance*. The list below gives a description of each process.

### 5.4.1 Requirements Engineering

During this process, developers and clients meet to discuss ideas for the new software product. Developers use a variety of techniques in order to assess the real needs of the client. One such technique is rapid prototyping in which a prototype program is built that can mimic the functionality of the desired software. Using this prototype, clients can better understand how the final product will behave and can determine whether this behaviour is what they really need. Unless the requirements engineering process is done properly, the resulting software will not be useful to the client even though it may run correctly. The requirements engineering process is completed when the specifications for the new software product are written in a formal document called the “*requirements specification document*” [21].

### **5.4.2 Design**

During this process, the developers decide how they will construct the software so that it meets the specifications agreed upon in the *requirements specification document*. Usually the design of the software goes through several stages in which it becomes progressively more detailed.

This approach to design is called stepwise refinement, and it allows the developers to manage the complexity of software by postponing decisions about details as late as possible in order to concentrate on other important design issues. When the design is complete, it is recorded in the “*design specification document*” [21].

### **5.4.3 Programming**

During this process, teams of programmers write the actual code of the software. The software is divided into separate units called modules in order to handle the complexity of the programming process. Not only are these teams responsible for coding their modules, they are also responsible for proper documentation describing their code and for testing the code to insure correctness [21].

### **5.4.4 Integration**

During this process, the individual modules of the software product are combined to form the integrated software product. Since the modules were developed separately, testing is crucial to the integration process. Even with a good design, incompatibilities between modules are likely to exist. These problems need to be identified and corrected to complete the integration [21].

### **5.4.5 Delivery**

During this process, the developers deliver the completed software to the clients. Usually the clients will conduct acceptance testing on the software to determine whether or not it meets the specifications agreed upon in the requirements specification document. Once the software is accepted, it is installed and used by the client [21].

### **5.4.6 Maintenance**

During this process, the software undergoes various changes after delivery in order to fix bugs, add new functionality, port the software to new platforms, or adapt the software to new technologies. Although it may seem that the software should be finished after delivery, this is far from true [21].

All successful software products evolve over time to meet the changing needs of the clients. You may be surprised to discover that of all these processes, maintenance dominates the cost of the lifecycle. The graph below shows the relative cost of the processes that make up the software lifecycle.

Because maintenance costs are so important, many developers are beginning to use design approaches that result in software which is easier to maintain [21 & 22].

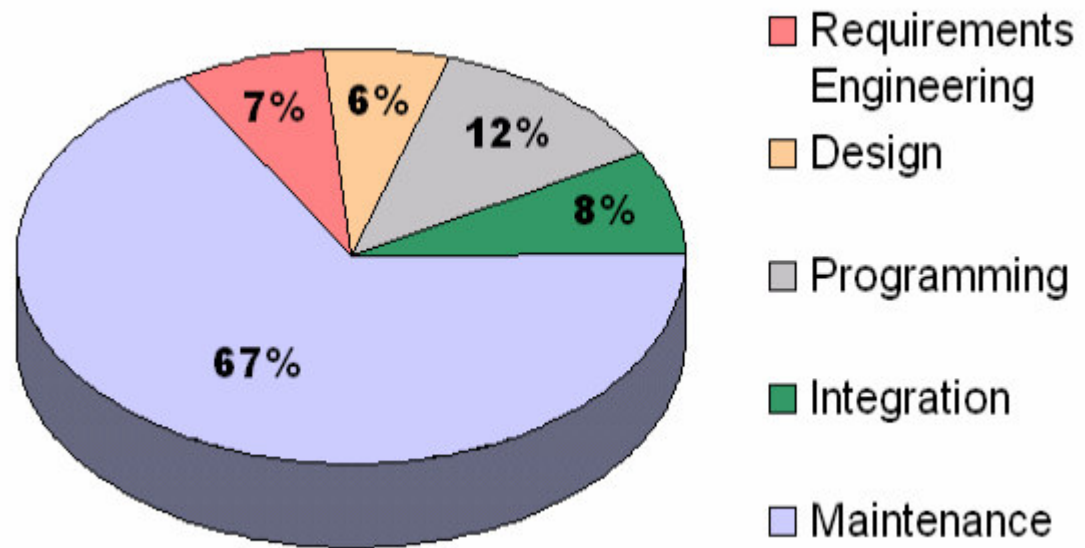


Figure 18 : Approximate relative costs of the phases of the software life cycle

Many different models have been created to represent the software life cycle. Although these models use various names for the processes of the life cycle, they all include the six processes described above in one way or another. In addition, these models usually emphasize some other aspect of software development such as a particular design technique (e.g., rapid prototyping), management technique (e.g., risk management), or the model describes a limited domain of software development (e.g., real-time software) [20].

## 5.5 Software Quality Characteristics

The goal of software engineering is, of course, to design and develop better software. However, what exactly does "better software" mean?

In order to answer this question, this section introduces some common software quality characteristics. Six of the most important quality characteristics are *maintainability*, *correctness*, *reusability*, *reliability*, *portability*, and *efficiency* [22].

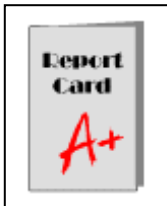
### **Maintainability:**

Maintainability is "the ease with which changes can be made to satisfy new requirements or to correct deficiencies". Well designed software should be flexible enough to accommodate future changes that will be needed as new requirements when it comes to light. Since maintenance accounts for nearly 70% cost of the software life cycle, the importance of this quality characteristic cannot be overemphasized. Quite often the programmer responsible for writing a section of code is not the one who must maintain it. For this reason, the quality of the software documentation significantly affects the maintainability of the software product.



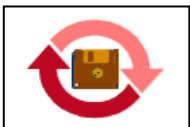
### **Correctness:**

*Correctness* is "the degree with which software adheres to its specified requirements". At the start of the software lifecycle, the requirements for the software are determined and formalized in the requirements specification document. Well designed software should meet all the stated requirements. While it might seem obvious that software should be correct, the reality is that this characteristic is one of the hardest to assess. Because of the tremendous complexity of software products, it is impossible to perform exhaustive execution-based testing to insure that no errors will occur when the software is run. Also, it is important to remember that some products of the software lifecycle such as the design specification cannot be "executed" for testing. Instead, these products must be tested with various other techniques such as formal proofs, inspections, and walkthroughs.



### **Reusability:**

*Reusability* is "the ease with which software can be reused in developing other software". By reusing existing software, developers can create more complex software in a shorter amount of time. Reuse is already a common technique employed in other engineering disciplines.



For example, when a house is constructed, the trusses which support the roof are typically purchased preassembled. Unless a special design is needed, the architect will not bother to design a new truss for the house. Instead, he or she will simply reuse an existing design that has proven it to be reliable. In much the same way, software can be designed to accommodate reuse in many situations. A simple example of software reuse could be the development of an efficient sorting routine that can be incorporated in many future applications.

### **Reliability:**



*Reliability* is "the frequency and criticality of software failure, where failure is an unacceptable effect or behaviour occurring under permissible operating conditions". The frequency of software failure is measured by the average time between failures. The criticality of software failure is measured by the average time required for repair. Ideally, software engineers want their products to fail as little as possible (i.e., demonstrate high correctness) and be as easy as possible to fix (i.e., demonstrate good maintainability). For some real-time systems such as air traffic control or heart monitors, reliability becomes the most important software quality characteristic. However, it would be difficult to imagine a highly reliable system that did not also demonstrate high correctness and good maintainability.

### **Portability:**



*Portability* is "the ease with which software can be used on computer configurations other than its current one". Porting software to other computer configurations is important for several reasons. First, "good software products can have a life of 15 years or more, whereas hardware is frequently changed at least every 4 or 5 years. Thus good software can be implemented, over its lifetime, on three or more different hardware configurations". Second, porting software to a new computer configuration may be less expensive than developing analogous software from scratch. Third, the sales of "shrink-wrapped software" can be increased because a greater market for the software is available.

## Efficiency:



*Efficiency* is "the degree with which software fulfils its purpose without waste of resources". Efficiency is really a multifaceted quality characteristic and must be assessed with respect to a particular resource such as execution time or storage space. One measure of efficiency is the speed of a program's execution. Another measure is the amount of storage space the program requires for execution. Often these two measures are inversely related, that is, increasing the execution efficiency causes a decrease in the space efficiency. This relationship is known as the space-time trade-off. When it is not possible to design a software product with efficiency in every aspect, the most important resources of the software are given priority.

The table below summarizes each of the six quality characteristics. With these characteristics, the answer to the question "What is better software?" becomes much more precise. Using these characteristics, software engineers can assess software products for strengths and weaknesses. The two most prominent software development paradigms are the *classical* or *procedural paradigm* and the *object oriented paradigm*.







 Maintainability	The ease with which changes can be made to satisfy new requirements or to correct deficiencies.
 Correctness	The degree with which software adheres to its specified requirements.
 Reusability	The ease with which software can be reused in developing other software.
 Reliability	The frequency and criticality of software failure, where failure is an unacceptable effect or behavior occurring under permissible operating conditions.
 Portability	The ease with which software can be used on computer configurations other than its current one.
 Efficiency	The degree with which software fulfils its purpose without waste of resources.

Table 1: Software Quality Characteristics