UNIT 4 CASE TOOLS

Structure **Page Nos.**

4.0 INTRODUCTION

CASE (Computer Aided Software Engineering) represents a comprehensive philosophy for modeling business, their activities and information system development. The CASE philosophy involves using the computer as a development tool to build models that describe the business, the business environment, and corporate planning, and to document computer system development from planning through implementation.

The complete picture of the CASE philosophy enables specification for corporate plans, system design and system development so that they become fully integrated. This occurs by sharing for the three functions of corporate planning, system analysis and design and system development across CASE component.

4.1 OBJECTIVES

After going through this unit, you should be able to:

- know about software crisis;
- know about the current development tools;
- reasons for CASE failure;
- know about various categories of CASE tools, and
- know about the benefits of CASE tools.

4.2 SOFTWARE CRISIS

The software industry is facing a crisis today because of the following:

- Increase in the size of software packages.
- Increase in complexity of problem areas.

- Project management/coordination problem due to increase in personnel requirements for projects.
- Duplication of effort because most software packages are built manually, i.e., with no automation, no methodology for the most part.
- The increase in cost of software compared to hardware.

Large software projects have become feasible for medium as well as large software organizations due to the falling cost of hardware systems. One of the major areas where the software industry has expanded is the area of specialized applications. This area is usually characterized by a high level of complexity. This has given rise to an increases in the use of formal methodologies for software development. However, these methodologies, for the most part, do not completely solve the problem of complexity.

Medium/large software projects require considerable manpower. This leads to management and coordination problems which in turn give rise to time and cost overruns. Typically in large software development efforts, the engineers working on different parts of the software package do not 'interact' very closely. It could also lead to problems in integration. As software development teams are becoming larger, it has been observed that productivity per person reduces. On the other hand the cost of technically skilled personnel is rising. Most software packages built today have almost no flexibility or scope for extensibility. This contributes to higher maintenance costs. These are of the factors which have contributed to rising software costs as opposed to falling hardware costs.

Advances in hardware technology and the lack of matching pace in software technology is a well known fact. Software continues to increase its share of the EDP budget. Studies, surveys and symposia are being conducted the world over to pinpoint the concerns and issues so that they may be addressed.

- Requirements analysis has been voted as the most troublesome phase of the life cycle in some surveys. The basic problem is one of communication.
- Maintenance is costly, entailing an expenditure amounting to at least 50 per cent of budgets.
- There is a movement towards decentralization of control. End-user computing is being encouraged and being made effective and practical. There is still a long way to go before end users can really make a dent on the back log software applications.
- The applications backlog in the USA has been quoted to be between a staggering three to seven years.
- The concept of building software systems as characterized by the waterfall model is changing. Today, people are talking of growing software systems through rapid prototyping and manufacturing software systems through 'software factories'. The basic problem is one of communication.

In another survey, most of the respondents recognized information technology as a real or potential source of competitive advantage. In India, with an increase in competition in the marketplace in general, information is soon going to be a vital resource. The new system development technologies must address this issue. The top four concerns described by Management Information Systems (MIS) executives were: Facilitating/managing end user computing, Translating information technology to a competitive advantage, having the top management understand needs and perspective of MIS, and measuring and improving MIS/ Data Processing (DP) effectiveness / productivity.

A third survey shows a 15 to 20 per cent average personnel turnover rate in the country. Undocumented and unstructured systems become the nightmare for successors.

Another survey conducted by AT & T many years ago projected that if the demand for telephones continued to grow, very soon, every man, woman and child in USA would have to be a telephone operator. Of course, it did not happen. From surveys, the numbers have been extrapolated and by 2000 AD it is said that every man, woman and child in USA will have to be a programmer.

4.3 WHAT IS WRONG WITH CURRENT DEVELOPMENT METHODS?

We have a wide variety of methods, and the life cycle curve is well-known, so what is going wrong? The computer press is full of articles on The Maintenance Crisis, Skills Shortages and other related topics.

The analysis and programming tools were developed in an environment that was highly centralized. The machine and the data processing professionals were at a central site, often remote from the users. Also, there is a large commitment in terms of costs for the organization which has such a site. (Salaries of the development and support staff, as well as machine costs). Therefore, it should be no surprise that this has led to development methods which do not offer sufficient flexibility, since one of the requirements for management was the increase in control over the projects offered by the tools and methodologies. The adoption of the production environment is different from a more flexible decentralized development environment.

The application backlog appears as a result of development teams spending the time allocated to development on maintenance and upgrading of existing systems. This in turn leads to a maintenance crisis, whereby systems that are coming live were designed in the past, and hence were designed for past requirements. Modifications, sometimes called enhancements, therefore need to be added at a late stage in the development process. Systems were probably designed in such a way that end users were not considered at all. In fact they may have been the last people to be consulted.

Probably, the modern user was brought up to use sophisticated systems, or have seen television programs that show them what is possible. Therefore, the expectations from any systems that is delivered have increased.

To achieve what the user wants need skilled staff. But, there is still a shortage of skilled computer staff.

Another factor which has added to this crisis is the falling cost of hardware. This fall has been dramatic over the years, and it is now feasible for an organization to have a personal computer in every office, if not on every desk, and to have many machines scattered about a site or many sites, rather than a central mainframe. The result is more problems, including lack of standardization of hardware and software, and the decentralization of control as well as development. This leads to a conflict between existing methods for the centralized production of software and the user who wishes to have more control, and a quicker response on a decentralized set of machines. If a typical commercial project is 18 to 24 months in gestation, then due to the applications backlog this lips to around 48 months. Also, there is no need to get surprised if the requirements will alter during this time.
Often the delivered system is unreliable, its accuracy questionable and the functions offered are variants of the one planned in various stages. The end result is that the user is frustrated or loses interest in the project. The cost of not getting it right is quite high, and the place where the errors or omissions occurred is significant.

Despite the advent of structured methods, (which not all development teams use) the major resource cost is the result of errors or omissions in the analysis phase of the project. Hence, the three problems exacerbate each other, compounding the difficulties of an already complex task. The solution lies in the use of automated analysis environments, structured programming methods and closer consultation with the end user.

Let us examine the cause of the software crisis more closely.

4.3.1 Software and its Increasing Cost

Software costs increase because of the following reasons:

- Each application will need the generation of new programs;
- Purchasing of new equipment means that existing software will either have to be modified or rewritten;
- Programming is labour intensive and therefore strongly affected by inflation despite new techniques.

4.3.2 Software Errors and their Impact

In the majority of current systems, the cost of testing and maintenance is around 40 per cent of the total expenditure. Some would go so far as to say it is 60 per cent. In any case it is a significant portion of the software budget. Indeed, in certain cases, data processing departments are so tied up with maintenance of current systems that no development can take place. The increasing costs of error correction can be identified with the following components:

- The increase in software complexity increase the testing complexity;
- Notification and communication of errors become widespread and more costly and there are frequent changes in documentation;
- Repeating tests that have been done before is costly. It should be possible to test the portions of code that are changed but often this is not the case;
- The project team would have been disbanded due to leavers and over commitments;
- Inadequate or inaccurate requirements used throughout the project due to lack of user involvement and consultation in the early stages of development;
- The cost of maintaining a system is often underestimated at the outset of a project.

☞ Check Your Progress 1

1) MIS stands for _____.

2) The maximum expenditure on a software project will be for _____.

4.4 AN ENGINEERING APPROACH TO SOFTWARE

Computer Science is one of the most evolving technologies of the twentieth century. CASE is among the most talked about these techniques and terms.
Science provides the fundamental laws, technology uses these fundamental laws, and, engineering uses technology. When a technology is used for an engineering product, primarily, the additional considerations lead to cost, standardization and marketability.

It is interesting to note that software is also evolving from a science to an engineering discipline. The science of accessing information, computing, organizing information and applying logic to it developed into an implementable technology with the advent

of programming languages. In the course of time, programming languages graduated from low-level Assembly languages to high-level languages like FORTRAN, PASCAL, COBOL and C. This helped software technology to gather momentum and proliferate. With proliferation of this technology and the increase in its complexity, the need for mass production, cost reduction, and standardization was felt – which is why, today, we are talking about Software Engineering.

Software Engineering, or affordable production of complex software is becoming possible today, because we have technology to build tools for Software Development. These CASE tools are based on techniques and methods that span the entire **Software Development Life Cycle (SDLC)**, and are implementable on a computer. To get a good understanding of CASE methodology or CASE tools, it is important to appreciate the techniques and methods which have made CASE possible.

The important techniques and tools available at each stage of the SDLC are shown in *Figure 4.1*.

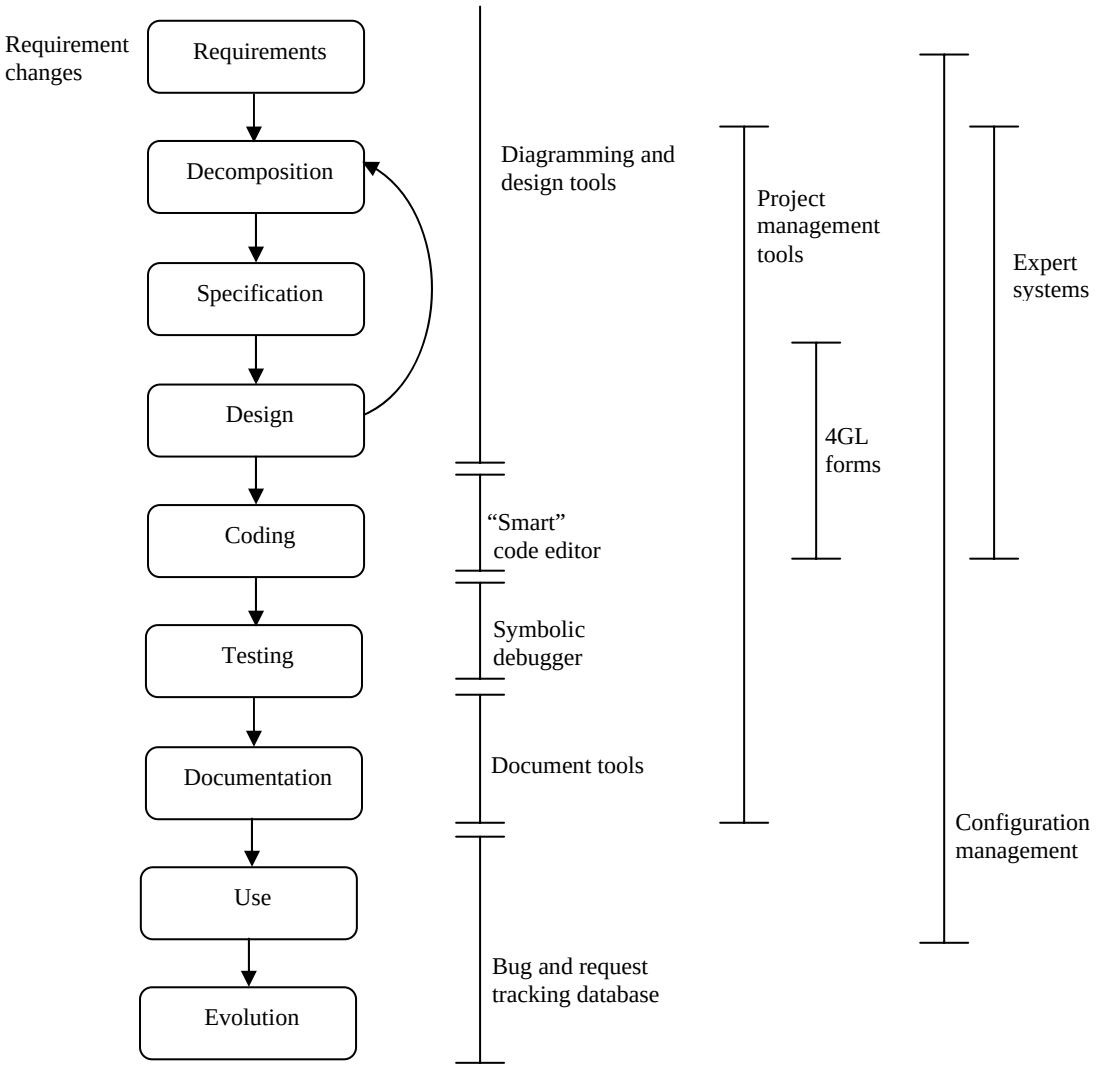

Figure 4.1: Typical Phases of the Software Development Life Cycle and Associated Development Tools

a) **Feasibility Study:** This is the stage when the need for automation is felt, but its cost-benefit is not clear. Simple tools like spreadsheets can be used for carrying out a cost-benefit analysis. These tools may or may not be included in a CASE tools kit.

b) **Requirements Study:** Once a green signal is obtained for automation, the first exercise to be undertaken is a requirements study. This study decides the automation boundary and specifies the requirements of the user in detail. The techniques used for requirements study are varied, but process modeling techniques like structured systems analysis and design (SSAD) and data

modelling techniques like E-R modeling are popular. In both these techniques, a detailed data dictionary is prepared CASE tools provide a means to document these techniques, and facilitate completeness and correctness.

c) **Hardware Sizing Capacity Planning:** The information collected during requirements study can be processed by tools to arrive at hardware sizing. Since the tool needs to operate on the requirements study information, it is necessary for this tool to be integrated with the CASE tools for requirements study. Another advantage of integrating size with study is prediction of capacity changes resulting from changes or enhancements to requirements.

d) **Software Development Estimates:** The effort and duration for software design and development can be estimated using the information gathered for requirements study. Formal techniques like function point analysis (FPA) and construction cost model (COCOMO) are available for estimation. These techniques should ideally be integrated into a CASE tool.

e) **Design:** The requirements study of a system identifies its inputs (screens), output (reports), inquiries, data dictionary and processing logic. During the design phase, these get translated to programs, libraries and files or databases. This translation can easily be aided by a CASE tool.

f) **Software Development:** Once the design is complete, the code needs to be produced. Several tools are available for this. Fourth generation languages and program generators can be used to produce code. These often interface, or form an integral part of a CASE tool.

g) **Testing and quality Assurance:** Testing software systematically, with every release, is essential. Tools for testing must allow easy preparation of test data, and automatic testing of the total system (on-line and batch) with error reporting and automatic recovery from error. Other tools like the Test Coverage Analyser (TCA) are also useful.

h) **Implementation:** Implementation of developed software involves loading or distributing developed software to user sites and, training user personnel. Tools are available for loading and distributing software. Tools are also available for building tutorials and other forms of training material.

i) **Maintenance:** Released and operational software requires maintenance, for error correction and enhancements. Traditionally it has been found that the highest costs are associated with this phase, and fortunately for us, it is this stage that the CASE tool addresses best. The documentation facilitated by proper usage of CASE tools form analysis though implementation becomes vital for maintenance. Besides, the information repository makes tools for Impact Analysis, Version Control and Amendment log possible.

j) **Project Management:** Project Management spans the entire life cycle of a project. A tool for effective Project Management is essential for the success of a large project. The Project Management tool could very well be integrated in a CASE Tool kit, because much of the information required for Project Management is available at various stages of SDLC.

Some tools that can be used for Project Management are:

1) **Network and Bar chart Drawing:** A tool for drawing bar charts and similar charts can be effectively used for planning time, cost and resources. Such a tool facilitates updating changes to plans, and maintaining records of variance between plan and actuals.

2) **Skills Inventory:** A skills inventory system can be very useful for selection of suitable manpower for a project.

3) **Project Costing:** We have seen earlier how estimation can be integrated into CASE. With these estimates available, and with manpower allocation using a skills inventory system, manpower costing can be easily automated. Costing of other resources can be obtained from the network drawing tool. Any records maintained here can be very useful for sizing and estimating future projects.

4) **Software Metrics:** Tools for maintenance of software metrics can record errors detected at each stage of testing for future risk prediction.

5) **Quality Assurance Calendar:** A tool could help set up a quality assurance calendar (this could interface with the network drawing tool). This tool could also maintain suggestions and recommendations from Quality Assurance review. CASE tools that address even a subset of the requirements are also very useful. However, the ultimate CASE tools for quickly developed, easily maintainable, low-cost, standardized quality software should address all aspects of SDLC.

☞ **Check Your Progress 2**

1) SDLC stands for _____.

2) Released and operational software requires maintenance, for _____ and _____.

4.5 WHY CASE FAILS?

Being aware of the most cited causes of CASE failure will, obviously, increase the chance of success. Among the most often cited reasons are the following:

- Poor involvement of management
- Unrealistic expectations
- No standards of methodologies already in place
- Lack of integration with current practices.

Some of the other causes that are cited are the following:

- Misuse of CASE tools
- Too much emphasis on tools as total solution
- Ignoring the importance of management support and interaction
- Poor documentation (of tools)
- Not enough functionality
- Looking at CASE as a risk element.

There is a small and simple set of do's that are recommended. They are:

- Start slowly and do not invest very heavily on CASE tools.
- Begin with a few relatively small pilot projects.
- Get senior level management's blessings.

- Give the movement credibility by 'selling' the ideas within the company.
- Be sure to spend on training.
- Be patient.
- Make an analysis of needs and priorities and identity the tools for them.
- Start a metrics program. This will help quantify the benefits of CASE which would be critical when making more requests from the management.

4.6 CASE TOOLS

There are two generations of CASE tools. The first generation of CASE tools can be broadly classified into three groups: information generations or 4GLs, front-end design/analysis tools.

The variety of 4GL products include the following: report generators, query languages, DBMS front ends and modeling languages. Most suffer from shortcomings: they are tied too closely to a proprietary database system thereby offering a very restricted solution; they are functionally too weak to be more than a building block in a larger application solution or they are not easily integrated with existing production system and data.

Design tools help a user to draw blue prints or design diagrams based on some predetermined methodology. Typically, high-level design documentation is provided automatically. The obvious flow in these tools is that they are standalone and their results are not easily integrated into the subsequent phases of the life cycle.

A major shortcoming of these first generation CASE products is their inability to bridge the gap between design and application generation. The second generation CASE tools evolved into two major categories: life cycle automation and solution software. The first category of tools is aimed at data processing profession to provide general solutions to their problems. The second category of tools is aimed at analyst or application specialist, in a restricted domain of application, to provide fast solution to the end user. Electronic spread sheets represent such tools in the very restricted domain of financial analysis.

4.6.1 Categories of CASE Tools

CASE tools might be classified into four broad categories according to the CASE problems on which they focus.

1) **Front-end CASE Tools, or Upper CASE Tools**

   These deal with the high-level design, specification and analysis of software and requirements. These tools include computer-aided diagramming tools oriented toward a particular programming design methodology, more recently including object-oriented design.

2) **Back-end CASE Tools, or Lower CASE Tools**

   These deal with the detailed design, coding, assembly, and testing of software. These tools may aid the programmer directly.

3) **Maintenance Tools**

These deals with software after initial release. These tools may assist in tracking bag fixes and enhancement requests, porting to new platforms or performing new releases.

4) **Support Software and Frameworks**

These provide basic functionality required in tools of type 1, 2 and 3. Support software includes basic operating-system functionality as well as higher-level support such as project management and scheduling software, and database support to track different versions of software releases. Various projects have been directed towards standardizing frameworks to support and integrate CASE applications.

In addition to CASE methodology based on traditional programming languages and tools, there are two quite different approaches to CASE, particularly for back-end tools:

1) *Higher-level languages and packages*

Some commercial products have focused on specific applications. For example, there are dozens of fourth-generation languages (4GLs), forms packages, and database design tools oriented towards the large market for business database applications on character terminals. Some more recent products are targeted at simplifying development of user interfaces in windows systems. It is possible to make large gains in application programming productivity by focusing on a single application area.

2) *Expert Systems*

The application of artificial intelligence to programming, to select designs and produce code automatically in limited domains is another approach. So far, this approach has seen limited application.

4.6.2 Selecting Case Tools

This brings us to the task of identifying, evaluating and selecting CASE tools. There are large number of CASE tools available in the market. Also, the price range of these CASE tools greatly varies. The incredible variance in the type and functionality of tools which fall under the banner of CASE can thus be gauged. Selecting one from among them is not a trivial risk. If, maximizing performance per unit is the objective then the obvious choice is **Turbo Analyst**. Turbo analyst is a low risk acquisition for organizations not yet convinced of the efficacy of the technology. As already mentioned, there are many other tools to choose from.

Developing a menu of features and facilities that tools offer-a checklist of features/facilities/functionality that one desires for one's environment. Importance must be given to each feature so as to facilitate a computation of scores for each short listed tool. The best is selected subject to the budget constraints. One such methodology has been developed by P-CUBE Corp., USA. We should adapt this to our context and include certain factors like after-sales services, support and training, foreign exchanges restrictions, duties, interfaces to packages/languages common in India, etc.

After all this evaluation, selection and implementation, one notices a drop in productivity. Have patience as CASE manages to improve long-term productivity and quality significantly.

4.6.3 DEFT Case Tools

DEFT supplies Computer Aided Software Engineering (CASE) products to engineers who work with Relational Data Base Management Systems. RDBMS engineers use the DEFT CASE systems to:

- Assist them in gathering the initial requirements from end-user.
- Analyze these requirements and determine their feasibility.
- Design the system's general algorithms.
- Design an actual detailed implementation in terms of the target environment (hardware and operating system, specific RDBMS, etc.)
- Check their designs for completeness and consistency, and for contravention of specific RDBMS naming conventions.
- Automatically generate the RDBMS (tables, indices and forms) from the design.
- Maintain their existing systems by reverse engineering the original databases from their host machines to DEFT.
- Control their development efforts through the medium of our configuration management tools.

The DEFT CASE System

DEFT CASE system consists of both tools and methods. DEFT is flexible – it allows you the luxury of using either your own methods or The DEFT way is easy to use and adopts a simple methodology. In either case, your projects will run on time and within budget, predictably and consistently (it is expected!).

The DEFT way

CASE methods employ the structured approach to software engineering and comprise various methods in which one draws diagrams or models of the computer system to be built. The models each portray a certain aspect of the system, with four view required to adequately model a system that uses an RDBMS as the data repository. These four views are:

- **Data Flow Diagrams (DFDs)** show the path of the data from one process to another and from to the users of the system. The diagram represents the processes to be performed and identifies the data itself.
- **Entity Relationship Diagrams (ERDs)** show the relationships of various data entities to each other. With DEFT's approach, you can model not only your logical analysis, but also the physical database design itself, since DEFT allows you to define key or index structures right in the model.
- **Program Structure Diagrams (PSDs)** describe the logic or business rules involved in the processes. PSDs can be used to depict pseudo code for either 3GL or 4GL programs, and to break a module into subroutines or functions in additions to graphic ally showing the main procedures.
- Form/Report templates and prototypes complete the views required to model the target system. DEFT provides a tool that allows you to rapidly create these forms or report templates. Using DEFT's Gateway products, you can actually create these templates on your host machine within your RDBMS environment.

4.7 FACTORS AFFECTING SOFTWARE DEVELOPMENT

Main factors

- The people who need to develop the product
- The work environment in which they develop the software
- The methodologies and tools that they use
- The need to produce quality software

Other factors

- The need for experimentation and error
- The appointment of the correct person as development controller
- The psychology of the team members
- The need for standardization.

4.8 THE BENEFITS OF USING CASE

The benefits of upper CASE are more direct if you usually perform corporate planning. By using an upper CASE system to build an enterprise model, you gain greater insight into the importance of certain functions and how the activities they control affect the entire organization. You can better understand:

- corporate and departmental mechanisms and responsibilities;
- the goals of the company and its departments;
- the influence of operations on achieving these goals;
- their place within corporate and departmental administration and operations;
- the timeliness and sequence of operations;
- factors influencing operations and achievement of goals;
- allocation of resources in support of operations;
- the effect of external influences of the organization;
- problems facing the organization; and
- the importance of information relative to the success of the organization.

☞ Check Your Progress 3

1) _____ tools help a user to draw blue prints or design diagrams based on some predetermined methodology.

2) _____show the path of the data from one process to another and from and to the users of the system.

4.9 SUMMARY

The purpose of this unit was to provide a broad perspective of the merging CASE field. With software expenditure skyrocketing, CASE has become a competitive edge for both major corporations and nations. Considering the fact that some countries spend around 30 to 40 billion on software, a 50% reduction in cost means billions of saving each year.

Currently, CASE products are classified into upper CASE and lower CASE. Upper CASE tools support the front end of development life cycle. Lower CASE tools support the back end of development life cycle. Many of the tools are designed to support a particular methodology. One approach is to integrate these tools to cover more of the development life cycle. Another is the development of a CASE shell, which is an environment that provides advanced facilities for the user to build his/her own tools.

4.10 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Management Information Systems

2) Maintenance

Check Your Progress 2

1) Software Development Life Cycle

2) error correction, enhancements

Check Your Progress 3

1) Design

2) Data Flow Diagrams

4.11 FURTHER READINGS

1) *Software Engineering*, Sixth Edition, 2001, Ian Sommerville; Pearson Education.

2) *Software Engineering – A Practitioner's Approach*, Roger S. Pressman; McGraw-Hill International Edition.

Reference Websites

- http://www.cs.queensu.ca/FAQs/SE/case.html
- http://www.rspa.com