Information Systems and Software Engineering

Computer Science & Information Technology (CS)

20 Rank under AIR 100

Postal Correspondence

- ✓ Examination Oriented Theory, Practice Set
- ✓ Key concepts, Analysis & Summary

ENGINEERS INSTITUTE OF INDIA
India's Best Institute for IES, GATE & PSUs
A Team of IES & GATE Toppers

CONTENT

Information Systems and Software Engineering: information gathering, requirement and feasibility analysis, data flow diagrams, process specifications, input/output design, process life cycle, planning and managing the project, design, coding, testing, implementation, maintenance.

CHAPTER-1
INTRODUCTION

1.1 INTRODUCTION

Have you ever noticed how the invention of one technology can have profound and unexpected effects on other seemingly unrelated technologies, on commercial enter-prices, on people and even on culture as a whole? This phenomena is often called “the law of unintended consequences”

Today, computer software is the single most important technology on the world stage and it is also prime example of the low of unintended consequences. No one in the 1950s could have predicted that software would become an indispensable technology for business, science and engineering; that software would enable the creation of new technologies (*e.g.*, genetic engineering), the extension of existing technologies (*e.g.*, telecommunications), and demise of other technologies (*e.g.*, the printing industries); that the software would be driving force behind the personal computer revolution; that shrink wrapped software products would be purchased by consumers in the neighborhood malls; that a software company would become larger and more influential than the vast majority of industrial era companies, that a vast s/w driven network called the internet would evolve and change everything from library search to consumer shopping to the dating habits of young adults.

No one could have foreseen that software would become embedded in systems of all kinds: transportation, medical, telecommunications, military, industrial, entertainment, office machines – the list is almost endless. And if we are to believe the low of unintended consequences, there are many effects that we cannot yet predict.

And, finally no one could have forseen that millions of computer programs would have to be corrected, adapted and enhanced as time passed and that the burden of performing these “maintenance” activities would absorb more people and more resources than all work applied to the creation of new software.

1.2 WHAT IS SOFTWARE ENGINEERING?

Software has become critical to advancement in almost all areas of human endeavor. 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

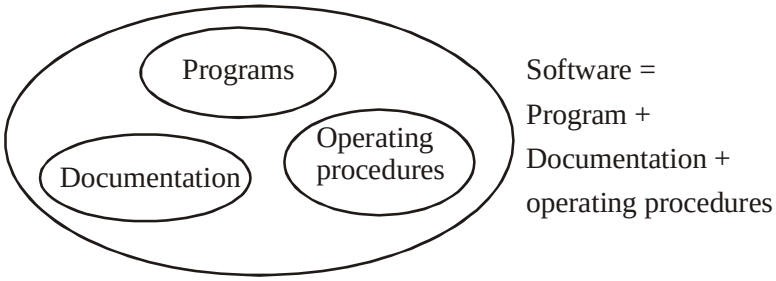
Fritz Bauer defined software engineering as "The establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines".

Stephen Schach defined software engineering as "A discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that 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 procedure used to setup and operate the software system.

The components of software systems are shown in figure 1.1.



Software = Program + Documentation + Operating procedures

Fig. 1.1. Components of software.

→ Program is a combination of source code and object code.

→ Documentation consists of different types of manuals as shown in figure 1.2.

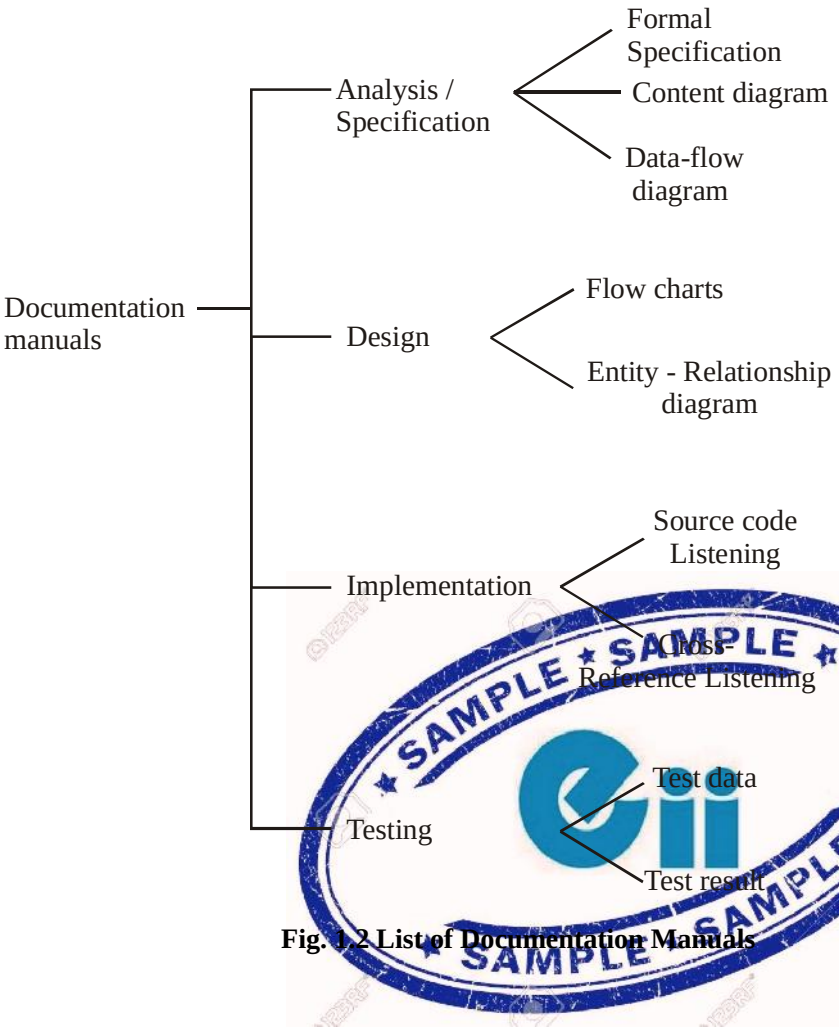


Fig. 1.2 List of Documentation Manuals

→ List of operating procedure manuals/ documents is shown in figure 1.3.

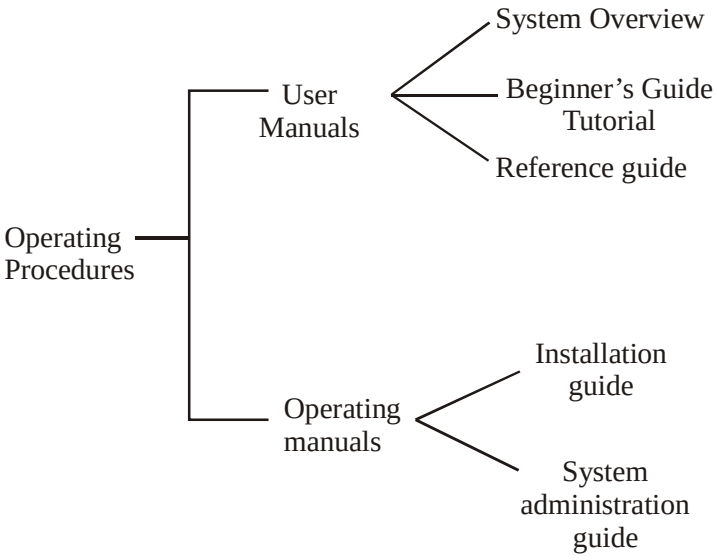


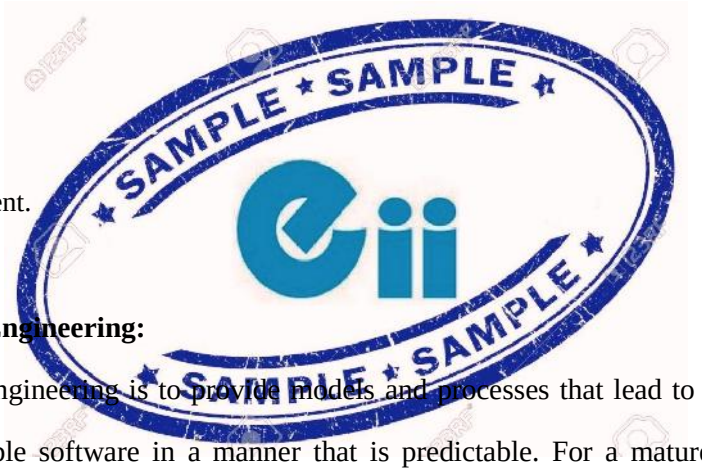
Figure-1.3: List of operating procedure manuals

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 that hiring smart, knowledgeable developers and being 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 following are few reasons why it is difficult to improve software process?

1. Not enough time
2. Lack of knowledge
3. Wrong motivations
4. Insufficient commitment.



1.3 Goal of Software Engineering:

The goal of software engineering is to provide models and processes that lead to the production of well documented maintainable software in a manner that is predictable. For a mature process it should be possible to determine in advance how much time and effort will be required to produce the final product. This can only be done using data from past experience, which requires that we must measure the software.

Software development organizations follow some process when developing a software product. A key component of any software development process is the life cycle model on which the process is based. In the IEEE standard Glossary of software Engineering Terminology, the software life cycle is

"The period of time that starts when a software product is conceived and ends when product is no larger available for use. The software life cycle typically includes a requirement phase, design phase, implementation phase, test phase, installation and check out phase, operation and maintenance phase, and sometimes retirement phase".

A software life cycle model is often called a software development life cycle (SDLC)

1.4 SDLC Models

1.4.1 Build and fix model

Sometimes a product is constructed without specifications or any attempt at design. Instead, the developer simply builds a product that is reworked as many times as necessary to satisfy the client.

→ This is an adhoc approach and not well defined.

→ It is simply two phase model

First phase → Write code

Second phase → Fix it

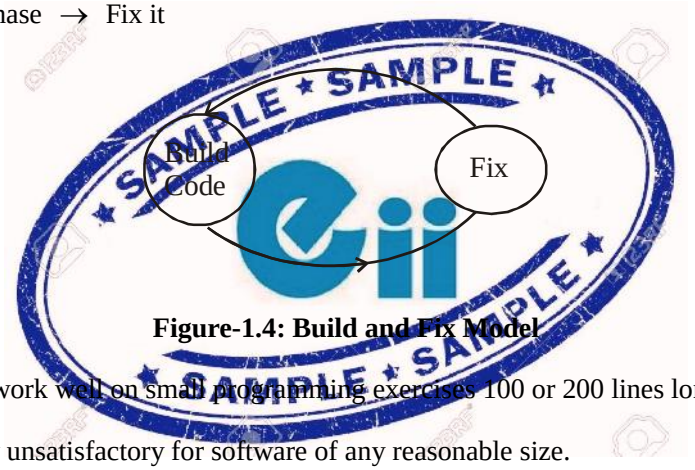


Figure-1.4: Build and Fix Model

→ This approach may work well on small programming exercises 100 or 200 lines long,

→ This model is totally unsatisfactory for software of any reasonable size.

→ The cost of development using this approach is actually very high as compared to the cost of a properly specified and carefully designed product.

→ Maintenance of the product can be extremely difficult without specification or design document.

1.4.2 The classical Water fall Model

This model has five phases

1. Requirement analysis and specification 2. Design

3. Implementation and unit testing 4. Integration and system testing.

5. Operation and maintenance

→ The phases always occur in this order and do no overlap.

→ The developer must complete each phase before the next phase begins.

1. Requirement Analysis and Specification Phase:

The goal of this phase is to understand the exact requirements of the customer and to document them properly. This activity is usually executed together with the customer, as the goal is to document all functions, performance and interfacing requirements for the software. This requirements describe the "what" of a system, not the "how". This phase produce a large document, written in natural language, contains a description of what the system will do without describing how it will be done. The resultant document is known as software requirement specification (SRS) document.

The SRS document may act as contract between the developer and customer.

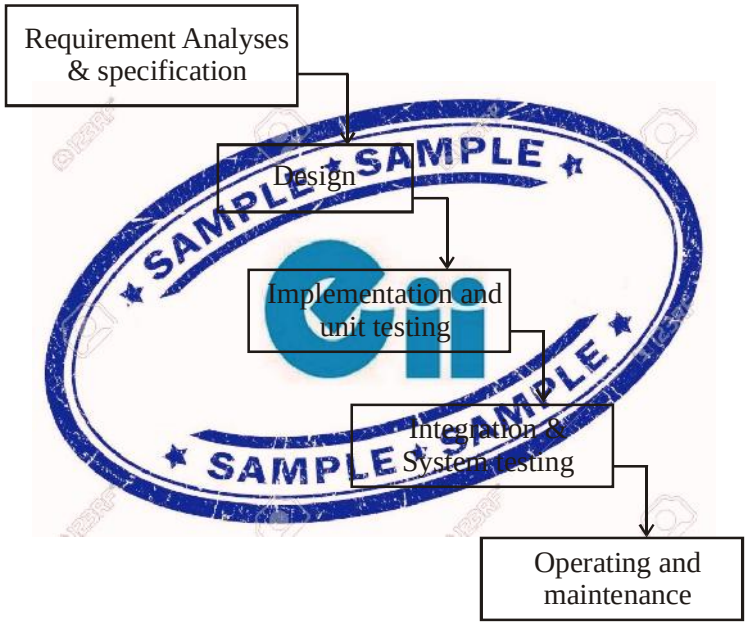


Figure-1.5: Waterfall Model

2. Design Phase

The SRS document is produced in the previous phase, which contains the exact requirements of the customer. The goal of this phase is to transform the requirements specification into a structure that is suitable for implementation in some programming language. Here, overall architecture is defined, and the high level and detailed design work is performed. This work is documented and known as software design description. (SDD) document. The information contained in SDD should be sufficient to begin the code.

3. Implementation and Unit Testing

During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by the software developers is contained in SDD.

During testing, the major activities are centered around the examination and modification of the code. Initially, small modulus are tested in isolation form the rest of the software product. These are problems associated with testing a module in isolation. How do we run a module without anything to call it, to be called by it or, possibly, to output intermediate values obtained during execution? Such problems are solved in this phase and modulus are tested after writing some overhead code.

4. Integration and System Testing Phase:

This is very important phase. Effective testing will contribute to the delivery of higher quality software products, more satisfied users, lower maintenance costs and more accurate and reliable results. It is very expensive activity and consumes one third to one half of the cost of a typical development project.

As we know, the purpose of unit testing is to determine that each independent module is correctly implemented. This gives little chance to determine that each independent module is correctly implemented. This gives little chance to determine that the interface between modulus is also correct, and for this reason integration testing is performed. System testing involves the testing of the entire system, where as software is a part of the system. This is essential to build confidence in the developers before software is delivered to the customer or released in the market.

5. Operation and Maintenance Phase

Software maintenance is a task that every development group has to face, when the software is delivered to the customer's site, installed and is operational. Therefore, release of software inaugurates the operation and maintenance phase of the life cycle.

The time spend and effort required to keep the software operational after release is very significant. Despite the fact that it is a very important and challenging task; it is routinely the poorly managed headache that nobody wants to face.

Software maintenance is very broad activity that includes error correction, enhancement of capabilities, deletion of absolute capabilities, and optimization. The purpose of this phase is to preserve the value of the software over time. This phase may span for 5 to 50 years where as development may be 1 to 30 years.

Problems of Waterfall Model

1. It is difficult to define all the requirements at the beginning of a project.
2. This model is not suitable for accommodating any change.
3. A working version of the system is not seen until late in the projects life.
4. It does not scale up well to large projects.
5. Real projects are rarely sequential.

1.4.3 Iterative Waterfall Model:

The classical waterfall model is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases. However, in practical development environments, the engineers do commit a large number of errors in almost every phase of life cycle. The source of the defects can be many oversight, wrong assumptions, use of inappropriate technology, communication gap among the project engineers etc. These defects usually get detected much later in the life cycle. For example a design defect might go unnoticed till we reach the coding or testing phase. Once the defect is detected, the engineers need to go back to the phase where the defect had occurred and redu some of the work done during that phase and the subsequent phases to correct the defect and its effect on the later phases. Therefore, in any practical software development work, it is not possible to strictly follow the classical waterfall model.

Feedback paths are needed in the classical waterfall model from every phase to its preceding phase as shown in figure 1.6 to allow for the correction of the errors committed during a phase that are detected in later phases.

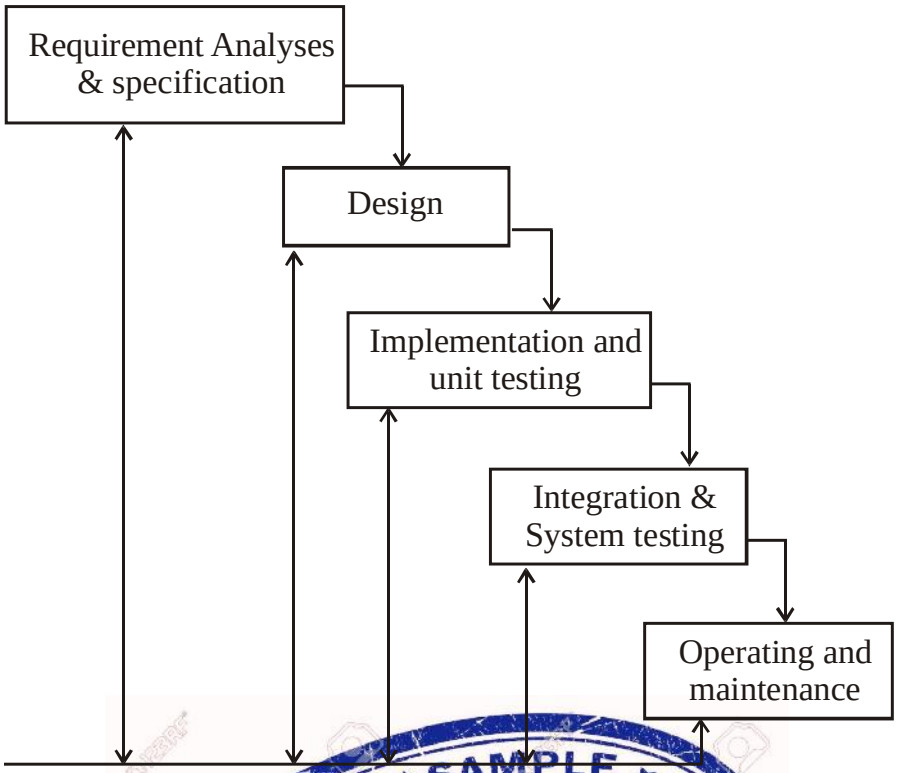


Figure 1.6: Iterative waterfall model