Module 1

Introduction to Software Engineering

Lesson 2

Structured Programming

Version 2 CSE IIT, Kharagpur

Specific Instructional Objectives

At the end of this lesson the student will be able to:

- Identify the important features of a structured program.
- Identify the important advantages of structured programming over unstructured ones.
- Explain how software design techniques have evolved over the last 50 years.
- Differentiate between exploratory style and modern style of software development.

Important features of a structured program.

A structured program uses three types of program constructs i.e. selection, sequence and iteration. Structured programs avoid unstructured control flows by restricting the use of GOTO statements. A structured program consists of a well partitioned set of modules. Structured programming uses single entry, single-exit program constructs such as if-then-else, do-while, etc. Thus, the structured programming principle emphasizes designing neat control structures for programs.

Important advantages of structured programming.

Structured programs are easier to read and understand. Structured programs are easier to maintain. They require less effort and time for development. They are amenable to easier debugging and usually fewer errors are made in the course of writing such programs.

Evolution of software design techniques over the last 50 years.

During the 1950s, most programs were being written in assembly language. These programs were limited to about a few hundreds of lines of assembly code, i.e. were very small in size. Every programmer developed programs in his own individual style - based on his intuition. This type of programming was called Exploratory Programming.

The next significant development which occurred during early 1960s in the area computer programming was the high-level language programming. Use of high-level language programming reduced development efforts and development time significantly. Languages like FORTRAN, ALGOL, and COBOL were introduced at that time.

As the size and complexity of programs kept on increasing, the exploratory programming style proved to be insufficient. Programmers found it increasingly difficult not only to write cost-effective and correct programs, but also to understand and maintain programs written by others. To cope with this problem, experienced programmers advised other programmers to pay particular attention to the design of the program's control flow structure (in late 1960s). In the late 1960s, it was found that the "GOTO" statement was the main culprit which makes control structure of a program complicated and messy. At that time most of the programmers used assembly languages extensively. They considered use of "GOTO" statements in high-level languages were very natural because of their familiarity with JUMP statements which are very frequently used in assembly language programming. So they did not really accept that they can write programs without using GOTO statements, and considered the frequent use of GOTO statements inevitable. At this time, **Dijkstra [1968]** published his (now famous) article "GOTO Statements Considered Harmful". Expectedly, many programmers were enraged to read this article. They published several counter articles highlighting the advantages and inevitably of GOTO statements. But, soon it was conclusively proved that only three programming constructs – sequence, selection, and iteration – were sufficient to express any programming logic. This formed the basis of the structured programming methodology.

After structured programming, the next important development was data structure-oriented design. Programmers argued that for writing a good program, it is important to pay more attention to the design of data structure, of the program rather than to the design of its control structure. Data structure-oriented design techniques actually help to derive program structure from the data structure of the program. Example of a very popular data structure-oriented design technique is Jackson's Structured Programming (JSP) methodology, developed by Michael Jackson in the1970s.

Next significant development in the late 1970s was the development of data flow-oriented design technique. Experienced programmers stated that to have a good program structure, one has to study how the data flows from input to the output of the program. Every program reads data and then processes that data to produce some output. Once the data flow structure is identified, then from there one can derive the program structure.

Object-oriented design (1980s) is the latest and very widely used technique. It has an intuitively appealing design approach in which natural objects (such as employees, pay-roll register, etc.) occurring in a problem are first identified. Relationships among objects (such as composition, reference and inheritance) are determined. Each object essentially acts as a data hiding entity.

Exploratory style vs. modern style of software development.

An important difference is that the exploratory software development style is based on error correction while the software engineering principles are primarily based on error prevention. Inherent in the software engineering principles is the realization that it is much more cost-effective to prevent errors from occurring than to correct them as and when they are detected. Even when errors occur, software engineering principles emphasize detection of errors as close to the point where the errors are committed as possible. In the exploratory style, errors are detected only during the final product testing. In contrast, the modern practice of software development is to develop the software through several well-defined stages such as requirements specification, design, coding, testing, etc., and attempts are made to detect and fix as many errors as possible in the same phase in which they occur.

In the exploratory style, coding was considered synonymous with software development. For instance, exploratory programming style believed in developing a working system as quickly as possible and then successively modifying it until it performed satisfactorily.

In the modern software development style, coding is regarded as only a small part of the overall software development activities. There are several development activities such as design and testing which typically require much more effort than coding.

A lot of attention is being paid to requirements specification. Significant effort is now being devoted to develop a clear specification of the problem before any development activity is started.

Now there is a distinct design phase where standard design techniques are employed.

Periodic reviews are being carried out during all stages of the development process. The main objective of carrying out reviews is phase containment of errors, i.e. detect and correct errors as soon as possible. Defects are usually not detected as soon as they occur, rather they are noticed much later in the life cycle. Once a defect is detected, we have to go back to the phase where it was introduced and rework those phases - possibly change the design or change the code and so on.

Today, software testing has become very systematic and standard testing techniques are available. Testing activity has also become all encompassing in the sense that test cases are being developed right from the requirements specification stage.

There is better visibility of design and code. By visibility we mean production of good quality, consistent and standard documents during every phase. In the past, very little attention was paid to producing good quality and consistent documents. In the exploratory style, the design and test activities, even if carried out (in whatever way), were not documented satisfactorily. Today, consciously good quality documents are being developed during product development. This has made fault diagnosis and maintenance smoother.

Now, projects are first thoroughly planned. Project planning normally includes preparation of various types of estimates, resource scheduling, and development of project tracking plans. Several techniques and tools for tasks such as configuration management, cost estimation, scheduling, etc. are used for effective software project management.

Several metrics are being used to help in software project management and software quality assurance.

The following questions have been designed to test the objectives identified for this module:

1. Identify the problem one would face, if he tries to develop a large software product without using software engineering principles.

Ans.: - Without using software engineering principles it would be difficult to develop large programs. In industry it is usually needed to develop large programs to accommodate multiple functions at various levels. The problem is that the complexity and the difficulty levels of the programs increase exponentially with their sizes as shown in fig. 1.3.

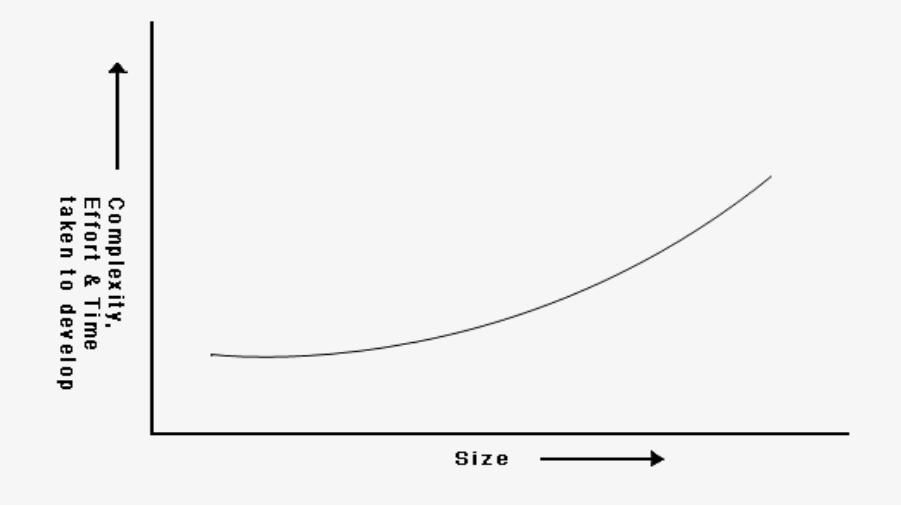

Fig. 1.3: Increase in development time and effort with problem size

For example, a program of size 1,000 lines of code has some complexity. But a program with 10,000 LOC is not 10 times more difficult to develop, but may be 100 times more difficult unless software engineering principles are used. Software engineering helps to reduce the programming complexity.

2. Identify the two important techniques that software engineering uses to tackle the problem of exponential growth of problem complexity with its size.

Ans.: - Software engineering principles use two important techniques to reduce problem complexity: abstraction and decomposition.

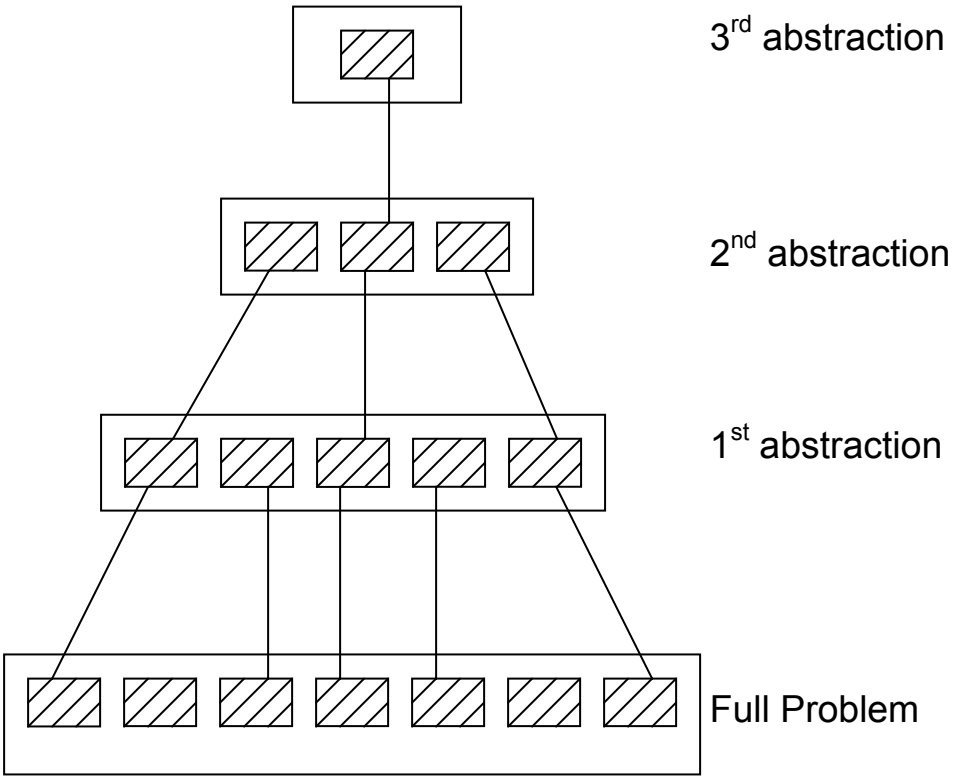

Fig. 1.4: A hierarchy of abstraction

The principle of abstraction (in fig.1.4) implies that a problem can be simplified by omitting irrelevant details. Once simpler problem is solved then the omitted details can be taken into consideration to solve the next lower level abstraction. In this technique any random decomposition of a problem into smaller parts will not help. The problem has to be decomposed such that each component of the decomposed problem can be solved in solution and then the solution of the different components can be combined to get the full solution.

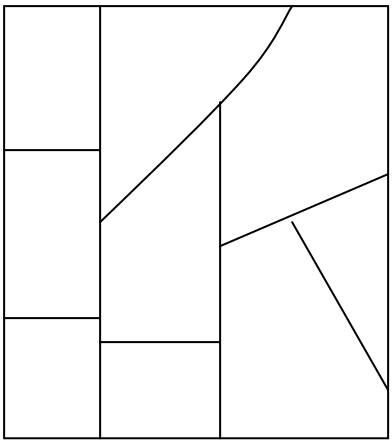

Fig. 1.5: Decomposition of a large problem into a set of smaller problems.

In other words, a good decomposition as shown in fig.1.5 should minimize interactions among various components.

3. State five symptoms of the present software crisis.

Ans.: - Software engineering appears to be among the few options available to tackle the present software crisis. To explain the present software crisis in simple words, it is considered the following that are being faced. The expenses that organizations all around the world are incurring on software purchases compared to those on hardware purchases have been showing a worrying trend over the years (as shown in fig.1.6).

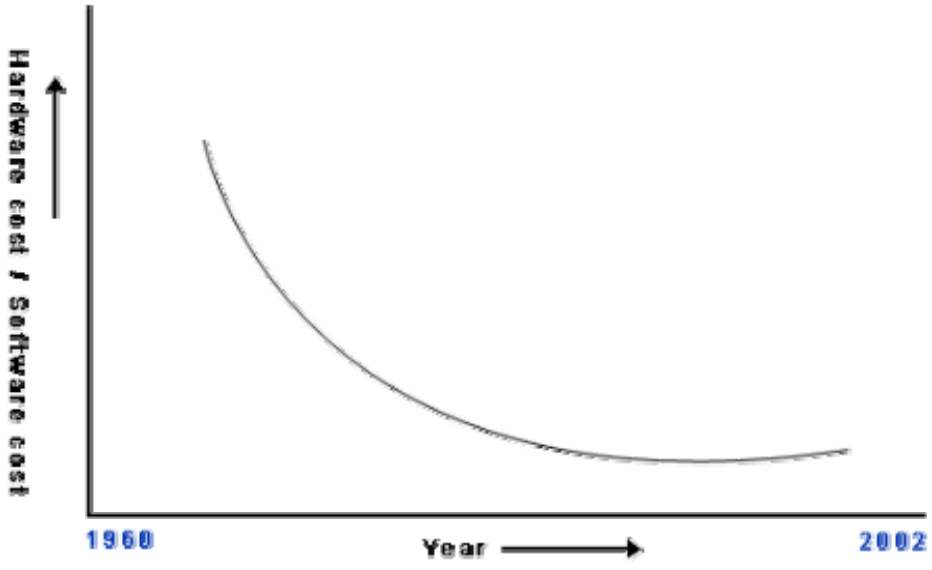

Fig. 1.6: Change in the relative cost of hardware and software over time

Organizations are spending larger and larger portions of their budget on software. Not only are the software products turning out to be more expensive than hardware, but they also present a host of other problems to the customers: software products are difficult to alter, debug, and enhance; use resources non-optimally; often fail to meet the user requirements; are far from being reliable; frequently crash; and are often delivered late. Among these, the trend of increasing software costs is probably the most important symptom of the present software crisis.

4. State four factors that have contributed to the making of the present software crisis.

Ans.: - There are many factors that have contributed to the making of the present software crisis. Those factors are larger problem sizes, lack of adequate training in software engineering, increasing skill shortage, and low productivity improvements.

5. Suggest at least two possible solutions to the present software crisis.

Ans.: - It is believed that the only satisfactory solution to the present software crisis can possibly come from a spread of software engineering practices among the engineers, coupled with further advancements in the software engineering discipline itself.

6. Identify at least four basic characteristics that differentiate a simple program from a software product.

Ans.: - Programs are developed by individuals for their personal use. They are therefore, small in size and have limited functionality but software products are extremely large. In case of a program, the programmer himself is the sole user but on the other hand, in case of a software product, a large number of users who are not involved with the development are attached. In case of a program, a single developer is involved but in case of a software product, a large number of developers are involved. For a program, user interface may not be so important because programmer is the sole user. On the other hand, for a software product, user interface must be very important because developers of that product and users of that product are totally different. In case of a program, very little documentation is expected but a software product must be well documented. A program can be developed according to the programmer's individual style of development but a software product must be developed using software engineering principles.

7. Identify two important features of that a program must satisfy to be called as a structured program.

Ans.: - First, a structured program uses three type of program constructs i.e. selection, sequence and iteration. Structured programs avoid unstructured control flows by restricting the use of GOTO statements. Secondly, structured program consists of a well partitioned set of modules. Structured programming uses single entry, single-exit program constructs such as if-then-else, do-while, etc. Thus, the structured programming principle emphasizes designing neat control structures for programs.

8. State three important advantages of structured programming.

Ans.: - Structured programs are easier to read and understand. Structured programs are easier to maintain. They require less effort and time for development. They are amenable to easier debugging and usually fewer errors are made in the course of writing such programs.

9. Explain exploratory program development style.

Ans.: - The exploratory software development style is based on error correction while the software engineering principles are primarily based on error prevention. Inherent in the software engineering principles is the realization that it is much more cost-effective to prevent errors from occurring than to correct them as and when they are detected. Even when errors occur, software engineering principles emphasize detection of errors as close to the point where the errors are committed as possible. In the exploratory style, errors are detected only during the final product testing.

In the exploratory style, coding was considered synonymous with software development. For instance, the naïve way of developing a software product (which is called the exploratory programming style) believed in developing a working system as quickly as possible and then successively modifying it until it performed satisfactorily.

10. Show at least three important drawbacks of the exploratory programming style.

Ans.: - As the size and complexity of programs kept on increasing, the exploratory programming style proved to be insufficient. The exploratory programming style proved to be insufficient because -

- People wanted more sophisticated things to be done by software and as a result the size and complexity of programs increased. Exploratory style proved to be insufficient for developing large and complex programs.

- Programmers found that it was very difficult to write cost effective and correct programs using the exploratory style.

- Programmers also found that it was very difficult to understand and maintain the programs which were written by others.

11. Identify at least two advantages of using high-level languages over assembly languages.

Ans.: - Assembly language programs are limited to about a few hundreds of lines of assembly code, i.e. are very small in size. Every programmer develops programs in his own individual style - based on intuition. This type of programming is called Exploratory Programming.
But use of high-level programming language reduces development efforts and development time significantly. Languages like FORTRAN, ALGOL, and COBOL are the examples of high-level programming languages.

12. State at least two basic differences between control flow-oriented and data flow-oriented design techniques.

Ans.: - Control flow-oriented design deals with carefully designing the program's control structure. A program's control structure refers to the sequence, in which the program's instructions are executed, i.e. the control flow of the program. But data flow-oriented design technique identifies:

- Different processing stations (functions) in a system

- The data items that flows between processing stations

13. State at least five advantages of object-oriented design techniques.

Ans.: - Object-oriented techniques have gained wide acceptance because of it's:

- Simplicity (due to abstraction)

- Code and design reuse

- Improved productivity

- Better understandability

- Better problem decomposition
- Easy maintenance

14. State at least three differences between the exploratory style and modern styles of software development.

Ans.: - An important difference is that the exploratory software development style is based on error correction while the software engineering principles are primarily based on error prevention. Inherent in the software engineering principles is the realization that it is much more cost-effective to prevent errors from occurring than to correct them as and when they are detected. Even when errors occur, software engineering principles emphasize detection of errors as close to the point where the errors are committed as possible. In the exploratory style, errors are detected only during the final product testing. In contrast, the modern practice of software development is to develop the software through several well-defined stages such as requirements specification, design, coding, testing, etc., and attempts are made to detect and fix as many errors as possible in the same phase in which they occur.

In the exploratory style, coding was considered synonymous with software development. For instance, the naïve way of developing a software product (which is called the exploratory programming style) believed in developing a working system as quickly as possible and then successively modifying it until it performed satisfactorily.

In the modern software development style, coding is regarded as only a small part of the overall software development activities. There are several development activities such as design and testing which typically require much more effort than coding.

A lot of attention is being paid to requirements specification. Significant effort is devoted to develop a clear specification of the problem before any development activity is started.

Now there is a distinct design phase where standard design techniques are employed.

Periodic reviews are being carried out during all stages of the development process. The main objective of carrying out reviews is phase containment of errors, i.e. detect and correct errors as soon as possible. Defects are usually not detected immediately after when they occur, rather they are noticed much later in the life cycle. Once a defect is detected we have to go back to the phase where it was introduced and rework those phases - possibly change the design or change the code and so on.

Today, software testing has become very systematic and standard testing techniques are available. Testing activity has also become all encompassing in

the sense that test cases are being developed right from the requirements specification stage.

There is better visibility of design and code. By visibility we mean production of good quality, consistent and standard documents during every phase. In the past, very little attention was paid to producing good quality and consistent documents. In the exploratory style, the design and test activities, even if carried out (in whatever way), were not documented satisfactorily. Today, consciously good quality documents are being developed during product development. This has made fault diagnosis and maintenance far more smoother.

Now, projects are first thoroughly planned. Project planning normally includes preparation of various types of estimates, resource scheduling, and development of project tracking plans. Several techniques and tools for tasks such as configuration management, cost estimation, scheduling, etc. are used for effective software project management.

Several metrics are used to help in software project management and software quality assurance.

Mark the following as either True or False. Justify your answer.

1. **All software engineering principles are backed by either scientific basis or theoretical proof.**

Ans.: - False.

Explanation: - Many software engineering principles are just thumb rules and lack any scientific basis or theoretical proof.

2. **There are well defined steps through which a problem is solved using an exploratory style.**

Ans.: - False.

Explanation: - The exploratory software development style is based on error correction while the software engineering principles are primarily based on error prevention. Inherent in the software engineering principles is the realization that it is much more cost-effective to prevent errors from occurring than to correct them as and when they are detected. Even when errors occur, software engineering principles emphasize detection of errors as close to the point where the errors are committed as possible. In

the exploratory style, errors are detected only during the final product testing.

For the following, mark all options which are true.

1. Which of the following problems can be considered to be contributing to the present software crisis?

□ large problem size √
□ lack of rapid progress of software engineering √
□ lack of intelligent engineers
□ shortage of skilled manpower √

2. Which of the following are essential program constructs (i.e. it would not be possible to develop programs for any given problem without using the construct)?

□ sequence √
□ selection √
□ jump
□ iteration √