Unit 5 Software Development, Design and Testing

Structure:

5.1 Introduction

Computer software has become a driving force. It is the engine that drives business decision making. It serves as the basis for modern scientific investigation and engineering problem solving. It is a key factor that differentiates modern products and services. It is embedded in systems of all kinds: transportation, medical, telecommunications, military, industrial processes, entertainment, office products etc. Software is virtually inescapable in the modern world. And as we move into the twenty-first century, it will become the driver for new advances in everything from elementary education to genetic engineering.

Computer software is the product that software engineers design and build. It encompasses programs that execute within a computer of any size and architecture, documents that encompass hard-copy and virtual forms and data that combine numbers and text but also include representations of pictorial, video and audio information.

Software is a set of application programs that are built by software engineers and are used by virtually everyone in the industrialized world either directly or indirectly. Software is important because it affects nearly every aspect of our lives and has become pervasive in our commerce, our culture and our everyday activities.

Software is generally built like you build any other successful product, by applying a process that leads to a high quality result that meets the needs of the people who will use the product. We apply a software engineering approach to develop this product.

From the point of view of a software engineer, the work product is the programs, documents and data that are computer software. But from the user's point of view, the work product is the resultant information that somehow makes the user's world better.

Objectives:

- explain software development methodology and process
- explain the analysis and design of software
- describe various steps involved in software testing
- list software paradigms
- discuss on various programming methods
- list some software applications

5.2 Software Development

Software is a logical rather than a physical system element. Therefore software has characteristics that are considerably different than those of hardware:

a) Software is developed or engineered; it is not manufactured in the classical sense.
b) Software doesn't "wear out".
c) Although the industry is moving toward component-based assembly, most software continues to be custom built.

Software Development Methodology is a series of processes that, if followed, can lead to the development of an application.

Niklaus Wirth, the inventor of Pascal says:

Algorithms + Data structures = Programs

‘A software system is a set of mechanisms for performing certain action on certain data’. Traditional development techniques focus on the functions of the system.

The software development process: System development can be viewed as a process. Furthermore, the development itself, in essence, is a process of change, refinement, transformation, or addition to the existing product.

The process can be divided into small, interacting phases – sub processes. Each sub process must have the following:

- A description in terms of how it works
- Specification of the input required for the process
- Specification of the output to be produced

Generally, the software development process can be viewed as a series of transformations, where the output of one transformation becomes the input of the subsequent transformation:

- **Transformation 1 (analysis)** – translates the users’ needs into system requirements and responsibilities.
- **Transformation 2 (design)** – begins with a problem statement and ends with a detailed design that can be transformed into an operational system.
- **Transformation 3 (implementation)** – refines the detailed design into the system deployment that will satisfy users’ needs.

An example of the software development process is the waterfall approach, which starts with deciding what is to be done (what is the problem). Once the requirements have been determined, we must next decide how to accomplish them. This is followed by a step in which we do it, whatever “it” has required us to do. We then must test the result to see if we have satisfied the users’ requirements. Finally, we use what we have done.

In the real world, the problems are not always well-defined and that is why the waterfall model has limited utility.

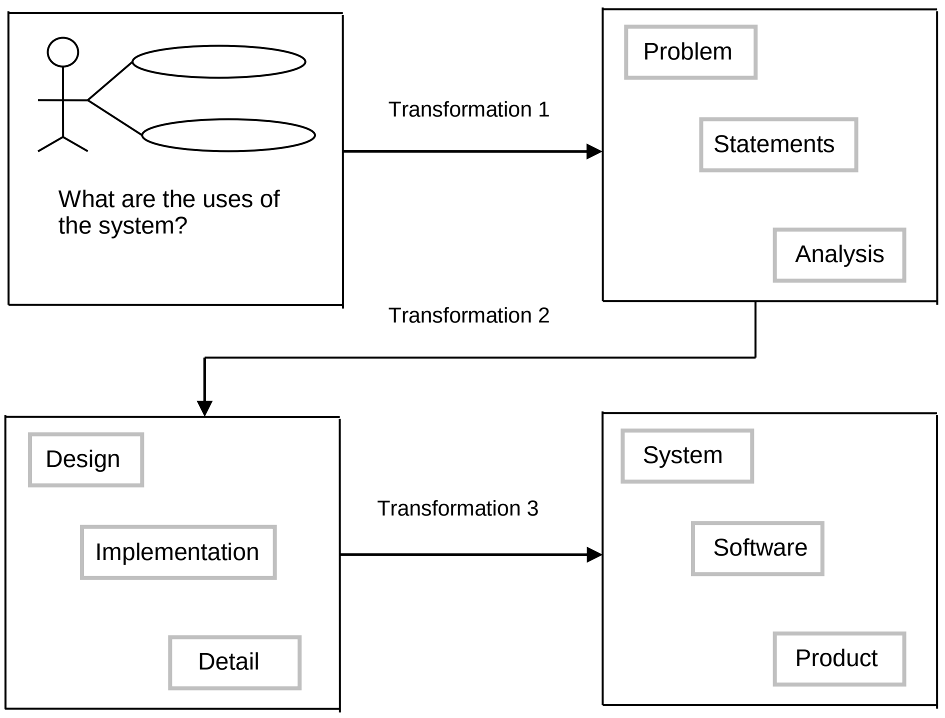

Fig. 5.1: Software process reflecting transformation from needs to a software product that satisfies those needs

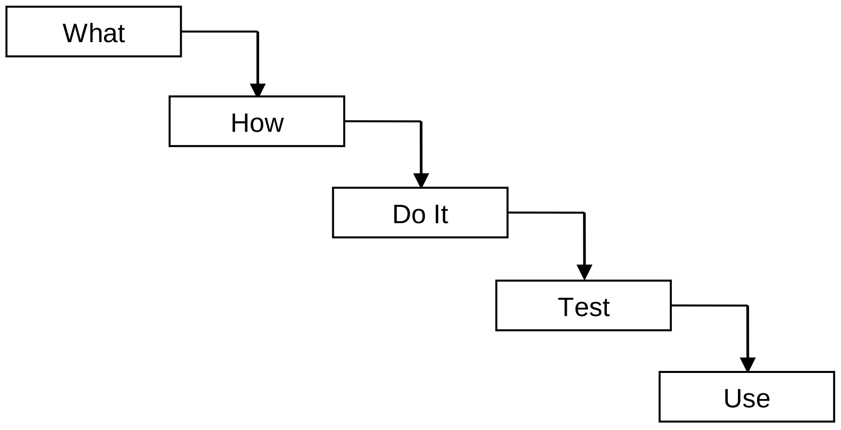

Fig. 5.2: The waterfall software development process

Building quality software: To achieve quality software we need to be able to answer the following questions:

- How do we determine when the system is ready for delivery?
- Is it now an operational system that satisfies users' needs?

- Is it correct and operating as we thought it should?
- Does it pass an evaluation process?

Four quality measures are:

- **Correspondence** – measures how well the delivered system matches the needs of the operational environment, as described in the original requirements statement.
- **Validation** – task of predicting correspondence.
- **Correctness** – measures the consistency of the product requirements with respect to the design specification.
- **Verification** – exercise of determining correctness.

Verification: Am I building the product right?

Validation: Am I building the right product?

Validation begins as soon as the project starts, but verification can begin only after a specification has been accepted.

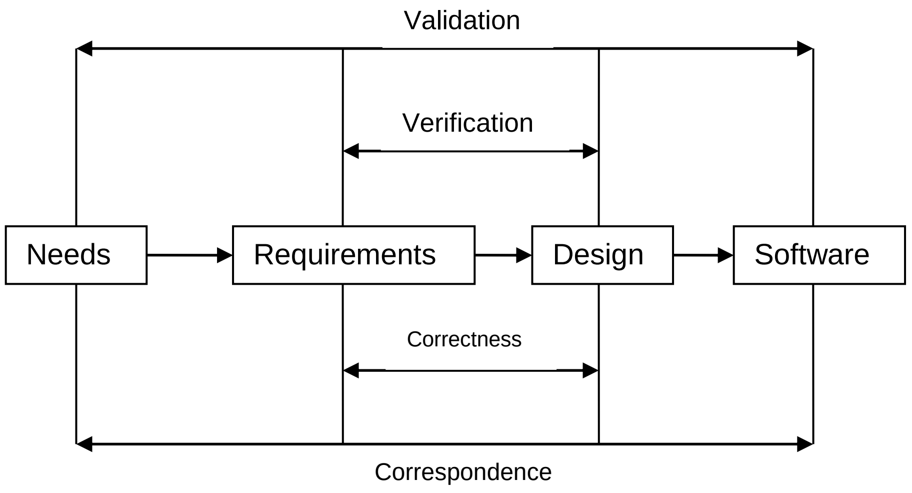

Fig. 5.3: Four quality measures for software evaluation.

Self Assessment Questions

1. A process can be divided into small, interacting phases called ________.
2. ________ measures the consistency of the product requirements with respect to the design specification.

3. ________ measures how well the delivered system matches the needs of the operational environment, as described in the original requirements statement.
4. ________ is a series of processes that, if followed, can lead to the development of an application.
5. Algorithms + Data structures = ________________.

5.3 Analysis and Design

This section describes about software analysis and design. A more methodical approach to software design is proposed by structured methods, these structured methods are sets of notations and guidelines for software design. Budgen (1993) describes some of the most commonly used methods such as structured design and structured systems analysis, he also described the Jackson System Development and various approaches to object-oriented design.

The use of structured methods involves producing large amounts of diagrammatic design documentation. CASE tools have been developed to support particular methods. Structured methods have been applied successfully in many large projects. They can deliver significant cost reductions because they use standard notations and ensure that standard design documentation is produced.

A mathematical method (such as the method for long division) is a strategy that will always lead to the same result irrespective of who applies the method. The term structured methods suggests that the designers should normally generate similar designs from the same specification. A structured method includes a set of activities, notations, report formats, rules and design guidelines. So structured methods often support some of the following models of a system:

1. **A data-flow model** where the system is modeled using the data transformations, which take place as it, is processed.
2. **An entity-relation model**, which is used to describe the logical data, structures being used.
3. **A structural model** where the system components and their interactions are documented.

4. If the method is object-oriented it will include an inheritance model of the system, a model of how objects are composed of other objects and, usually, an object-use model which shows how objects are used by other objects.

Particular method supplement these with other system models such as state transition diagrams, entity life histories that show how each entity is transformed as it is processed and so on. Most methods suggest a centralized repository for system information or a data dictionary should be used.

Design description: A software design is a model system that has many participating entities and relationships. This design is used in a number of different ways. It acts as a basis for detailed implementation; it serves as a communication medium between the designers of sub-systems; it provides information to system maintainers about the original intentions of the system designers, and so on.

Designs are documented in a set of design documents that describes the design for programmers and other designers. There are three main types of notation used in design documents:

1. **Graphical notations:** These are used to display the relationships between the components making up the design and to relate the design to the real-world system is modeling. A graphical view of a design is an abstract view. It is most useful for giving an overall picture of the system.
2. **Program description languages** these languages (PDLs) use control and structuring constructs based on programming language constructs but also allow explanatory text and (sometimes) additional types of statement to be used. These allow the intention of the designer to be expressed rather than the details of how the design is to be implemented.
3. **Informal text** much of the information that is associated with a design cannot be expressed formally. Information about design rationale or non-functional considerations may be expressed using natural language text.

All of these different notations may be used in describing a system design.

Design strategies: The most commonly used software design strategy involved decomposing the design into functional components with system state information held in a shared data area. Since from late 1980s that this alternative, object oriented design has been widely adopted.

Two design strategies are summarized as follows:

1. **Functional design:** The system is designed from a functional viewpoint, starting with a high-level view and progressively refining this into a more detailed design. The System State is centralized and shared between the functions operating on that state. Methods such as Jackson Structured Programming (extension of the Jackson Structured Programming (JSP) method. JSP, developed by Michael Jackson) and the Warnier-Orr method (The technique is based on only a few simple principles of design that are very easy to learn and to apply) are techniques of functional decomposition where the structure of the data is used to determine the functional structure used to process that data.
2. **Object-oriented design:** The system is viewed as a collection of objects rather than as functions. Object-oriented design is based on the idea of information hiding and has been described by Meyer, Booch, and Jacobsen. And many others. JSD is a design method that falls somewhere between function-oriented and object-oriented design.

In an object-oriented design, the System State is decentralized and each object manages its own state information. Objects have a set of attributes defining their state and operations, which act on these attributes. Objects are usually members of an object class whose definition defines attributes and operations of class members. These may be inherited from one or more super-classes so that a class definition need only set out the differences between that class and its super-classes. Objects communicate by exchanging messages; an object calling a procedure associated with another object achieves most object communication.

There is no ‘best’ design strategy, which is suitable for all projects and all types of application. Functional and object-oriented approaches are complementary rather than opposing techniques. Software engineers select the most appropriate approach for each stage in the design process. In fact, large software systems are complex entities that different approaches might be used in the design of different parts of the system.

An object-oriented approach to software design seems to be natural at the highest and lowest levels of system design. Using different approaches to design may require the designer to convert his or her design from one model to another. Many designers are not trained in multiple approaches so prefer to use either **object-oriented or functional design.**

Design quality

A good design might be a design that allows competent code to be produced; it might be a minimal design where the implementation is as compact as possible; or it might be the most maintainable design.

A sustainable design can be adapted to modify existing functions and add new functionally. The design must therefore be understandable and changes should be local in effect. The design components should be cohesive which means that all parts of the component should have a close logical relationship. They should be loosely coupled which means that they should not be tightly integrated. Coupling is a measure of the independence of components. The looser the coupling, the easier it is to adapt the design as the effects of change are localized.

Quality characteristics are equally applicable to object-oriented and function-oriented design. Because of the nature of object-oriented design, which encourages the development of independent components, it is usually easier to achieve maintainable designs as information is concealed within objects.

Object Oriented Design Characteristics:

Object–oriented design is a design strategy based on information hiding. It differs from the functional approach to design in that it views a software system as a set of interacting objects, with their private state, rather than as a set of functions that share a global state.

The characteristics of an object-oriented design (OOD) are:

1. Objects are abstraction of system entities, which are responsible for managing their own private state and offering services to other objects.
2. Objects are independent entities that may readily be changed because state and representation information is held within the objects. Changes to the representation may be made without reference to other system objects.

3. System functionality is expressed in terms of operations or services associated with each object.
4. Shared data areas are eliminated. Objects communicate by calling on services offered by other objects rather than sharing variables. This reduces overall system coupling. There is no possibility of unexpected modifications to shared information.
5. Objects may be distributed and may execute either sequentially or in parallel. Decisions on parallelism need not be taken at an early stage of the design process.

Object-oriented systems are easier to maintain as the objects are independent. They may be understood and modified as stand-alone entities. Changing the implementation of an object or adding services should not affect other system objects. There is a clear mapping between real-world entities and their controlling objects in the system. This improves the understandability and hence maintainability of the design.

Self Assessment Questions

6. In which model system is modeled using the data transformations, which take place as it, is processed?
7. Which model is used to describe the logical data, structures being used?
8. A software design is a model system that has many participating entities and ___________
9. ___________ is easier to maintain as the objects are Independent.

5.4 Coding

Once the design is complete, most of the major decisions about the system have been made. The goal of the coding phase is to translate the design of the system into code in a given programming language. For a given design, the aim of this phase is to implement the design in the best possible manner. The coding phase affects both testing and maintenance profoundly. A well written code reduces the testing and maintenance effort. Since the testing and maintenance cost of software are much higher than the coding cost, the goal of coding should be to reduce the testing and maintenance effort. Hence, during coding the focus should be on developing programs that are easy to write. Simplicity and clarity should be strived for, during the coding phase.

5.5 Software Testing

Software Testing is the process of executing a program or system with the intent of finding errors. Testing presents a stimulating variance for the software engineer. During earlier software engineering activities, the engineer attempts to build software from an abstract concept to a tangible product. Now comes testing. The engineer creates a series of test cases that are intended to 'demolish' the software that has been built. In fact, Testing is the one step in the software process that could be viewed (psychologically, at least) as destructive rather than constructive.

Software engineers are by their nature constructive people. Testing requires the developer discard preconceived notions of the 'correctness' of software just developed and overcome a conflict of interest that occurs when errors are uncovered. Beizer describes this situation effectively when he states:

There's a myth that if we were really good at programming, there would be no bugs to catch.

Testing Objectives

In an excellent book on software testing, Glen Myers states a number of rules that can serve well as testing objectives:

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has a high probability of finding an as-yet-undiscovered error.
3. A successful test is one that uncovers an as-yet-undiscovered error.

These objectives imply a dramatic change in viewpoint. They move counter to the commonly held view that a successful test is one in which no errors are found. Our objective is to design tests that systematically uncover different classes of errors and to do so with a minimum amount of time and effort.

If testing is conducted successfully (according to the objectives stated previously), it will uncover errors in the software. As a secondary benefit, testing demonstrates that software functions appear to be working according to specification, that behavioral and performance requirements appear to have been met. In addition, data collected as testing is conducted provide a good indication of software reliability and some indication of software quality as a whole. But testing cannot show the absence of errors and defects, it can only that software errors and defects are present. It is important to keep this (rather gloomy) statement in mind as testing is being conducted.

Software Testing Strategy

Software testing strategy is explained in figure 5.4. This strategy defines the role of software and leads to software requirements analysis, where the information domain, function, behavior, performance, constraints, and validation criteria for software- are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software, we spiral inward along streamlines that decrease the level of abstraction on each turn.

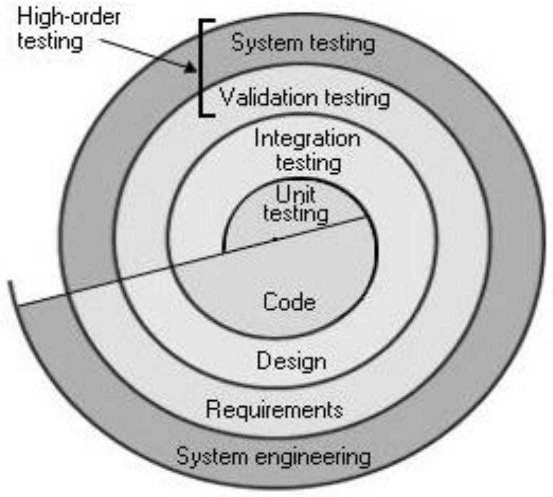

Fig. 5.4: Software Testing Strategy

A strategy for software testing may also be viewed in the context of the spiral shown in figure 5.4. Unit testing begins at the vortex of the spiral and concentrates on each unit (i.e., component) of the software as implemented in source code. Testing progresses outwards along the spiral to integration testing, where the focus is on design and the construction of the software architecture. Taking another turn outward on the spiral, we encounter validation testing, where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally, we arrive at system testing, where the software and other system elements are tested as a whole. To test computer software, we spiral out along stream-lines that broaden the scope of testing with each turn.

Considering the process from a procedural point of view, testing within the context of software engineering is actually a series of four steps that are implemented sequentially. The steps are shown in Figure 5.4. Initially, tests focus on each component individually, ensuring that it functions properly as a unit. Hence, the name is unit testing. Unit testing makes heavy use of white-box testing techniques, exercising specific paths in a module's control structure to ensure complete coverage and maximum error detection. Next, components must be assembled or integrated to form the complete software package. Integration testing addresses the issues associated with the dual problems of verification and program construction. Black-box test case design techniques are the most prevalent during integration, although a limited amount of white-box testing may be used to ensure coverage of major control paths. After the software has been integrated (constructed), a set of high-order tests are conducted. Validation criteria (established during requirements analysis) must be tested. Validation testing provides final assurance that software meets all functional, behavioral, and performance requirements. Black-box testing techniques are used exclusively during validation.

Self Assessment Questions

10. _________ begins at the vortex of the spiral and concentrates on each unit (i.e., component) of the software as implemented in source code.
11. _____________ addresses the issues associated with the dual problems of verification and program construction.
12. ___________ provides final assurance that software meets all functional, behavioral, and performance requirements.

5.6 Software Paradigms

The meaning of Paradigm

Paradigm (a Greek word meaning example) is commonly used to refer to a category of entitles that share a common characteristic. Numerous authors have examined the concept of a paradigm. Perhaps the foremost user of the word has been Thomas Kuhn (1996), who wrote the seminal book “the structure of Scientific Revolution’. Kuhn used the notion of paradigm in the scientific of his assumptions and theories, which affects that view. In his definition, he included the concepts of law, theory, application, and instrumentation; in effects, both the theoretical and the practical.

Kuhn saw a “paradigms” as emerging as the result of social processes in which people develop new ideas and create principles and practices that embody these ideas. Large software development is a social process, because it is often a team effort. Each new software application is a unique creation in its own right. Rarely, if ever, does team of programmers set out to create a program that exactly mimics the code and structure of some other program. Kuhn’s definition has been applied beyond his original application to the history of physical science. Other believe that paradigms are the basis of normal sciences; indeed, the basics of established scientific tradition. In this view, the formation of a paradigm is a sign of maturity for a given science. The notion of a paradigm for programming was first expressed by Floyd (1979) as far as I have been able to discern.

We are going to apply the notion of paradigm to the investigation of programming language and software architecture to determine how well we can solve different types of problems. The question we would like to answer takes the forms: Is there taxonomy within a particular domain that serves to organize the element of that domain? In programming languages, most computer scientists would answer “yes.” In software architecture, the answer is more likely a definite ‘maybe’.

The paradigms are not exclusive, but reflect the different emphasis of language designers. Most practical languages embody features of more than one paradigm.

5.6.1 Classification of Paradigm

1. Imperative Paradigms

It is constructed on commands that update variables in storage. The Latin word *imperium* means “to command”. The language provides statements, such as *assignment statements*, which explicitly change the *state* of the memory of the computer. This model closely matches the actual executions of computer and usually has high execution efficiency. Many people also find the imperative paradigm to be a more natural way of expressing themselves.

2. Functional programming Paradigms

In this paradigm we express computations as the evaluation of mathematical functions. Functional programming paradigms treat values as single entities. Unlike variables, values are never modified. Instead, values are transformed into new values. Computations of functional languages are performed largely through applying functions.

3. Logic programming Paradigms

In this paradigm we express computation in exclusively in terms of ***mathematical logic***. While the functional paradigm emphasizes the idea of a mathematical function, the logic paradigm focuses on predicate logic, in which the basic concept is a *relation*. Logic languages are useful for expressing problems where it is not obvious what the functions should be. For example consider the *uncle* relationship: a given person can have many *uncles*, and another person can be *uncle* to many *nieces* and *nephews*.

4. The Object-Oriented Paradigm

OO programming paradigm is not just a few new features added to a programming language, but it a new way of thinking about the process of decomposing problems and developing programming solutions. Alan Kay characterized the fundamental of OOP as follows: Everything is modeled as object. Computation is performed by message passing: objects communicate with one another via message passing. Every object is an instance of a class where a class represents a grouping of similar objects. Inheritance: defines the relationships between classes. The Object Oriented paradigm focuses on the *objects* that a program is representing, and on allowing them to exhibit "behavior". Unlike imperative paradigm, where data are passive and procedures are active, in the O-O paradigm data is combined with procedures to give *objects*, which are thereby rendered active. Table 5.1 shows the classification of paradigms explained above.

Table 5.1: Classification

Imperative/ Algorithmic	Declarative		Object-Oriented
	Functional Programming	Logic Programming	
Algol Cobol PL/1 Ada C Modula-3	Lisp Haskell ML Miranda APL	Prolog	Smalltalk Simula C++ Java

5.7 Programming Methods

Computer programming (often shortened to programming or coding) is the process of designing, writing, testing, debugging / troubleshooting, and maintaining the source code of computer programs. This source code is written in a programming language. The purpose of programming is to create a program that exhibits a certain desired behavior. The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic.

5.7.1 Structured programming

It is a subset of procedural programming that enforces a logical structure on the program being written to make it more efficient and easier to understand and modify. Certain languages such as Ada, Pascal, and dBASE are designed with features that encourage or enforce a logical program structure.

Structured programming frequently employs a top-down design model, in which developers map out the overall program structure into separate subsections. A defined function or set of similar functions is coded in a separate module or sub module, which means that code, can be loaded into memory more efficiently and that modules can be reused in other programs. After a module has been tested individually, it is then integrated with other modules into the overall program structure.

Program flow follows a simple hierarchical model that employs looping constructs such as "for," "repeat," and "while." Use of the "Go To" statement is discouraged.

Structured programming was first suggested by Corrado Bohm and Guiseppe Jacopini. The two mathematicians demonstrated that any computer program can be written with just three structures: decisions, sequences, and loops. Edsger Dijkstra's subsequent article, Go to Statement Considered Harmful was instrumental in the trend towards structured programming. The most common methodology employed was developed by Dijkstra. In this model (which is often considered to be synonymous with structured programming, although other models exist) the developer separates programs into subsections that each has only one point of access and one point of exit.

Virtually any language can use structured programming techniques to avoid common pitfalls of unstructured languages. Unstructured programming must rely upon the discipline of the developer to avoid structural problems, and as a consequence may result in poorly organized programs. Most modern procedural languages include features that encourage structured programming. Object-oriented programming can be thought of as a type of structured programming, uses structured programming techniques for program flow, and adds more structure for data to the model.

5.7.2 Object-oriented programming (OOP)

It is a programming language model organized around "objects" rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

The programming challenge was seen as how to write the logic, not how to define the data. Object-oriented programming takes the view that what we really care about are the objects we want to manipulate rather than the logic required to manipulate them. Examples of objects range from human beings (described by name, address, and so forth) to buildings and floors (whose properties can be described and managed) down to the little widgets on your computer desktop (such as buttons and scroll bars).

The first step in OOP is to identify all the objects you want to manipulate and how they relate to each other, an exercise often known as data modeling. Once you've identified an object, you generalize it as a class of objects (think of Plato's concept of the "ideal" chair that stands for all chairs) and define the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. A real instance of a class is called (no surprise here) an "object" or, in some environments, an "instance of a class." The object or class instance is what you run in the computer. Its methods provide computer instructions and the class object characteristics provide relevant data. You communicate with objects - and they communicate with each other - with well-defined interfaces called messages.

The concepts and rules used in object-oriented programming provide these important benefits:

- The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics. Called inheritance, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.
- Since a class defines only the data it needs to be concerned with, when an instance of that class (an object) is run, the code will not be able to accidentally access other program data. This characteristic of data hiding provides greater system security and avoids unintended data corruption.
- The definition of a class is reusable not only by the program for which it is initially created but also by other object-oriented programs (and, for this reason, can be more easily distributed for use in networks).
- The concept of data classes allows a programmer to create any new data type that is not already defined in the language itself.

Simula was the first object-oriented programming language. Java, Python, C++, Visual Basic .NET and Ruby are the most popular OOP languages today. The Java programming language is designed especially for use in distributed applications on corporate networks and the Internet. Ruby is used in many Web applications. Curl, Smalltalk, Delphi and Eiffel are also examples of object-oriented programming languages.

OOPSLA is the annual conference for Object-Oriented Programming Systems, Languages and Applications.

Self Assessment Questions

13. Name few Structured Programming Languages?
14. What is the first step in object oriented Programming?
15. In Logic programming paradigms we express computation exclusively in terms of _________________

5.8 Software Applications

Software may be applied in any situation for which a pre specified set of procedural steps has been defined. Information content and determinacy are important factors in determining the nature of a software application. Content refers to the meaning and form of incoming and outgoing information. Software that controls an automated machine accepts discrete data items with limited structure and produces individual machine commands in rapid succession.

Information determinacy refers to the predictability of the order and timing of information. An engineering analysis program accepts data that have a predefined order, executes the analysis algorithm without interruption and produces resultant data in report or graphical format. Such applications are determinate.

A multi-user operating system, on the other hand, accepts inputs that have varied content and arbitrary timing, executes algorithms that can be interrupted by external conditions, and produces output that varies as a function of environment and time. Applications with these characteristics are indeterminate.

Software applications can be neatly compartmentalized into different categories.

System software: System software is a collection of programs written to service other programs. Some system software processes complex information structures. Other systems applications process largely indeterminate data. It is characterized by heavy interaction with hardware, heavy usage by multiple users, concurrent operation that requires scheduling, resource sharing, and sophisticated process management, complex data structures and multiple external interfaces.

Real time software: Software that monitors / analyzes / controls real-world events as they occur is called real time.

Business Software: Business information processing is the largest single software application area. Discrete systems like payroll, accounts receivable/payable have evolved into management information systems (MIS) software that accesses one or more large databases containing business information. Applications in this area restructure existing data in a way that facilitates business operations or management decision making.

Engineering and scientific software: Engineering and scientific software has been characterized by "number crunching" algorithms. Applications range from astronomy to volcano logy, from automotive stress analysis to space shuttle orbital dynamics and from molecular biology to automated manufacturing.

Embedded software: Embedded software resides only in read-only memory and is used to control products and systems for the consumer and industrial markets. Embedded software can provide very limited and esoteric functions or provide significant function and control capability.

Personal computer software: Day to day useful applications like word processing, spreadsheets, multimedia, database management, personal and business financial applications are some of the common examples for personal computer software.

Web-based software: The web pages retrieved by a browser are software that incorporates executable instructions and data. In essence, the network becomes a massive computer providing an almost unlimited software resource that can be accessed by anyone with a modem.

Artificial Intelligence software: Artificial Intelligence software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Expert systems, also called knowledge based systems, pattern recognition, game playing are representative examples of applications within this category.

Software crisis: The set of problems that are encountered in the development of computer software is not limited to software that does not function properly rather the affliction encompasses problems associated with how we develop software, how we support a growing volume of existing software, and how we can expect to keep pace with a growing demand for more software.

Self Assessment Questions

16. Software that controls real-world events as they occur is called _______________.
17. ____________ software is a collection of programs written to service other programs.

5.9 Summary

In this unit you learnt about the various concepts of Communication.

Let’s recap the important points covered in the unit:

- Software is a logical rather than a physical system element.
- Software Development Methodology is a series of processes that, if followed, can lead to the development of an application.
- ‘A software system is a set of mechanisms for performing certain action on certain data’.
- A software design is a model system that has many participating entities and relationships.
- This design is used in a number of different ways. It acts as a basis for detailed implementation; it serves as a communication medium between the designers of sub-systems; it provides information to system maintainers about the original intentions of the system designers, and so on.
- Testing presents an interesting anomaly for the software engineer. During earlier software engineering activities, the engineer attempts to build software from an abstract concept to a tangible product.
- The engineer creates a series of test cases that are intended to 'demolish' the software that has been built. In fact, Testing is the one step in the software process that could be viewed (psychologically, at least) as destructive rather than constructive.
- System engineering defines the role of software and leads to software requirements analysis, where the information domain, function,

behavior, performance, constraints, and validation criteria for software-are established.

- Structured programming (sometimes known as modular programming) is a subset of procedural programming that enforces a logical structure on the program being written to make it more efficient and easier to understand and modify is a programming language model organized around "objects" rather than "actions" and data rather than logic.
- Software may be applied in any situation for which a pre specified set of procedural steps has been defined. Information content and determinacy are important factors in determining the nature of a software application. Content refers to the meaning and form of incoming and outgoing information.
- Object-oriented programming is a programming language model organized around "objects" rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

5.10 Terminal Questions

1. Briefly explain software development process.
2. What is analysis and design?
3. Explain different stages in Software Testing.
4. Explain various methods of Programming.
5. Name few Software Applications

5.11 Answers

Self Assessment Questions

1. Sub Process
2. Correctness
3. Correspondence
4. Software Development Methodology
5. Programs
6. Data-flow Model
7. Entity-relation model
8. Relationships
9. object-oriented systems
10. Unit testing

11. Integration testing
12. Validation testing
13. Ada, Pascal, and dBASE
14. Is to identify all the objects you want to manipulate and how they relate to each other, an exercise often known as data modeling.
15. Mathematical logic
16. Real time
17. System

Terminal Questions

1. System development can be viewed as a process. Furthermore, the development itself, in essence, is a process of change, refinement, transformation, or addition to the existing product. (Refer section 5.2)
2. A more methodical approach to software design is proposed by structured methods, these structured methods are sets of notations and guidelines for software design. (Refer section 5.3)
3. The software engineering process may be viewed as the spiral. Initially, system engineering defines the role of software and leads to software requirements analysis, where the information domain, function, behavior, performance, constraints, and validation criteria for software-are established. (Refer section 5.4)
4. To general methods of programing are Structured programming and Object-oriented programming (Refer section 5.6)
5. Software applications can be neatly compartmentalized into different categories. System software Real time, software Business, Software is few to be mentioned (Refer section 5.7)

Book References:

- A Practitioner's Guide to Software Test Design by Lee Copeland (Jan 2004)
- Testing Computer Software, 2nd Edition by Cem Kaner, Jack Falk and Hung Q. Nguyen (Apr 12, 1999)
- Software Requirements by Karl E. Wiegers (Mar 26, 2003)
- Systematic Software Testing (Artech House Computer Library) by Rick D. Craig and Stefan P. Jaskiel (May 2002)

E-References

- www.seas.gwu.edu
- www.searchsoa.techtarget.com
- www.searchcio-idmarket.techtarget.com