Unit-3

Topics covered: *Software Construction, Coding Standards, Coding Framework, Reviews - Desk checks (Peer Reviews), Walkthroughs, Code Reviews, Inspections, Structured Programming, Structured Programming, Object-Oriented Programming, Automatic Code Generation, Pair Programming, Test-Driven Development, Configuration Management, Software Construction Artifacts.*

1. Software Construction

- The term software construction refers to the detailed creation of working software through a combination of coding, verification, unit testing, integration testing, and debugging.
- The Software Construction knowledge area (KA) is linked to all the other KAs, but it is most strongly linked to Software Design and Software Testing because the software construction process involves significant software design and testing.
- The process uses the design output and provides an input to testing ("design" and "testing" in this case referring to the activities, not the KAs). Boundaries between design, construction, and testing (if any) will vary depending on the software life cycle processes that are used in a project. Although some detailed design may be performed prior to construction, much design work is performed during the construction activity. Thus, the Software Construction KA is closely linked to the Software Design KA. Throughout construction, software engineers both unit test and integration test their work. Thus, the Software construction KA is closely linked to the Software Testing KA as well.
- Software construction typically produces the highest number of configuration items that need to be managed in a software project (source files, documentation, test cases, and so on). Thus, the Software Construction KA is also closely linked to the Software Configuration Management KA. While software quality is important in all the KAs, code is the ultimate deliverable of a software project, and thus the Software Quality KA is closely linked to the Software Construction KA. Since software construction requires knowledge of algorithms and of coding practices, it is closely related to the Computing Foundations KA, which is concerned with the computer science foundations that support the design and construction of software products. It is also related to project management, insofar as the management of construction can present considerable challenges.

Software Construction Fundamentals

Software construction fundamentals include

Minimizing Complexity

Most people are limited in their ability to hold complex structures and information in their working memories, especially over long periods of time. This proves to be a major factor influencing how people convey intent to computers and leads to one of the strongest drives in software construction: minimizing complexity. The need to reduce complexity applies to essentially every aspect of software construction and is particularly critical to testing of software constructions. In software construction, reduced complexity is achieved through emphasizing code creation that is simple and readable rather than clever.

Anticipating Change

Most software will change over time, and the anticipation of change drives many aspects of software construction; changes in the environments in which software operates also affect software in diverse ways. Anticipating change helps software engineers build extensible software, which means they can enhance a software product without disrupting the underlying structure.

Constructing for Verification

Constructing for verification means building software in such a way that faults can be readily found by the software engineers writing the software as well as by the testers and users during independent testing and operational activities. Specific techniques that support constructing for verification include following coding standards to support code reviews and unit testing, organizing code to support automated testing, and restricting the use of complex or hard-to-understand language structures, among others.

Reuse

Reuse refers to using existing assets in solving different problems. In software construction, typical assets that are reused include libraries, modules, components, source code, and commercial off-the-shelf (COTS) assets. Reuse is best practiced systematically, according to a well-defined, repeatable process. Systematic reuse can enable significant software productivity, quality, and cost improvements. Reuse has two closely related facets:"construction for reuse" and "construction with reuse." The former means to create reusable software assets, while the latter means to reuse software assets in the construction of a new solution. Reuse often transcends the boundary of projects, which means reused assets can be constructed in other projects or organizations.

Standards in Construction

Applying external or internal development standards during construction helps achieve a project's objectives for efficiency, quality, and cost. Specifically, the choices of allowable programming language subsets and usage standards are important aids in achieving higher security. Standards that directly affect construction issues include

- communication methods (for example, standards for document formats and contents)
- programming languages (for example, language standards for languages like Java and C++)*coding standards (for example, standards for naming conventions, layout, and indentation)

- platforms (for example, interface standards for operating system calls)
- Tools (for example, diagrammatic standards for notations like UML (Unified Modeling Language)).

2. Coding Standards

Good software development organizations want their programmers to maintain to some well-defined and standard style of coding called coding standards. They usually make their own coding standards and guidelines depending on what suits their organization best and based on the types of software they develop. It is very important for the programmers to maintain the coding standards otherwise the code will be rejected during code review.

Purpose of Having Coding Standards:

- A coding standard gives a uniform appearance to the codes written by different engineers.
- It improves readability, and maintainability of the code and it reduces complexity also.
- It helps in code reuse and helps to detect error easily.
- It promotes sound programming practices and increases efficiency of the programmers.

Some of the coding standards are given below:

1. **Limited use of globals:**
   These rules tell about which types of data that can be declared global and the data that can't be.

2. **Standard headers for different modules:**
   For better understanding and maintenance of the code, the header of different modules should follow some standard format and information. The header format must contain below things that is being used in various companies:
   - Name of the module
   - Date of module creation
   - Author of the module
   - Modification history
   - Synopsis of the module about what the module does
   - Different functions supported in the module along with their input output parameters
   - Global variables accessed or modified by the module

3. **Naming conventions for local variables, global variables, constants and functions:**
   Some of the naming conventions are given below:

- Meaningful and understandable variables name helps anyone to understand the reason of using it.
- Local variables should be named using camel case lettering starting with small letter (e.g. **localData**) whereas Global variables names should start with a capital letter (e.g. **GlobalData**). Constant names should be formed using capital letters only (e.g. **CONSDATA**).
- It is better to avoid the use of digits in variable names.
- The names of the function should be written in camel case starting with small letters.
- The name of the function must describe the reason of using the function clearly and briefly.

4. **Indentation:**
   Proper indentation is very important to increase the readability of the code. For making the code readable, programmers should use White spaces properly. Some of the spacing conventions are given below:
   - There must be a space after giving a comma between two function arguments.
   - Each nested block should be properly indented and spaced.
   - Proper Indentation should be there at the beginning and at the end of each block in the program.
   - All braces should start from a new line and the code following the end of braces also start from a new line.

5. **Error return values and exception handling conventions:**
   All functions that encountering an error condition should either return a 0 or 1 for simplifying the debugging. On the other hand, Coding guidelines give some general suggestions regarding the coding style that to be followed for the betterment of understandability and readability of the code. Some of the coding guidelines are given below :

6. **Avoid using a coding style that is too difficult to understand:**
   Code should be easily understandable. The complex code makes maintenance and debugging difficult and expensive.

7. **Avoid using an identifier for multiple purposes:**
   Each variable should be given a descriptive and meaningful name indicating the reason behind using it. This is not possible if an identifier is used for multiple purposes and thus it can lead to confusion to the reader. Moreover, it leads to more difficulty during future enhancements.

8. **Code should be well documented:**

The code should be properly commented for understanding easily. Comments regarding the statements increase the understandability of the code.

9. **Length of functions should not be very large:**
   Lengthy functions are very difficult to understand. That's why functions should be small enough to carry out small work and lengthy functions should be broken into small ones for completing small tasks.

Advantages of Coding Guidelines:

- Coding guidelines increase the efficiency of the software and reduces the development time.
- Coding guidelines help in detecting errors in the early phases, so it helps to reduce the extra cost incurred by the software project.
- If coding guidelines are maintained properly, then the software code increases readability and understandability thus it reduces the complexity of the code.
- It reduces the hidden cost for developing the software.

3. Coding Framework

A framework is a set of tools in programming on which to build well-structured, reliable software and systems.

A framework in programming is a tool that provides ready-made components or solutions that are customized in order to speed up development.

A framework may include a library, but is defined by the principle of inversion of control (IOC). With traditional programming, the custom code calls into the library to access reusable code. With IOC, the framework calls on custom pieces of code when necessary.

A framework can include support programs, compilers, code libraries, toolsets, and APIs to develop software and create systems. Open-source frameworks are always being updated and improved.

Why Frameworks are used in Software Development?

The purpose of a framework is to assist in development, providing standard, low-level functionality so that developers can focus efforts on the elements that make the project unique. The use of high-quality, pre-vetted functionality increases software reliability, speeds up programming time, and simplifies testing. With an active base of users and ongoing code improvements, frameworks help improve security and offer a base of support. Ultimately, frameworks are used to save time and money.

What are the Features of a Good Framework?

There are many kinds of frameworks, with some more popular than others. Developers often choose frameworks they are most familiar with, but that framework may not be the right for the job. Instead, consider the following features of good frameworks when deciding on the right framework for the project at hand:

- **Functionality** – choose a framework that provides the functionality needed for the project at hand, respecting that each framework has its limits and not investing in a framework that does far more than your project will ever need.
- **Consistency** – a framework can assist in consistency for large or distributed teams
- **Documentation** – choose a framework that has well-documented code and that provides implementation training
- **Active Community** – frameworks are only as strong as the user base of support. Choose a framework that is well-established with an active user base.

Challenges of Using a Software Framework

- Software frameworks can become a costly crutch if developers are not strong in the language the framework is based upon or if the developer over-relies on the framework instead of custom code, a problem which can lead to software bloat and performance issues.
- There are risks in choosing a framework that is too new or not well supported, which could require costly re-tooling if the framework becomes obsolete. Similarly, if the framework has limitations that are not well understood up front, this could impact the project.

Types of Programming Frameworks

There are a variety of different programming frameworks, each built upon a programming language and specializing in its function, whether it's working on a web app, database, or mobile app. In this section, we will break down various types of programming frameworks, popular frameworks for each type, and common examples.

1. Web Frameworks

Web application frameworks (WAF), or web frameworks (WF), support the development of web applications with web services, web resources, and web APIs. There are different web frameworks for both the front-end (how the web app looks) and the back-end (how it works).

2. Front-End Frameworks

Front-end frameworks (client-side frameworks) provide basic templates and components of HTML, CSS, and JavaScript for building the front-end of a website or web app.

4. **Reviews - Desk checks (Peer Reviews)**

Software Review is systematic inspection of software by one or more individuals who work together to find and resolve errors and defects in the software during the early stages of Software Development Life Cycle (SDLC).

Software review is an essential part of Software Development Life Cycle (SDLC) that helps software engineers in validating the quality, functionality and other vital features and components of the software. It is a whole process that includes testing the software product and it makes sure that it meets the requirements stated by the client.
Usually performed manually, software review is used to verify various documents like requirements, system designs, codes, test plans and test cases.

Objectives of Software Review:
The objective of software review is:

- To improve the productivity of the development team.
- To make the testing process time and cost effective.
- To make the final software with fewer defects.
- To eliminate the inadequacies.

Types of Software Reviews:
There are mainly 3 types of software reviews:

Software Peer Review:
Peer review is the process of assessing the technical content and quality of the product and it is usually conducted by the author of the work product along with some other developers.
Peer review is performed in order to examine or resolve the defects in the software, whose quality is also checked by other members of the team.
Peer Review has following types:

(i) Code Review:
Computer source code is examined in a systematic way.
(ii) Pair Programming:
It is a code review where two developers develop code together at the same platform.

(iii) Walkthrough:

Members of the development team is guided by author and other interested parties and the participants ask questions and make comments about defects.

(iv) Technical Review:
A team of highly qualified individuals examines the software product for its client's use and identifies technical defects from specifications and standards.

(v) Inspection:
In inspection the reviewers follow a well-defined process to find defects.

Software Management Review:
Software Management Review evaluates the work status. In this section decisions regarding downstream activities are taken.

Software Audit Review:
Software Audit Review is a type of external review in which one or more critics, who are not a part of the development team, organize an independent inspection of the software product and its processes to assess their compliance with stated specifications and standards. This is done by managerial level people.

Advantages of Software Review:

- Defects can be identified earlier stage of development (especially in formal review).
- Earlier inspection also reduces the maintenance cost of software.
- It can be used to train technical authors.
- It can be used to remove process inadequacies that encourage defects.

5. Structured Programming

In structured programming, we sub-divide the whole program into small modules so that the program becomes easy to understand. The purpose of structured programming is to linearize control flow through a computer program so that the execution sequence follows the sequence in which the code is written.

The dynamic structure of the program than resemble the static structure of the program. This enhances the readability, testability, and modifiability of the program. This linear flow of control can be managed by restricting the set of allowed applications construct to a single entry, single exit formats.

Why we use Structured Programming?

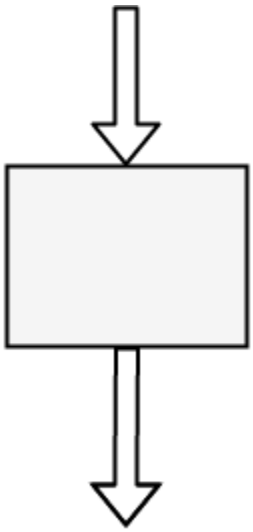
We use structured programming because it allows the programmer to understand the program easily. If a program consists of thousands of instructions and an error occurs then it is complicated to find that error in the whole program, but in structured programming, we can easily detect the error and then go to that location and correct it. This saves a lot of time.

These are the following rules in structured programming:

Structured Rule One: Code Block

If the entry conditions are correct, but the exit conditions are wrong, the error must be in the block. This is not true if the execution is allowed to jump into a block. The error might be anywhere in the program. Debugging under these circumstances is much harder.

Rule 1 of Structured Programming: A code block is structured, as shown in the figure. In flow-charting condition, a box with a single entry point and single exit point are structured. Structured programming is a method of making it evident that the program is correct.

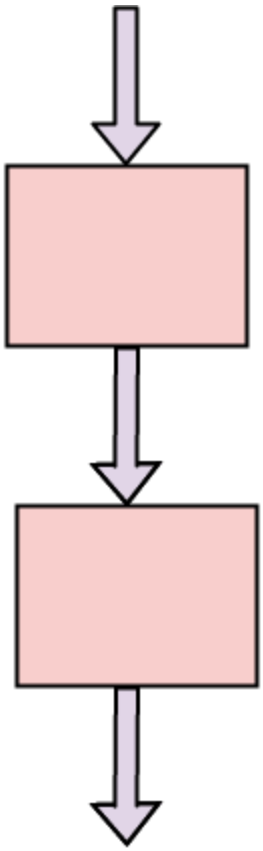


Rule1: Code block is structured

Structure Rule Two: Sequence

A sequence of blocks is correct if the exit conditions of each block match the entry conditions of the following block. Execution enters each block at the block's entry point and leaves through the block's exit point. The whole series can be regarded as a single block, with an entry point and an exit point.

Rule 2 of Structured Programming: Two or more code blocks in the sequence are structured, as shown in the figure.



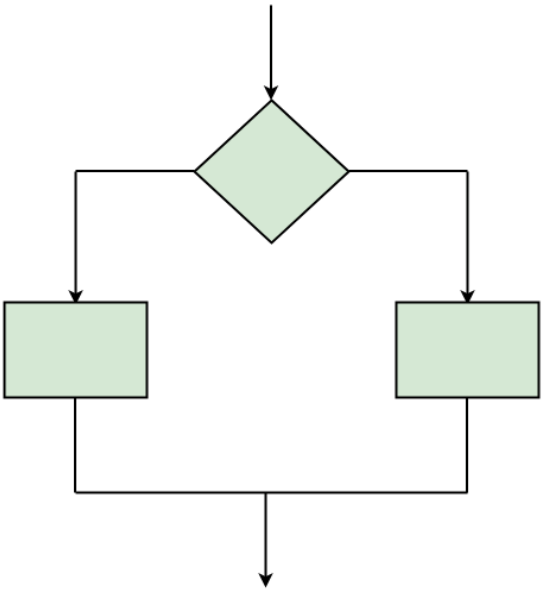
Rule2: A sequence of code blocks is structured

Structured Rule Three: Alternation

If-then-else is frequently called alternation (because there are alternative options). In structured programming, each choice is a code block. If alternation is organized as in the flowchart at right, then there is one entry point (at the top) and one exit point (at the bottom). The structure should be coded so that if the entry conditions are fulfilled, then the exit conditions are satisfied (just like a code block).

Rule 3 of Structured Programming: The alternation of two code blocks is structured, as shown in the figure.

An example of an entry condition for an alternation method is: register $8 includes a signed integer. The exit condition may be: register $8 includes the absolute value of the signed number. The branch structure is used to fulfill the exit condition.

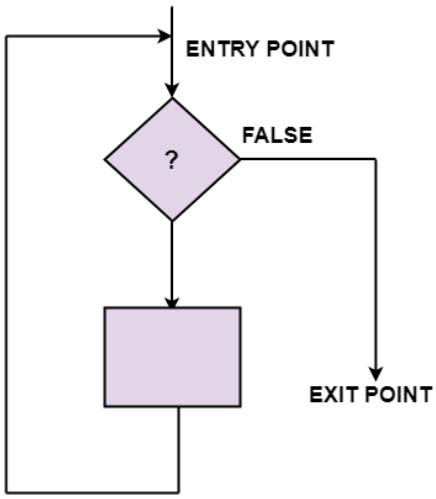


Rule 3: An alternation of code blocks is structured

Structured Rule 4: Iteration

Iteration (while-loop) is organized as at right. It also has one entry point and one exit point. The entry point has conditions that must be satisfied, and the exit point has requirements that will be fulfilled. There are no jumps into the form from external points of the code.

Rule 4 of Structured Programming: The iteration of a code block is structured, as shown in the figure.

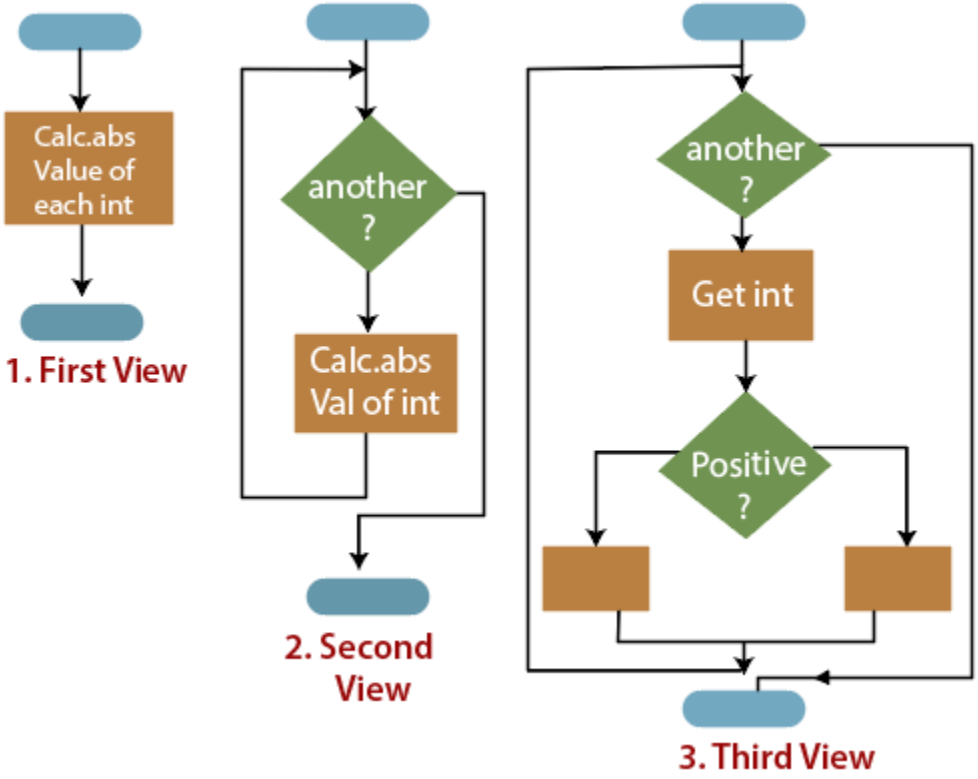


Rule 4: Iteration of code blocks is structured

Structured Rule 5: Nested Structures

In flowcharting conditions, any code block can be spread into any of the structures. If there is a portion of the flowchart that has a single entry point and a single exit point, it can be summarized as a single code block.

Rule 5 of Structured Programming: A structure (of any size) that has a single entry point and a single exit point is equivalent to a code block. For example, we are designing a program to go through a list of signed integers calculating the absolute value of each one. We may **(1)** first regard the program as one block, then **(2)** sketch in the iteration required, and finally **(3)** put in the details of the loop body, as shown in the figure.



The other control structures are the case, do-until, do-while, and for are not needed. However, they are sometimes convenient and are usually regarded as part of structured programming. In assembly language, they add little convenience.

6. **Object-Oriented Programming**

   Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Pre-requisites of OOPS:

- **Access Modifier**: Defines **access type** of the method i.e. from where it can be accessed in your application. In Java, there 4 type of the access specifiers.
    - **Public:** accessible in all class in your application.
    - **protected:** accessible within the package in which it is defined and in its **subclass(es)(including subclasses declared outside the package)**
    - **Private:** accessible only within the class in which it is defined.
    - **Default (declared/defined without using any modifier):** accessible within same class and package within which its class is defined.
- **The return type**: The data type of the value returned by the method or void if does not return a value.
- **Method Name**: the rules for field names apply to method names as well, but the convention is a little different.
- **Parameter list**: Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
- **Exception list**: The exceptions you expect by the method can throw; you can specify these exception(s).
- **Method body**: it is enclosed between braces. The code you need to be executed to perform your intended operations.

OOPs Concepts are as follows:

1. **Class**
2. **Object**
3. **Method and method passing**
4. **Pillars of OOPS**
    - **Abstraction**
    - **Encapsulation**
    - **Inheritance**
    - **Polymorphism**

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. **Modifiers**: A class can be public or has default access (Refer this for details).
2. **Class name:** The name should begin with a initial letter (capitalized by convention).
3. **Superclass (if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. **Interfaces (if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. **Body:** The class body surrounded by braces, { }.

Object is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:

1. **State:** It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behavior:** It is represented by methods of an object. It also reflects the response of an object with other objects.
3. **Identity:** It gives a unique name to an object and enables one object to interact with other objects.
4. **Method:** A method is a collection of statements that perform some specific task and return result to the caller. A method can perform some specific task without returning anything. Methods allow us to **reuse** the code without retyping the code. In Java, every method must be part of some class which is different from languages like C, C++ and Python. Methods are **time savers** and help us to **reuse** the code without retyping the code.

Pillars of OOPS:

Pillar 1: Abstraction

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

In java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

Pillar 2: Encapsulation

It is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

- Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
  - As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.
  - Encapsulation can be achieved by declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

Pillar 3: Inheritance

Inheritance is an important pillar of OOP (Object Oriented Programming). It is the mechanism in java by which one class is allows inheriting the features (fields and methods) of another class.

Important terminologies:

- **Super Class:** The class whose features are inherited is known as superclass (or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as subclass (or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Pillar 4: Polymorphism

It refers to the ability of OOPs programming languages to differentiate between entities with the same name efficiently. This is done by Java with the help of the signature and declaration of these entities.

7. **Automatic Code Generation**
    - Automatic code generation is the study of generative programming in the sense that the source code is generated automatically. In this paper, an approach based on component techniques that can produce in a systematic way correct, compatible and efficient database structures and manipulation function modules from abstract models is proposed. In contrast to some conventional software engineering methods, this approach has certain merits of improving software quality and shortening the software development cycle.

    - A fundamental notion of traditional software architectures is that of layers. Layers and hierarchical decomposition remain very useful in component systems. Each part of a component system, including the components themselves, can be layered as components may be located within particular layers of a larger architecture. To master the complexity of larger component systems, the architecture itself needs to be layered.

    - It is important to distinguish clearly between the layers formed by architecture and those formed by a meta-architecture. The layers formed by a meta-architecture are thus called tiers. Multi-tier client-server applications are an example of tiered architecture. However, the tiers proposed in the following differ from the tiers found in client-server architecture.

- Component system architecture, as introduced above, arranges an open set of component frameworks. This is a second-tier architecture, in which each of the component frameworks introduces first-tier architecture.

- It is important to notice the radical difference between tiers and traditional layers. Traditional layers, as seen from the bottom up, are of an increasingly abstract and increasingly application-specific nature. In a well-balanced layered system, all layers have their performance and resource implications. In contrast, tiers are of decreasing performance and resource relevance but of increasing structural relevance. Different tiers focus on different degrees of integration, but all are of similar application relevance.

- We studied automatic code generation extensively, including design the data table and data fields, create a data table, generate components and interface code, generate user interface code, set the menu structures, set the user code and extract metadata. The interface code and the component code of the middle layer corresponding to the target system are automatically generated.

- These functional modules are located in a proper layer in the multi-layer software system.

- Based on this architecture, programmers only need to design the structures of menus and data tables, and transform this information into a data dictionary. Furthermore, the system can automatically generate the metadata, user interfaces, data interfaces, and the business logic components and create the specified database table. The generated interface is unified and the middleware components are stable. The generated source code can be translated into executable code by a modem compiler. Moreover, the source code can be adapted to meet a specific software requirement. The generated target system provides flexibility to change due to global demands or local demands because some new components can be added to the existing system conveniently.

8. Pair Programming

- Pair programming is an agile software development technique originating from Extreme programming (XP) in which two developers team together on one computer. The two people work together to design, code and test user stories. Ideally, the two people would be equally skilled and would each have equal time at the keyboard.

- A common implementation of pair programming calls the programmer at the keyboard the *driver*, while the other is called the *navigator*. The navigator focuses on the overall direction of the programming. The collaboration between developers can be done in person or remotely.

- Pair programming is a collaborative effort that involves a lot of communication. The idea is to have the driver and navigator communicate, discuss approaches and solve issues that might be difficult for a single developer to detect.

How does pair programming work?

- Pair programming requires two developers, one workstation, one keyboard and a mouse. The pairings can be assigned or self-assigned.

- Pair programming uses the four eyes principle, which ensures two sets of eyes review the code that is being produced, even when there is a division of labor. While the driver writes the code, the navigator checks the code being written. The driver focuses on the specifics of coding, while the navigator checks the work, code quality and provides direction.

- The process starts with the developers receiving a well-defined task. They agree on one small goal at a time, such as writing code, testing or just taking notes. Any discussions on direction or corrections can be made after each goal, as to avoid interrupting the driver's flow. The two programmers can talk about the various techniques and challenges, with the results usually being higher quality code than when one person does the same work.

- The two developers take turns coding or reviewing and check each other's work as they go. Rotating roles regularly helps keep both developers alert and engaged. Organizations may also have the pair rotate roles to work on different tasks. This way, they get experience working on the different parts of the system being built.

- Depending on how the pairs are coordinated, junior and senior developers can work together, enabling senior developers to share their knowledge and working habits. It also helps new team members get up to speed on a project.

Benefits of pair programming

Pair programming includes the following advantages:

- **Fewer coding mistakes.** Another programmer is looking over the driver's code, which can help reduce mistakes and improve the quality of the code.
- **Knowledge is spread among the pairs.** Junior developers can pick up more skills from senior developers. And those unfamiliar with a process can be paired with someone who knows more about the process.
- **Reduced effort to coordinate.** Developers will get used to collaborating and coordinating their efforts.
- **Increased resiliency.** Pair programming helps developers understand each part of a codebase, meaning the environment will not be dependent on a single person for repairs if something breaks.

Challenges of pair programming

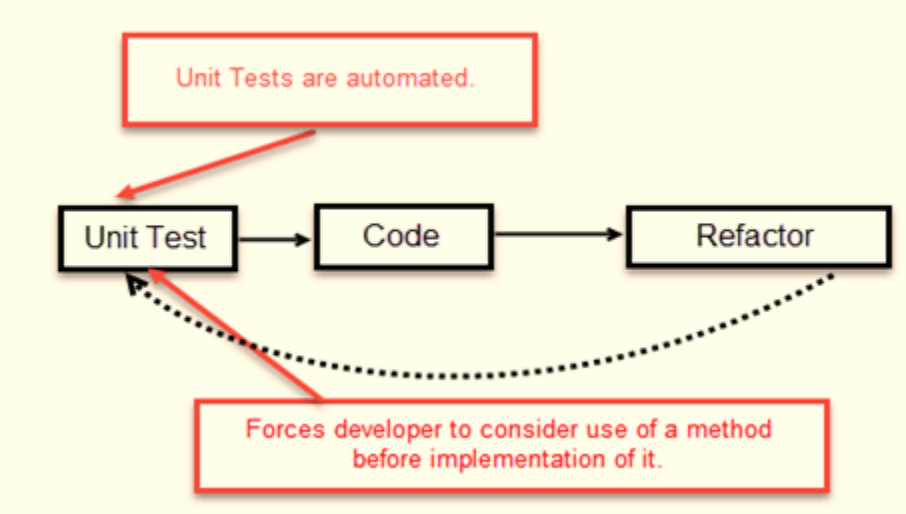
Pair programming includes the following pitfalls:

- **Efficiency.** Common logic might dictate that pair programming would reduce productivity by 50%, because two developers are working on the same project at one time.
- **Equally engaging pairs.** If both developers do not equally engage in the project, then there is less chance that knowledge will be shared, and it is highly probable that one developer will participate less than the other.
- **Social and interactive process.** It is hard for those who work better alone. Pairs that have trouble together might be better suited to work by themselves; forcing them to collaborate may hurt their work ethic.

- **Sustainability.** The pace may not be suited to practicing hours at a time. Likely, developers will need breaks at different times.

9. Test-Driven Development

Test Driven Development (TDD) is software development approach in which test cases are developed to specify and validate what the code will do. In simple terms, test cases for each functionality are created and tested first and if the test fails then the new code is written in order to pass the test and making code simple and bug-free.
Test-Driven Development starts with designing and developing tests for every small functionality of an application. TDD framework instructs developers to write new code only if an automated test has failed. This avoids duplication of code. The TDD full form is Test-driven development.



The simple concept of TDD is to write and correct the failed tests before writing new code (before development). This helps to avoid duplication of code as we write a small amount of code at a time in order to pass tests. (Tests are nothing but requirement conditions that we need to test to fulfill them).

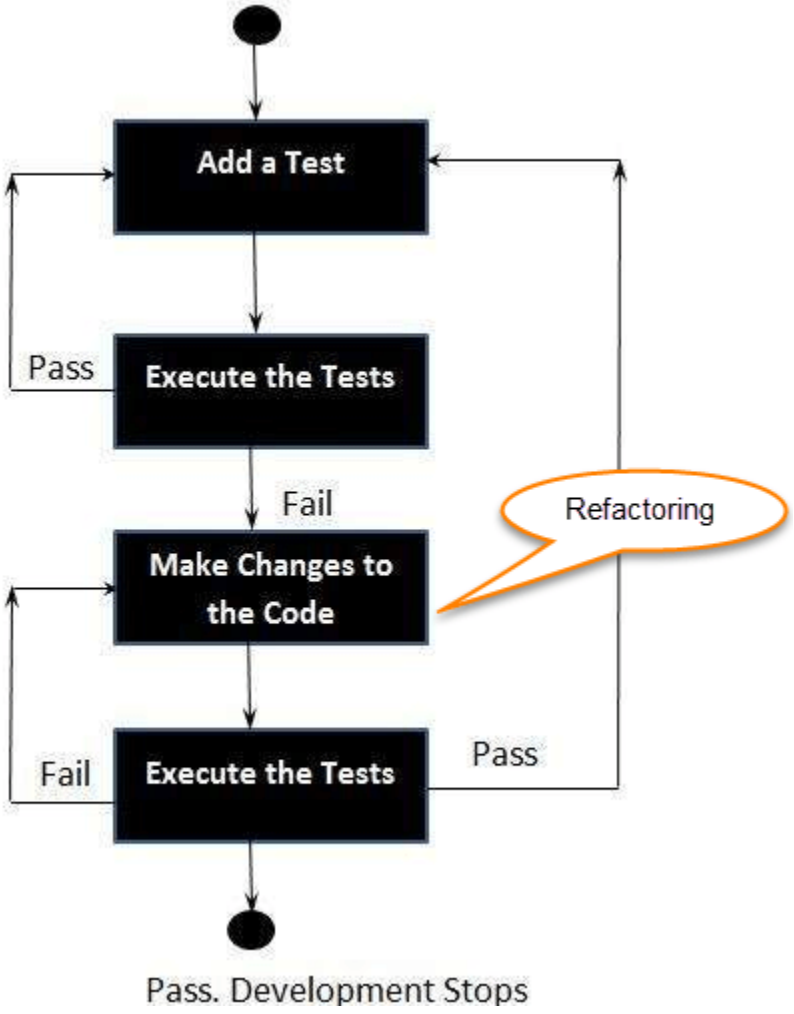
Test-Driven development is a process of developing and running automated test before actual development of the application. Hence, TDD sometimes also called as **Test First Development.**

How to perform TDD Test

Following steps define how to perform TDD test,

1. Add a test.

2. Run all tests and see if any new test fails.
3. Write some code.
4. Run tests and Refactor code.
5. Repeat.



Five Steps of Test-Driven Development

10. Configuration Management

- When we develop software, the product (software) undergoes many changes in their maintenance phase; we need to handle these changes effectively.
- Several individual (programs) works together to achieve these common goals. This individual produces several work product (SC Items) e.g., Intermediate version of modules or test data used during debugging, parts of the final product.
- The elements that comprise all information produced as a part of the software process are collectively called a software configuration.

- As software development progresses, the number of Software Configuration elements (SCI's) grow rapidly.
- **These are handled and controlled by SCM. This is where we require software configuration management.**
- A configuration of the product refers not only to the product's constituent but also to a particular version of the component.
- Therefore, SCM is the discipline which

  - Identify change
  - Monitor and control change
  - Ensure the proper implementation of change made to the item.
  - Auditing and reporting on the change made.

- Configuration Management (CM) is a technique of identifying, organizing, and controlling modification to software being built by a programming team.
- **The objective is to maximize productivity by minimizing mistakes (errors).**
- CM is used to essential due to the inventory management, library management, and updation management of the items essential for the project.

Why do we need Configuration Management?

Multiple people are working on software which is consistently updating. It may be a method where multiple version, branches, authors are involved in a software project, and the team is geographically distributed and works concurrently. It changes in user requirements, and policy, budget, schedules need to be accommodated.

Importance of SCM

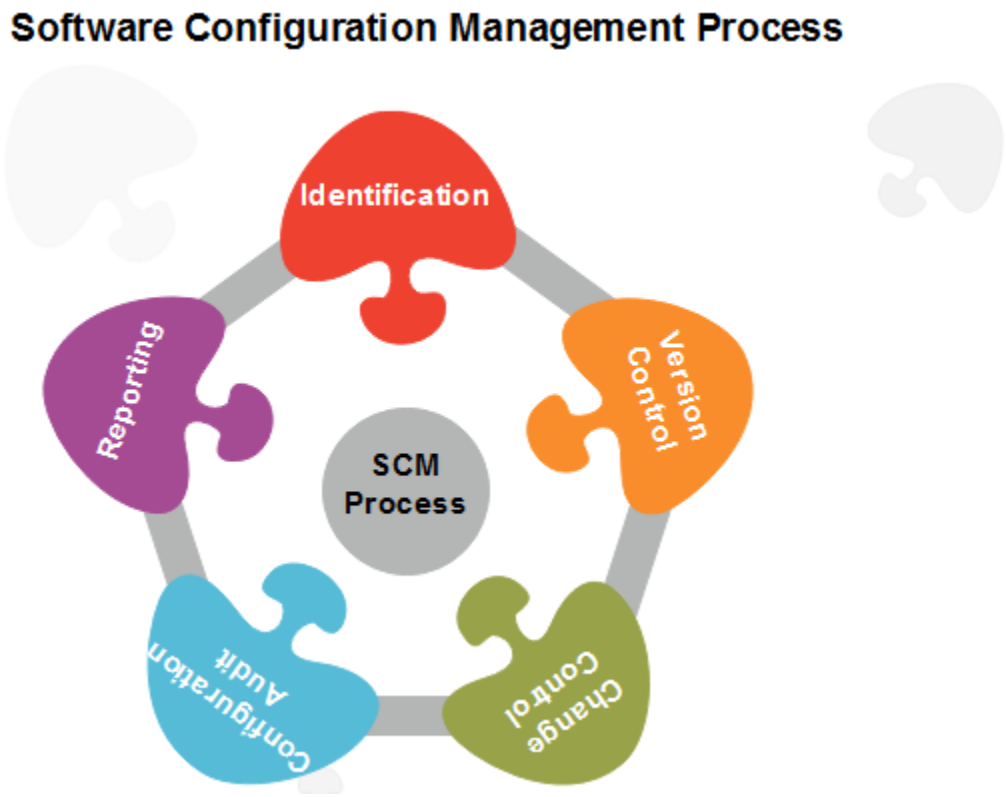
- It is practical in controlling and managing the access to various SCIs e.g., by preventing the two members of a team for checking out the same component for modification at the same time.
- **It provides the tool to ensure that changes are being properly implemented.**
- It has the capability of describing and storing the various constituent of software.
- SCM is used in keeping a system in a consistent state by automatically producing derived version upon modification of the same component.

SCM Process

It uses the tools which keep that the necessary change has been implemented adequately to the appropriate component. The SCM process defines a number of tasks:

- Identification of objects in the software configuration

- Version Control
- Change Control
- Configuration Audit
- Status Reporting



Identification

- **Basic Object:** Unit of Text created by a software engineer during analysis, design, code, or test.
- **Aggregate Object:** A collection of essential objects and other aggregate objects. Design Specification is an aggregate object.
- Each object has a set of distinct characteristics that identify it uniquely: a name, a description, a list of resources, and a "realization."
- **The interrelationships between configuration objects can be described with a Module Interconnection Language (MIL).**

Version Control

- Version Control combines procedures and tools to handle different version of configuration objects that are generated during the software process.

Change Control

- James Bach describes change control in the context of SCM is: Change Control is Vital. But the forces that make it essential also make it annoying.
- We worry about change because a small confusion in the code can create a big failure in the product. But it can also fix a significant failure or enable incredible new capabilities.
- A burdensome change control process could effectively discourage them from doing creative work.
- A change request is submitted and calculated to assess technical merit; potential side effects, the overall impact on other configuration objects and system functions, and projected cost of the change.
- The results of the evaluations are presented as a change report, which is used by a change control authority (CCA) - a person or a group who makes a final decision on the status and priority of the change.
- The "check-in" and "check-out" process implements two necessary elements of change control-**access control** and **synchronization control**.
- **Access Control** governs which software engineers have the authority to access and modify a particular configuration object.
- **Synchronization Control** helps to ensure that parallel changes, performed by two different people, don't overwrite one another.

Configuration Audit

- SCM audits to verify that the software product satisfies the baselines requirements and ensures that what is built and what is delivered.
- SCM audits also ensure that traceability is maintained between all CIs and that all work requests are associated with one or more CI modification.

11. Software Construction Artifacts

- To put it simply, an artifact is a by-product of software development. It's anything that is created so a piece of software can be developed. This might include things like data models, diagrams, setup scripts — the list goes on.

- "Artifact" is a pretty broad term when it comes to software development. Most pieces of software have a lot of artifacts that are necessary for them to run. Some artifacts explain how a piece of software is supposed to work, while others actually allow that program to run.

Why Is It Called an Artifact?

- Think about an archaeological dig site. During an archaeological dig, any man-made object that's dug up is an artifact. These artifacts help us determine what civilization may have been like and help us paint a picture of their society.

- Similarly, digital artifacts are all products of people. They help other developers see the thought process behind what goes into developing a piece of software. This, in turn, helps lead developers to further decisions and gives them a better understanding of how to proceed.

The Artifact Process

- Typically, a software development team will come up with a list of necessary artifacts for a piece of software before coding anything. These include things like risk assessments, source code, diagrams, and use cases. This is done during the research phase of the project.

- Developing a piece of software without any of this is like building a house without blueprints. The process would be a huge mess, and crucial pieces would inevitably be left out. In this way, getting all your artifacts together is one of the most crucial parts of the software development life cycle.

- Once all the initial artifacts are compiled, a development team can begin programming and building the actual program. Throughout this process, further artifacts might be developed. These can come up at any time and include everything from new sketches to use cases.

- Some artifacts, like the end-user agreement, may need to be built after the software is complete for total accuracy. These may be added in before the program is compiled and sent out for consumption.

Software Artifacts

There are a lot of different parts of any given piece of software that can be artifacts. Here are a few of the most common examples.

Use Cases

Use cases are descriptions of how users are meant to perform tasks on a given program, website, or piece of software. These relate directly to the function of the site, making them important artifacts.

Unified Modeling Language (UML)

- UML is a way of visualizing and plotting out the way a piece of software works. It works to map out links, processes, etc.

- Like use cases, UML don't directly help software run, but it's a key step in designing and programming a piece of software.

Class Diagrams

Like UML, a class diagram is a way to map out the structure of a piece of software or application. Class diagrams are used to map out links and processes that happen between clicks in a more visual way.

Images

Any images used to help develop the piece of software are considered artifacts. These might be example images used to help in the design of the product, or preliminary design images. It could even be simple sketches and diagrams used to help map out the software.

Software Documents

- The majority of the artifacts are software documents. Any document that describes the characteristics or attributes of a piece of software is an artifact. These can relate to the software's architecture, technical side, end-user processes, and marketing.

- The majority of these artifacts won't even cross the mind of the user. They're meant for developers and those who might be interested in the software from a large-scale business perspective. Much of it is technical, and simply not of interest to the typical user.

Source Code

The source code is the language used to program a given piece of software. It's not the physical code itself, but the system that allows that code to work. This, too, is an artifact according to software developers.