



Software Engineering Course's Code: CSE 305

Chapter 7. Implementation , Testing and Deployment

7.1. Implementation Issues

7.2. Build Process

7.3. Testing and Deployment



Implementation Issues

- Implementation is the work of developing the software itself.
- Following are some issues or tips which will help you as a developer, but from a process perspective

- ☐ You should code, when you are alert
- ☐ Write code for the people, not for the computers :
Write your code so that it's as self-documenting as possible. Modern compilers handle all sorts of optimizations

```
// if account flag is zero (HOW)
if ( accountFlag == 0 ) ...

// if establishing a new account (WHY)
if ( accountFlag == 0 ) ...

// establish a new account (COMMENT FOR SECTION HEADER, CODE TELLS WHY)
if ( accountType == AccountType.NewAccount ) {
    ...
}
```

Take a look at this code.

- It's a pretty good example of self-documenting code or rather how to get to self-documenting code
- But, saying the how, isn't as good as saying why.
- Comments should represent why, and How will be managed by Code
- From comments, other developers can get the idea, what you are doing

Implementation Issues

➤ Some Issues:

- ☐ Write your test, Comments, and Exceptional Handling before You write functional code (Test Driven Development)
- ☐ Break long method into manageable pieces: The Google style guide for C++ recommends that if a function is longer than 40 lines, you need to think about breaking it down
- ☐ Make each class represent a single, intuitive concept (**High Cohesion**)
- ☐ Methods focused on a single task, preferable without side effects : Changes to data sources outside of the method. (**Low Coupling**)
- ☐ Obviously this can't be entirely avoided or it may not even be desirable to do so, but you do want to minimize side effects, as those side effects are the very, very error-prone element to using methods in the first place.

Implementation Issues

➤ Some Issues:

- ☐ Writing the code for 2nd time? Extract (make) the code as a method
- ☐ Optimize Code, when you think it is necessary, otherwise it's better if you give the responsibility towards different tools.

[Code optimization is any method of code modification to improve code quality and efficiency. A program may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer input/output operations.]

- ☐ Tools will identify the areas of code that will actually benefit from optimization.

Implementation Issues

Implementation skills are training well elsewhere

These tips will help you understand the difference
between coder and developer

People, not computers, are the primary focus

Let the computer handle optimization for computers

These tips make the developers using them more effective

Coders

Anyone who can write some code is often referred to as a coder by the people outside of the tech industry. But, usually, coders are considered the least trained or experienced level of programmers. These individuals do not have the same algorithmic knowledge as a programmer or developer, as they are often a beginner in the field, skilled in just one coding language. Coders are usually given the job of writing forthright pieces of code that can easily be delegated by the developers. As some are put-off by the title, it is sometimes used interchangeably with “Junior Programmer” or “Junior Developer.”

Implementation Issues

Implementation skills are training well elsewhere

These tips will help you understand the difference
between coder and developer

People, not computers, are the primary focus

Let the computer handle optimization for computers

These tips make the developers using them more effective

Developers and Programmers

The titles Developer and programmer are often used interchangeably. They are more experienced code writers who are versed in at least two to three languages and write clean, error free codes. They can apply their algorithmic knowledge to create more sophisticated levels of software coding.

Developers in some firms are sometimes referred to as the start to finish overseers of a project, who are responsible for the overall design of the application.

Refactoring



Refactoring

Refactoring is the process of making changes to your code so that the external behaviors of the code are not changed, but the internal structure is improved.



Refactoring



- Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.
- Its heart is a series of small behavior preserving transformations.
- Each incremental transformation (called a "refactoring") does little, but a sequence of these transformations can produce a significant restructuring.
- Since each refactoring is small, it's less likely to go wrong. The system is kept fully working after each refactoring, reducing the chances that a system can get seriously broken during the restructuring.

Why Refactoring?



- Refactoring makes code understandable and easy to extend
- Refactoring makes it quicker to create complicated system
- Excess code can be removed, so that it is easier to understand modify
- By refactoring code, others' code can also be understood better
- Code can be written quicker

When to Refactor?



- You should refactor:
 - ❖ Any time that you see a better way to do things, to remove **code smells**
 - ❖ “Better” means making the code easier to understand and to modify in the future
 - ❖ You can do so without breaking the code
 - ❖ Unit tests are essential for this
- You should not refactor:
 - ❖ Stable code that won't need to change
 - ❖ Someone else's code
 - ❖ Unless the other person agrees to it or it belongs to you
 - ❖ Not an issue in Agile Programming since code is communal

Refactoring Process



- Make a small change
 - ❖ a single refactoring
- Run all the tests to ensure everything still works
- If everything works, move on to the next refactoring
- If not, fix the problem, or undo the change, so you still have a working system

Ways of Refactoring

➤ Extract Method:

Problem

You have a code fragment that can be grouped together.

```
void printOwing() {  
    printBanner();  
  
    // Print details.  
    System.out.println("name: " + name);  
    System.out.println("amount: " + getOutstanding());  
}
```

Solution

Move this code to a separate new method (or function) and replace the old code with a call to the method.

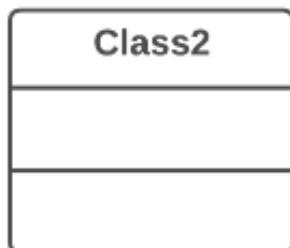
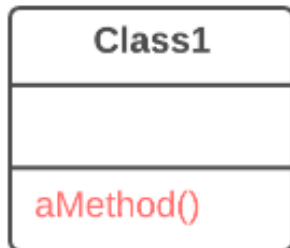
```
void printOwing() {  
    printBanner();  
    printDetails(getOutstanding());  
}  
  
void printDetails(double outstanding) {  
    System.out.println("name: " + name);  
    System.out.println("amount: " + outstanding);  
}
```

Ways of Refactoring

➤ Move Method:

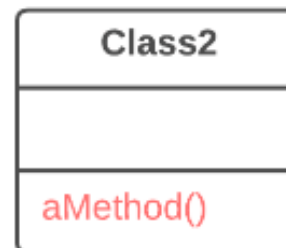
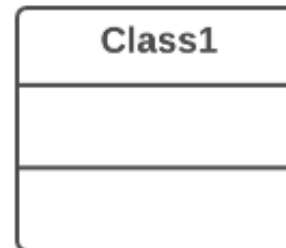
Problem

A method is used more in another class than in its own class.



Solution

Create a new method in the class that uses the method the most, then move code from the old method to there. Turn the code of the original method into a reference to the new method in the other class or else remove it entirely.

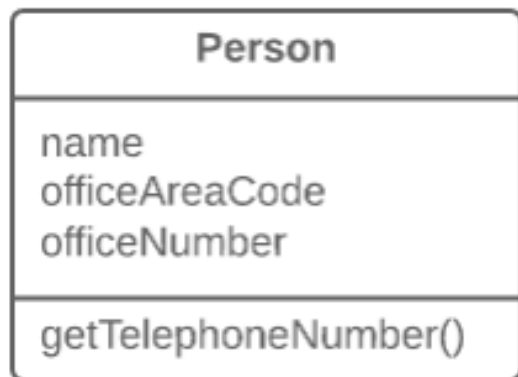


Ways of Refactoring

➤ Extract Class:

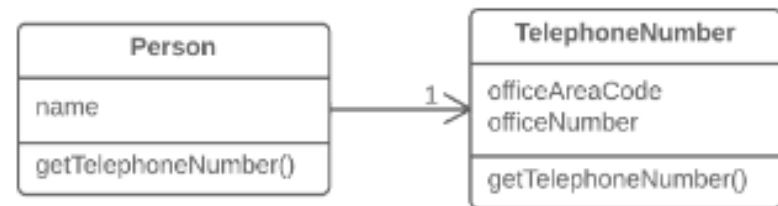
Problem

When one class does the work of two, awkwardness results.



Solution

Instead, create a new class and place the fields and methods responsible for the relevant functionality in it.

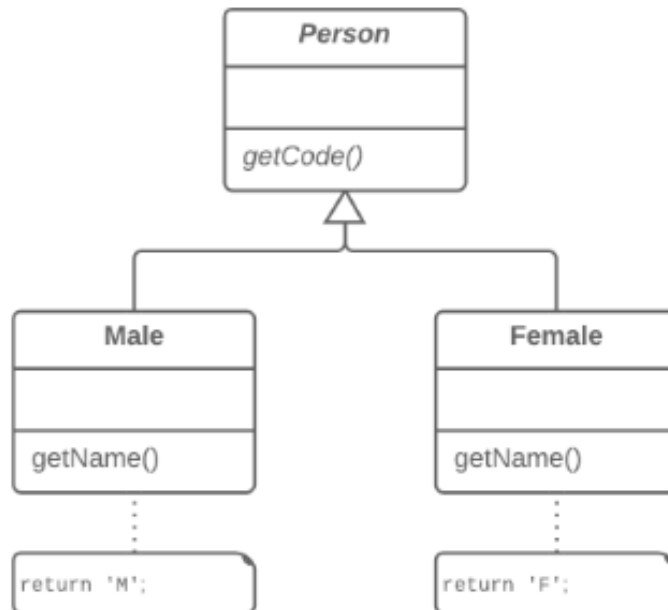


Ways of Refactoring

➤ Replace Subclasses with field:

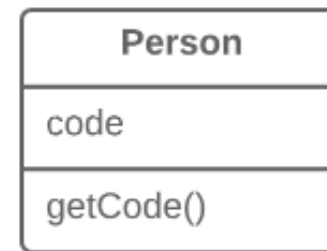
Problem

You have subclasses differing only in their (constant-returning) methods.



Solution

Replace the methods with fields in the parent class and delete the subclasses.



Ways of Refactoring



➤ Replace Conditional with Polymorphism:

Problem

You have a conditional that performs various actions depending on object type or properties.

Solution

Create subclasses matching the branches of the conditional. In them, create a shared method and move code from the corresponding branch of the conditional to it. Then replace the conditional with the relevant method call. The result is that the proper implementation will be attained via polymorphism depending on the object class.

Ways of Refactoring

➤ Replace Conditional with Polymorphism:

```
class Bird {
    // ...
    double getSpeed() {
        switch (type) {
            case EUROPEAN:
                return getBaseSpeed();
            case AFRICAN:
                return getBaseSpeed() - getLoadFactor() * numberOfCoco;
            case NORWEGIAN_BLUE:
                return (isNailed) ? 0 : getBaseSpeed(voltage);
        }
        throw new RuntimeException("Should be unreachable");
    }
}
```

```
abstract class Bird {
    // ...
    abstract double getSpeed();
}

class European extends Bird {
    double getSpeed() {
        return getBaseSpeed();
    }
}

class African extends Bird {
    double getSpeed() {
        return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
    }
}

class NorwegianBlue extends Bird {
    double getSpeed() {
        return (isNailed) ? 0 : getBaseSpeed(voltage);
    }
}

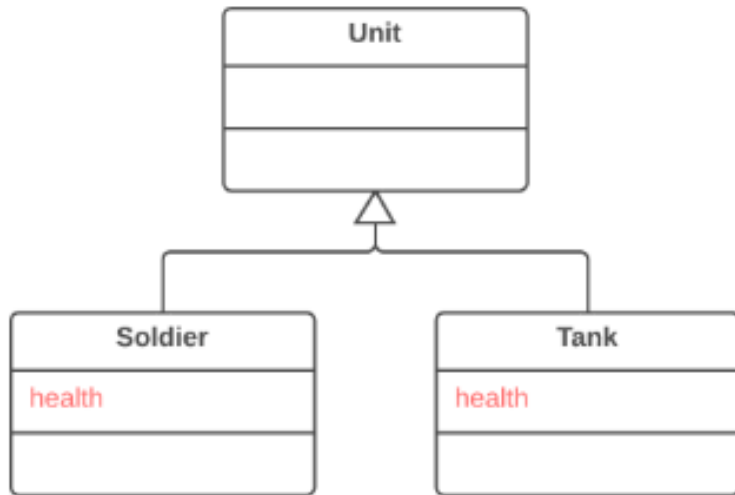
// Somewhere in client code
speed = bird.getSpeed();
```

Ways of Refactoring

➤ Pull Up Field:

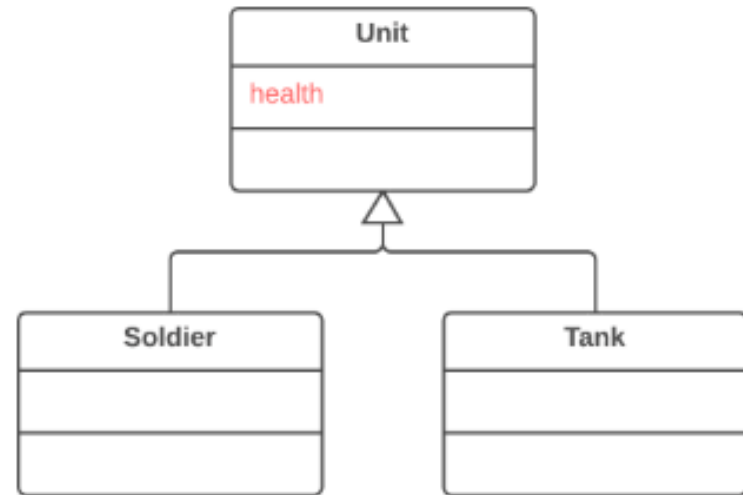
Problem

Two classes have the same field.



Solution

Remove the field from subclasses and move it to the superclass.

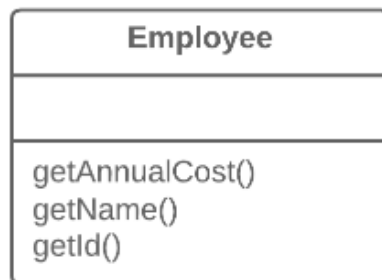
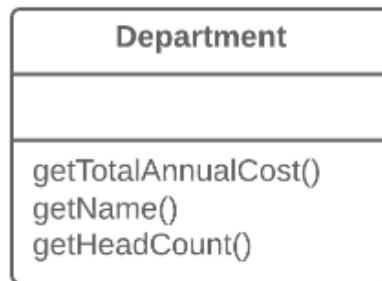


Ways of Refactoring

➤ Extract SuperClass:

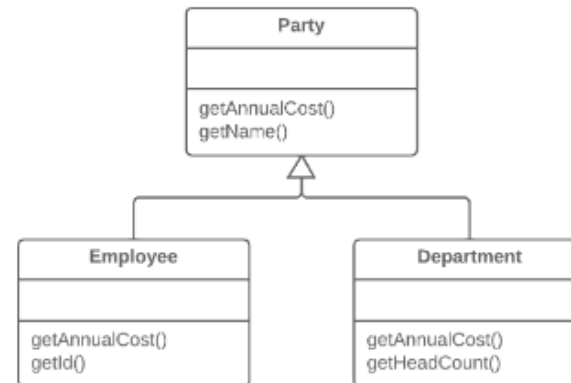
Problem

You have two classes with common fields and methods.



Solution

Create a shared superclass for them and move all the identical fields and methods to it.



Code Smells



- Similar to the way we see patterns emerge in design, we also see patterns of bad code emerge.
- These are referred to as **anti-patterns**.
- Many of these anti-patterns are identified in the book, Refactoring by Martin Fowler. Where he refers to them as **code smells**.
- The purpose of these code smells is to sniff out what is bad in the code.

Code Smells



- Code smells are common design problems
- Bad coding due to code smells are normally unclear, complicated or duplicated
- Examples:
 - ❖ Duplicate Code
 - ❖ Long Methods
 - ❖ Long Parameter Lists
 - ❖ Large Classes
 - ❖ Switch Statements
 - ❖ Feature Envy
 - ❖ Data Classes
 - ❖ Data Clumps

Code Smells : Duplicate Code

"Duplicated Code"

"Duplicated" code is when you have blocks of code that are similar, but have slight differences. These blocks of code appear in multiple places in your software.

An Example:

- Let's say you're working on a social networking website.
- A user can create **textual posts** that appear on their **news feed**, on a **friend's wall**, or in a **group**.



Code Smells : Duplicate Code

- You first create the ability to post on the **news feed**.
- These posts were visible by all the user's friends on their **news feeds**.
- Once that feature is complete, you then develop the code for posting on a **friend's wall**.
- Since you already have the functionality written for posting,, you just copy and paste that code to reuse it for posting on a friend's wall. When you create the functionality to **post in a group**, you do the same thing



Code Smells : Duplicate Code

- Now, as the website expands, you also want the ability to **post pictures and not just texts**.
- To add this functionality, you now have to update the code in all the three places.
- What if you had other places in the code where a user could post?
- What if you missed one of those when you had to update the code to allow the posting of pictures?
- As you can see, if you would just combine the ability to post in **one accessible method**, all forms of post could use it.



- when it came time to make a change, you would have just needed to update it in one location.

How do treat Duplicate Code through refactoring?

- Create a common method that can be **accessible** from all the places , which need it.

How to do this refactoring? Some solutions:

- If the same code is found in two or more methods in the same class: use “**Extract Method**” refactoring and place calls for the new method in both places.
- If the same code is found in two subclasses of the same level:
Use **Extract Method** for both classes, followed by **Pull Up Field** for the fields used in the method that you’re pulling up.
- If duplicate code is found in two different classes: use **Extract Superclass** in order to create a single superclass for these classes that maintains all the previous functionality

Code Smells : Long Method



"Long Method"

You don't want to have long methods. Having a "long method" can sometimes indicate that there is more occurring in that method than should be. Or it is more complex than it needs to be.

- Among all types of object-oriented code, classes with short methods live longest.
- The longer a method or function is, the harder it becomes to understand and maintain it.
- In addition, long methods offer the perfect hiding place for unwanted duplicate code.

How do treat Long Method through refactoring?



How to do this refactoring? Some solutions

- To reduce the length of a method body, use **Extract Method**
- Conditional operators and loops are a good clue that code can be moved to a separate method. For conditionals, use **Decompose Conditional**.
If loops are in the way, try **Extract Method**

Code Smells : Large Class



- These large classes are commonly referred to as **God classes**, **Blob classes**, or **Black Hole classes**. They just keep getting bigger and bigger
- They normally start off as **a regular size class**. But as more responsibilities are needed, these classes seem like the appropriate place to put the Responsibilities.
- To avoid this, you need to be explicit about the purpose of a class, and keep the class cohesive, so it does one thing well. If a functionality is not specific to the class's responsibility, you can place it in other classes

How do treat Long Classes through refactoring?

- When a class is consisting of too many features, think about splitting it up:
- **Extract Class** helps if part of the behavior of the large class can be spun off into a separate component.
- **Extract Subclass** helps if part of the behavior of the large class can be implemented in different ways or is used in rare cases.
- **Extract Interface** helps if it's necessary to have a list of the operations and behaviors that the client can use.
- If a large class is responsible for the graphical interface, you may try to move some of its data and behavior to a separate domain object. In doing so, it may be necessary to store copies of some data in two places and keep the data consistent. **Duplicate Observed Data** offers a way to do this.

Code Smell : Data Class

- On the contrary, having a too small class is also an issue, these are often called data classes

"Data Classes"

"Data classes" are classes that contain only data and no real functionality. Generally these classes would have getter and setter methods, but not much else.

How do treat Data Classes through refactoring? Some solutions

- Review the client code that uses the class. In it, you may find functionality that would be better located in the data class itself. If this is the case, use **Move Method** and **Extract Method** to migrate this functionality to the data class.

Code Smell : Data Clumps



"Data Clumps"

"Data clumps" are groups of data appearing together in the instance variables of a class, or parameters to methods.

- For example, Let's assume the following method that takes x, y and z as parameters

```
public void doSomething (int x, int y, int z) {  
    ...  
}
```

- Now, just like this method several other methods can also takes x, y and z as parameters.
- So, instead of treating x, y, and z as parameters, we can store them in an object.

Code Smell : Data Clumps



- Suppose, we create a 3D point object. Now the method will look like as follows

```
public void doSomething (Point3D point) {  
  
    ...  
}
```

- So, we also have a Point3D class



```
public class Point3D {  
  
    private int x;  
    private int y;  
    private int z;  
  
    public int getX() {  
        return x;  
    }  
  
    public int getY() {  
        return y;  
    }  
  
    public int getZ() {  
        return z;  
    }  
}
```

```
public void setX(int newX) {  
    x = newX;  
}  
  
public void setY(int newY) {  
    y = newY;  
}  
  
public void setZ(int newZ) {  
    z = newZ;  
}  
}
```

Code Smell : Data Clumps

- We need to be careful here though. We want to make sure that we are not just creating “**data classes**” type code smell.
- We want these classes to do more than just store data.
- So, we need to add in this class some meaningful methods

```
public class Point3D {  
  
    private int x;  
    private int y;  
    private int z;  
  
    public int getX() {  
        return x;  
    }  
  
    public int getY() {  
        return y;  
    }  
  
    public int getZ() {  
        return z;  
    }  
}
```

```
    public void setX(int newX) {  
        x = newX;  
    }  
  
    public void setY(int newY) {  
        y = newY;  
    }  
  
    public void setZ(int newZ) {  
        z = newZ;  
    }  
}
```

How do treat Data Clumps through refactoring?

- If repeating data comprises the fields of a class, use **Extract Class** to move the fields to their own class.
- If some of the data is passed to other methods, think about passing the entire data object to the method instead of just individual fields. **Preserve Whole Object** will help with this.
- An example of **Preserve Whole Object**

Problem:

```
int low = daysTempRange.getLow();  
int high = daysTempRange.getHigh();  
boolean withinPlan = plan.withinRange(low, high);
```

Solution:

```
boolean withinPlan = plan.withinRange(daysTempRange);
```

Code Smell : Long Parameter List



- More than three or four parameters for a method.
- Having a long parameter list can make the method difficult to use correctly.

➤ How do treat Long Parameter List through refactoring?

Some solution

- If there are several unrelated data elements, sometimes you can merge them into a single parameter object via **Introduce Parameter Object**.
- Check what values are passed to parameters. If some of the arguments are just results of method calls of another object, then passed that object as a method parameter

Code Smell : Long Parameter List

➤ Example of **Introduce Parameter Object**

Problem

Your methods contain a repeating group of parameters.

Customer
amountInvoicedIn (start : Date, end : Date) amountReceivedIn (start : Date, end : Date) amountOverdueIn (start : Date, end : Date)

Solution

Replace these parameters with an object.

Customer
amountInvoicedIn (date : DateRange) amountReceivedIn (date : DateRange) amountOverdueIn (date : DateRange)

Code Smell : Feature Envy

- A method accesses the data of another object more than its own data.
- **How do treat Feature Envy through refactoring? Some solution**
 - ❑ We need to keep data and functions that use those data in the same class.
 - ❑ If a method clearly should be moved to another place, use **Move Method**.
 - ❑ If only part of a method accesses the data of another object, use **Extract Method** to move the part in question.
 - ❑ If a method uses functions from several other classes, first determine which class contains most of the data used. Then place the method in this class along with the other data. Alternatively, use **Extract Method** to split the method into several parts that can be placed in different places in different classes.

Code Smell : Switch Statements



- You have a complex switch operator or sequence of if statements.
- Often code for a single switch can be scattered in different places in the program. When a new condition is added, you have to find all the switch code and modify it.
- As a rule of thumb, when you see switch you should think of polymorphism.
- **How do treat Feature Envy through refactoring? Some solution**
- After specifying the inheritance structure, use **Replace Conditional with Polymorphism**

Chapter 7. Implementation, Testing and Deployment

7.1. Implementation Issues

7.2. Build Process

7.3. Testing and Deployment

Build Process



- Through **build process**, the source code is compiled and compressed into **binary files** such as **jar/war/exe/zip**
- In general, IDE users click the green button (Run Button) and command users use a set of commands to make builds.
- But, we should go for automating this process
- **Build automation** can automate **a wide variety of tasks** that developers do in their day-to-day activities like: **Downloading dependencies. Compiling source code into binary code. Packaging that binary code. Running tests and deployment to production systems**
- Different automatic tools exist, which can help us build our projects a little bit more efficiently,
- There are a number of options : Make, Ant, Gradle, Maven

Build Process : Automated Tools



make and Ant

Traditional tools with rich history and extensive documentation

Ant primarily used for Java builds and various file system tasks

make – the original - portable and language-independent

Gradle and Maven

Both are expandable and maintainable build tools

Maven improved on Ant; Gradle incorporates benefits from both

Gradle has simpler, shorter, non-XML configuration files

Example of Configuration file in Make Tool

```
34 CC = g++
35 DEBUG = -g
36 CFLAGS = -Wall -std=c++11 -c $(DEBUG)
37 LFLAGS = -Wall $(DEBUG)
38
39 .PHONY : clean all passenger_factory_test passenger_generator_test
passenger_test route_test stop_test
40
41 all : $(EXEFILE)
42
43 $(EXEFILE) : route.o stop.o bus.o passenger.o passenger_factory.o
passenger_generator.o simulator.o $(BINDIR)
44 $(CC) $(LFLAGS) route.o stop.o bus.o passenger.o passenger_factory.o
passenger_generator.o simulator.o -o $@
45
46 simulator.o : route.h bus.h stop.h simulator.cc
47 $(CC) $(CFLAGS) simulator.cc
```

- It help us to build also dependencies
 - ❑ when you try and build one code, one program, it will automatically build anything that's necessary [Line 43 in this file]
- Official guide to Make from the GNU project:

<https://www.gnu.org/software/make/>

Example of Ant

```
12 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
13   <modelVersion>4.0.0</modelVersion>
14
15   <parent>
16     <groupId>org.jacoco</groupId>
17     <artifactId>org.jacoco.build</artifactId>
18     <version>0.7.6.201602180812</version>
19     <relativePath>../org.jacoco.build</relativePath>
20   </parent>
21
22   <artifactId>org.jacoco.agent</artifactId>
23
24   <name>JaCoCo :: Agent</name>
25   <description>JaCoCo Agent</description>
26
27   <build>
28     <sourceDirectory>src</sourceDirectory>
```

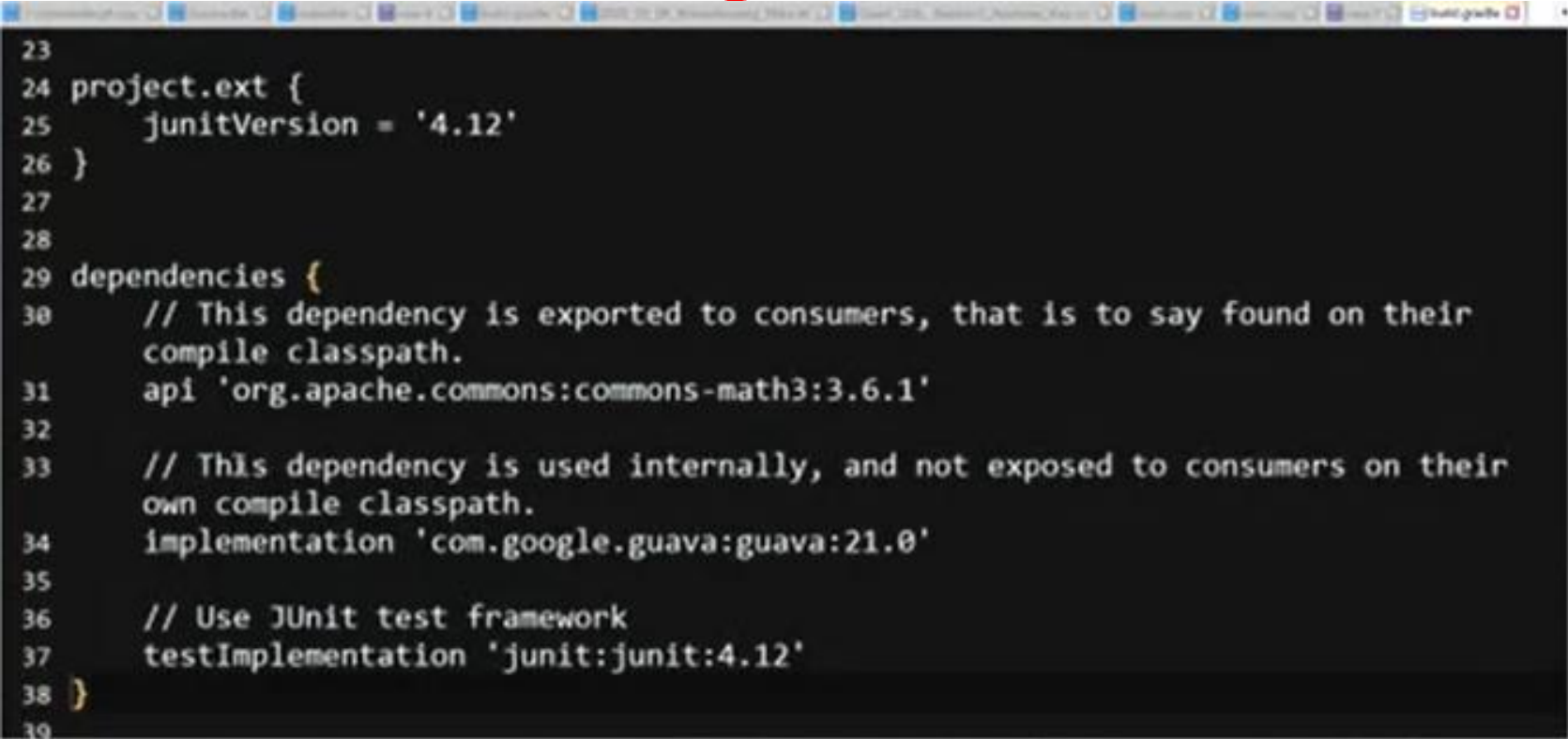
- Follow the XML standard
- At line 27, the build directory is displayed

Example of Gradle

```
 8  * Extended by Mike Whalen for Coursera "Introduction to Software Testing" course.
 9  * 8/2/2017
10  */
11
12 // Apply the java-library plugin to add support for Java Library
13 apply plugin: 'java-library'
14 apply plugin: 'jacoco'
15
16 // In this section you declare where to find the dependencies of your project
17 repositories {
18     // Use jcenter for resolving your dependencies.
19     // You can declare any Maven/Ivy/file repository here.
20     jcenter()
21     mavenCentral()
22 }
23
24 project.ext {
25     junitVersion = '4.12'
26 }
```

- simpler commands like apply plugin, Java library, apply plug-in, JaCoCo.
- Those are understood by the Gradle tool and it knows where to go looking for these kinds of information.
- In line 16 – you can see repositories, this is where we want to look for all of these kinds of dependencies

Example of Gradle



```
23
24 project.ext {
25     junitVersion = '4.12'
26 }
27
28
29 dependencies {
30     // This dependency is exported to consumers, that is to say found on their
    compile classpath.
31     api 'org.apache.commons:commons-math3:3.6.1'
32
33     // This dependency is used internally, and not exposed to consumers on their
    own compile classpath.
34     implementation 'com.google.guava:guava:21.0'
35
36     // Use JUnit test framework
37     testImplementation 'junit:junit:4.12'
38 }
39
```

- You'll see a little bit more detail on how this works,
- where various APIs are used,
- there is the JUnit testing framework [in line 36] being included into this build
- Adding in things is usually as simple as a single line in this Gradle file.

Build Process



Build Process

Modern build tools provide reproducible tasks on demand

make was the beginning and is still used widely, especially for C/C++

Ant provided the first “modern” build tool

Maven and Gradle are the prevailing build tools for Java.

Build Process



Build Process

Modern build tools provide reproducible tasks on demand

make was the beginning and is still used widely, especially for C/C++

Ant provided the first “modern” build tool

Maven and Gradle are the prevailing build tools for Java.

Chapter 7. Implementation, Testing and Deployment

7.1. Implementation Issues

7.2. Build Process

7.3. Testing and Deployment

What is Deployment



- Software deployment includes all the process required for preparing a software application to run and operate in a specific environment.
- It involves installation, configuration, testing and making changes to optimize the performance of the software. It can either be carried out manually or through automated systems.
- Tasks like installing, uninstalling and updating software applications on each computer are time consuming.
- Software deployment services reduce the time and make the process error free. The software can be easily controlled and managed through deployment.
- Getting software out of the hands of the developers into the hands of the Users

Application Environment

➤ An **application environment** is the combination of the hardware and software resources required to run an application

Include:

- Application code/executables
- Software stack (libraries, apps, middleware, OS)
- Networking infrastructure
- Hardware (compute, memory and storage)

➤ There are a **variety of environment types** depending on the application's stage in the lifecycle : **Development, QA, Staging and Production environment**



Pre-Production Environment

- The **pre-production environments** are those platforms that the application resides on in various forms as it gets prepared for production

Pre-production environments



Development

Project is
being actively
coded



QA

QA team tests
the
application's
components



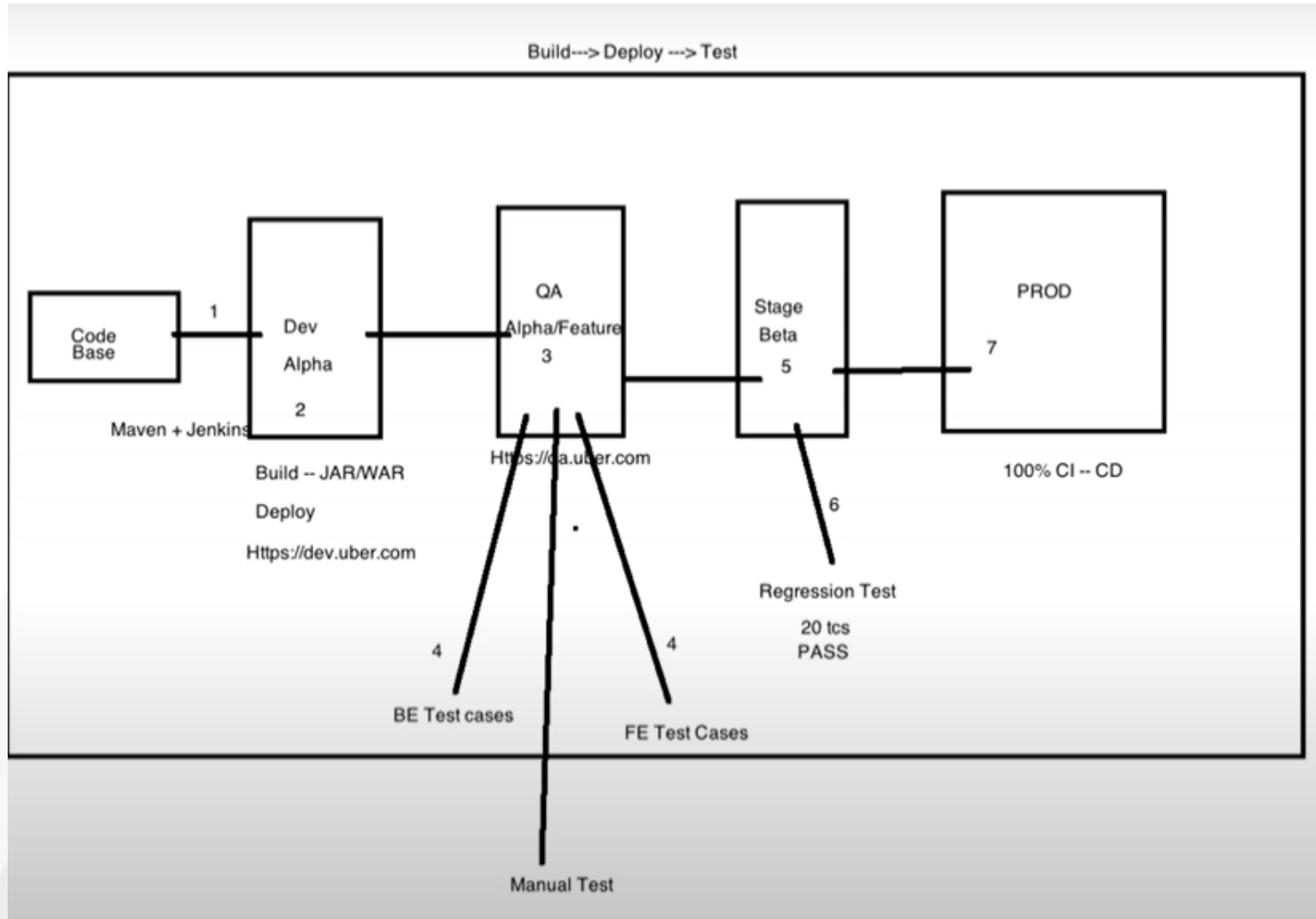
Staging

Replicates
production
environment but
not meant for
general users

Production/Deployment Environment

- Entire solution stack ++
- Intended for all users
- Take load into consideration
- Other non-functional requirements
 - Security
 - Reliability
 - Scalability
- More complex than pre-production environments

Build- Deploy and Test Process



- DevOps software development follow this kind of pipe
- Full Form of CI-CD is Continuous Integration Continuous Development

On-Premises Deployment

➤ There are **several options** for deploying application environments:
On-Premises and Cloud Based

- System and infrastructure reside in-house
- Offers greater control of the application
- Organization is responsible for everything
- Usually more expensive when compared to cloud deployment



Traditional software Deployment process

On-Premises Deployment are based on Traditional software Deployment process

- **The foot and hand model:**
 - ✓ Run around on foot and install software by hand.
 - ✓ Only viable for small client base.
 - ✓ Expensive for the companies.
- **The self-service model:**
 - ✓ The end users install the software themselves.
 - ✓ Scales well.
 - ✓ Low costs for the company.
 - ✓ Becomes difficult as the complexity of installation and configuration increases.

Cloud Deployment Types



Public

Shared infrastructure over internet and hardware is owned by the provider



Private

Infrastructure is provisioned for a single organization



Hybrid

Some infrastructure provided over the internet and some provisioned by a single organization

- **Public cloud providers** include Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform, and IBM Cloud. The public cloud is the most common due to its scalability and cost
- AWS is also a **private cloud service provider**. The main advantage of a private cloud is increased security

Cloud Deployment Types



Public

Shared infrastructure over internet and hardware is owned by the provider



Private

Infrastructure is provisioned for a single organization



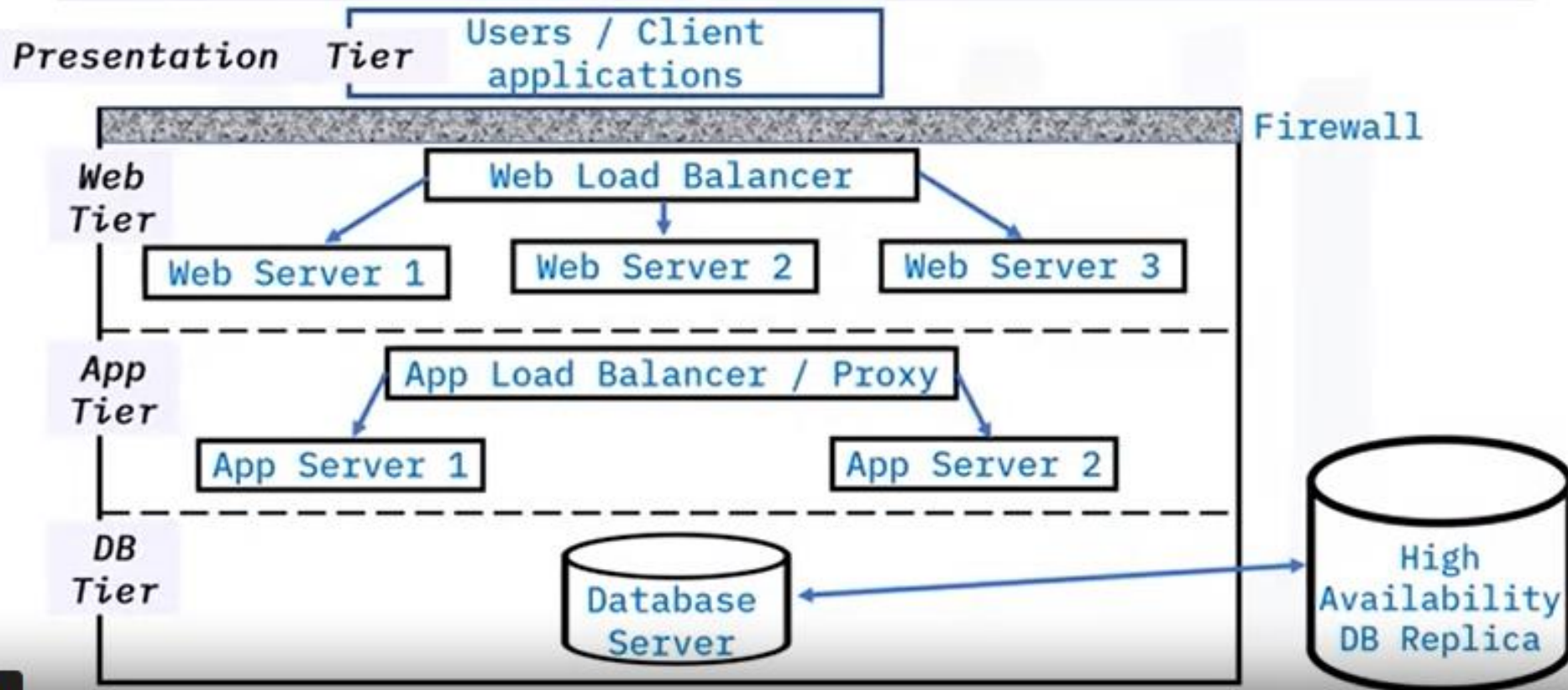
Hybrid

Some infrastructure provided over the internet and some provisioned by a single organization

- **Public cloud providers** include Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform, and IBM Cloud. The public cloud is the most common due to its scalability and cost
- AWS is also a **private cloud service provider**. The main advantage of a private cloud is increased security

Production Deployment Infrastructure

Let's consider an **n-tier architecture** required to **deploy an application** in a **production environment** and represent the infrastructure using a diagram

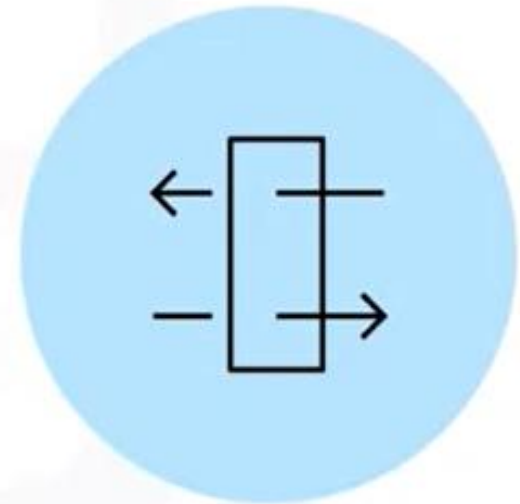


Different Components of Production Deployment Infrastructure



Firewall

- Monitors traffic between an interior and an exterior network
- Permits or blocks data based on a set of security rules
- Acts as a barrier between networks to block viruses and hackers from accessing the internal network



Different Components of Production Deployment Infrastructure



Load balancers

Purpose: distributes traffic efficiently amongst multiple servers

Functions:

- Prevents server traffic overload
- Maximizes server capabilities and responsiveness
- Ensures no one server is overworked
- Manages concurrent requests fast and reliably



Different Components of Production Deployment Infrastructure

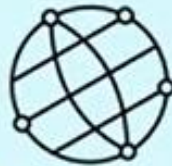
Web and Application Servers



Servers

Provide services, resources, data, or applications to a client

Store, process, and manage network data, devices, and systems



Web Servers

Delivers web pages, files, images, and videos to the client

Responds to HTTP requests from a browser



App Servers

Runs the apps and provides the app to the client or the web server

Stores code that determines how data can be created, stored, or changed

Different Components of Production Deployment Infrastructure



Databases and database servers

- Databases are a collection of related data stored on a computer that can be accessed in various ways
- DBMS controls a database by connecting it to users or other programs
- Database servers control the flow and storage of data



Different Components of Production Deployment Infrastructure



Proxy server

- An intermediate server that handles requests between two tiers
- Can be used for load balancing, system optimization, caching, as a firewall, obscuring the source of a request, encrypting messages, scanning for malware, and more
- Can improve efficiency, privacy, and security





Software Engineering

Course's Code: CSE 305

Chapter 7. Implementation, Testing and Deployment

7.1. Implementation Issues

7.2. Build Process

7.3. Testing and Deployment



Testing Definition



Testing: Definitions

The process of executing a program (or part of a program) with the intention of finding errors(Myers, via Humphrey)

The purpose of testing is to find errors

Testing is the process of trying to discover every conceivable fault or weakness in a work product (Myers, via Kit)

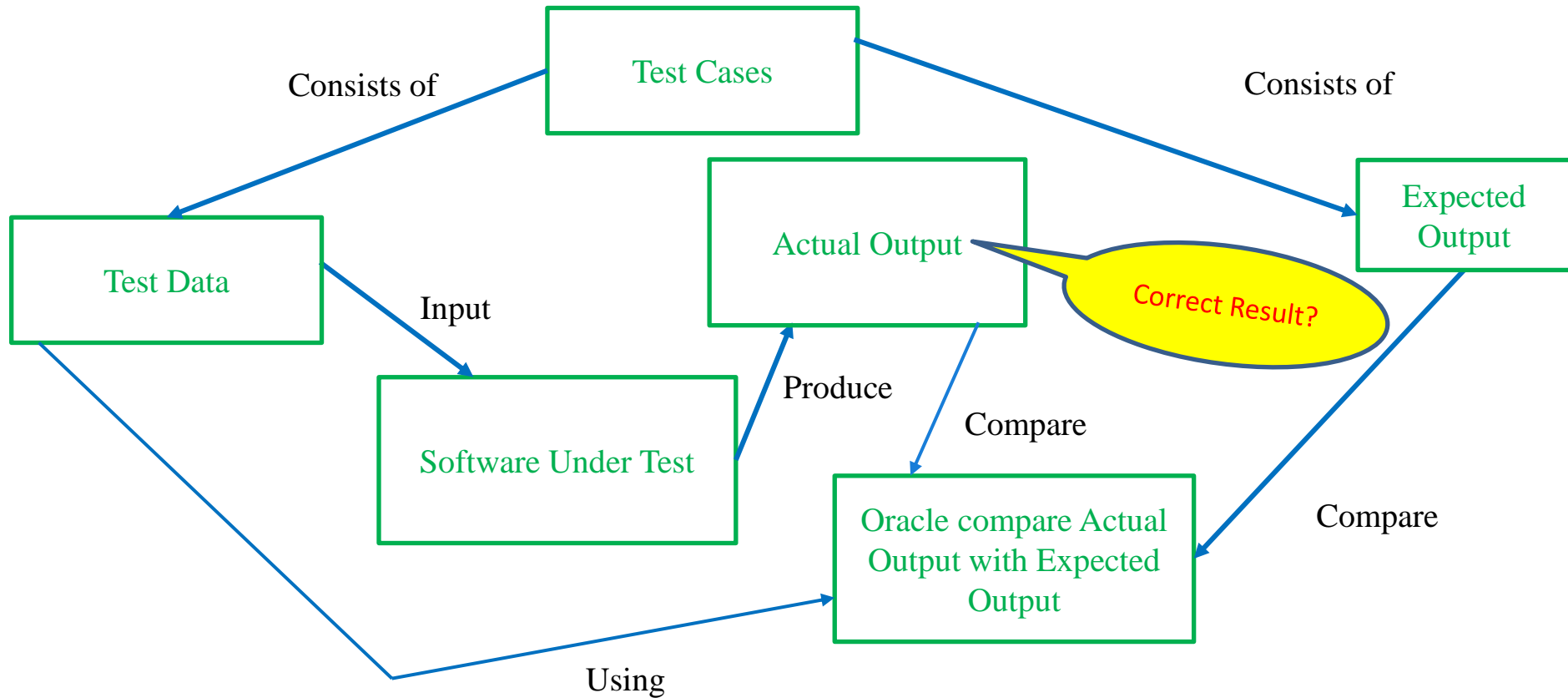
The process of searching for errors (Kaner)

Testing Objective



- Integrate quality checks throughout SDLC
- Purpose
 - Ensure software meets requirements

What is Test?



Test Case and Test Data



Test Data

Inputs which have been devised to test a system

Test Cases

Inputs to the system and the predicted outputs from operating the system on these inputs, if the system performs to its specification

Basic Terminology: Error, Defect, Failure

- ❖ You may have noticed that developers sometimes say errors, sometimes they say mistakes, or bugs, or faults, or failures.
- ❖ Although these words all look the same, they do have differences in their meaning.

Error, mistake, defect,
bug, fault, failure...

Aren't they all the
same?



Basic Terminology: Failure

- ❖ Failure represents the situation, when the final user is expecting the system to do something, but the system is doing something different.
- ❖ A common example is, when you have a blue screen.
- ❖ Instead of having the result you were expecting, you have this big crash.
- ❖ This is a failure.

Failure

A component or system behaves in a way that it is not expected.



STOP 0x00000000: A fatal exception 0x0 has occurred at 0x00000000 in 0x00000000 - 0x00000000. The current application will be terminated.

- Press any key to terminate the current application.
- Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue

Basic Terminology: Defect

- ❖ A failure is usually caused by a defect.
- ❖ Defect, bug or faults - all are synonyms.
- ❖ For example. it is an incorrect statement that you have in your source code, like a greater than ($>$) that should have been greater than or equal to ($>=$).
- ❖ It is important to notice that maybe the bug exists in the source code, but it never becomes a failure.
- ❖ That can happen for different reasons, for example, that part of the system was actually never executed.
- ❖ So, a bug only becomes a failure when it goes to the final user and makes the system to behave incorrectly.

Defect (or bug, fault)

A flaw in a component that can cause the system to behave incorrectly.
e.g., an incorrect statement.

A defect, if encountered during execution, may cause a failure.

Basic Terminology: Defect

- If you pay attention to this code, you don't have to understand it, but there's a fairly small mistake in there: there are two goto fail lines.
- This is definitely a bug and this bug really caused a failure in many of Apple's systems.

Fault in Apple's Secure Socket Layer code

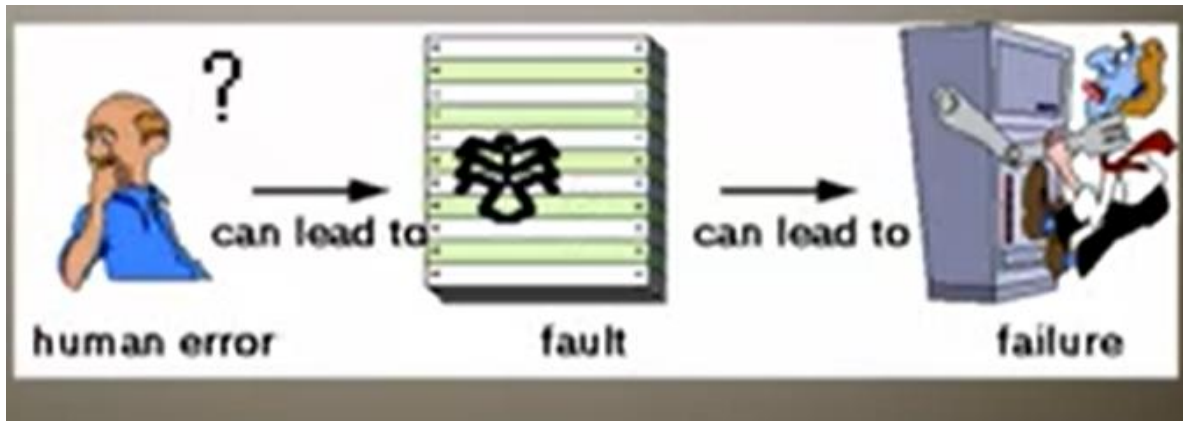
```
1 static OSStatus
2 SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer :
3                                uint8_t *signature, UInt16 signatureLen
4 {
5     OSStatus      err;
6     ...
7
8     if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
9         goto fail;
10    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11        goto fail;
12    goto fail;
13    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
14        goto fail;
15    ...
16
17 fail:
18     SSLFreeBuffer(&signedHashes);
19     SSLFreeBuffer(&hashCtx);
20     return err;
21 }
```

```
1 static OSStatus
2 SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer :
3                                uint8_t *signature, UInt16 signatureLen
4 {
5     OSStatus      err;
6     ...
7
8     if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
9         goto fail;
10    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11        goto fail;
12    goto fail;
13    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
14        goto fail;
15    ...
16
17 fail:
18     SSLFreeBuffer(&signedHashes);
19     SSLFreeBuffer(&hashCtx);
20     return err;
21 }
```

- A good example of a bug is the famous one on the Apple's SSL layer

Basic Terminology: Error

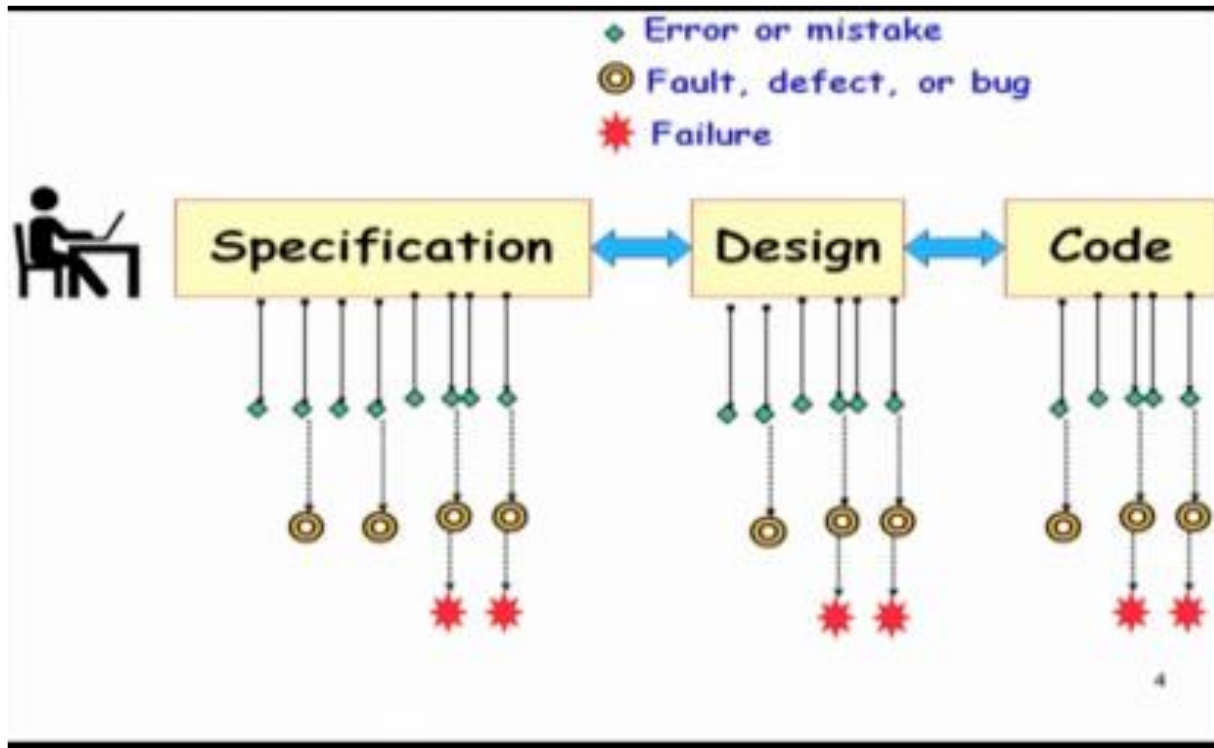
- ❖ So, it can be that the human forgot to think about a specific corner case, and because of this,
- ❖ the person introduced a bug, and this bug later became a failure, because system behaved in an incorrect way.



Error
(mistake)

A human action that
produces an incorrect
result.

Basic Terminology: Error, Defect, Failure



Test Driven Development (TDD)

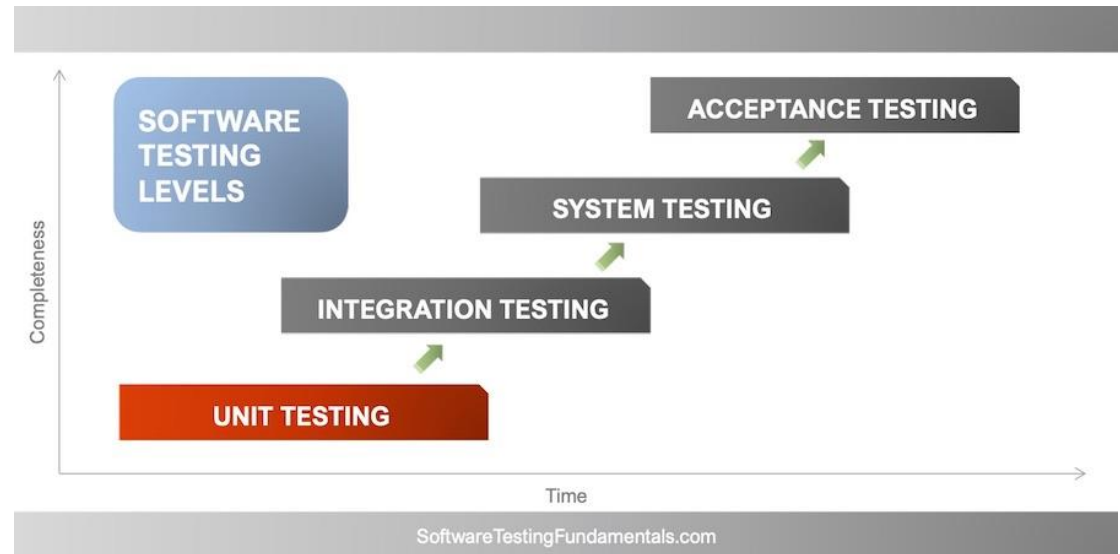


- Test Driven Development (TDD) is software development approach in which test cases are developed to specify and validate what the code will do.
- In simple terms, test cases for each functionality are created and tested first
- If the test fails then the new code is written in order to pass the test and making code simple and bug-free.



Test Levels

- The term Test Level provides an indication of the focus of the testing, and the types of problems it is likely to uncover.
- The typical levels of testing are:
 - ❖ Unit Testing
 - ❖ Integration Testing
 - ❖ System Testing
 - ❖ Acceptance Testing



Unit Testing



- Test each module individually
- A unit or a module is the smallest part of an application like functions/procedures, classes, interfaces
- Unit tests are usually written and run by software developers

Why Unit Testing?

- Unit tests help to fix bugs early in the development cycle and save costs.
- It helps the developers to understand the code base and enables them to make changes quickly
- Good unit tests serve as project documentation
- Unit tests help with code re-use. Migrate both your code and your tests to your new project. Tweak the code until the tests run again.

Unit Testing



When is Unit Testing performed?

- Unit Testing is the first level of software testing and is performed prior to Integration Testing.
- Though unit testing is normally performed after coding, sometimes, specially in test-driven development (TDD), automated unit tests are written prior to coding.

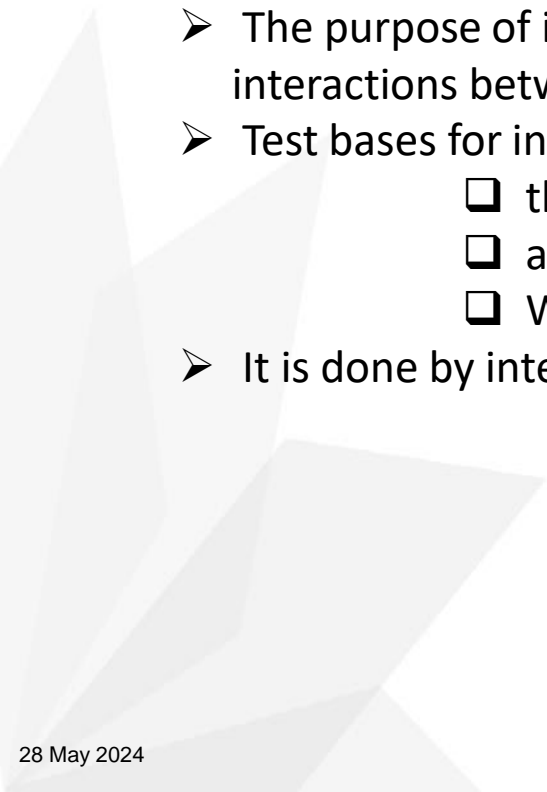
Who performs it?

- It is normally performed by software developers themselves or their peers.
- In rare cases, it may also be performed by independent software testers but they will need to have access to the code and have an understanding of the architecture and design.

Integration Testing

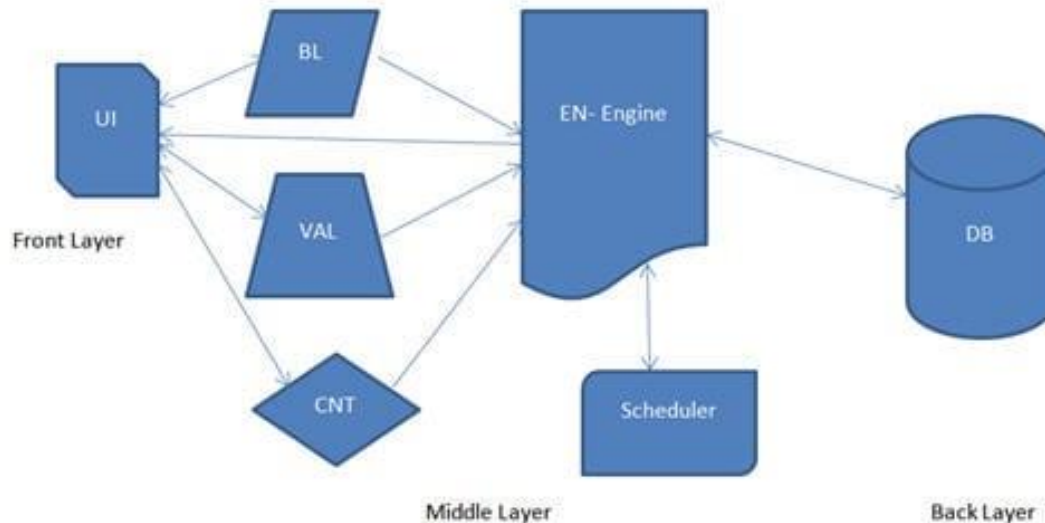


- Once the units have been written, the next stage is to put them together to create the system
 - ❑ This called integration
- The purpose of integration testing is to expose defects in the interfaces and in the interactions between integrated components or systems
- Test bases for integration testing can include
 - ❑ the software and the system design
 - ❑ a diagram of the system architecture
 - ❑ Workflows and use cases
- It is done by integration testers or test team



Example of Integration Testing

- I am the owner of an advertising company and I post ads on different websites. At the end of the month, I want to see how many people saw my ads and how many people clicked on my ads. I need a report for my ads displayed and I charge accordingly to my clients.
- GenNext software developed this product for me and below was the architecture:



Example of Integration Testing



- **UI** – User Interface module, which is visible to the end user, where all the inputs are given.
 - **BL** – Is the Business Logic module, which has all the all the calculations and business specific methods.
 - **VAL** – Is the Validation module, which has all the validations of the correctness of the input.
 - **CNT** – Is the content module which has all the static contents, specific to the inputs entered by the user. These contents are displayed in the reports.
 - **EN** – Is the Engine module, this module reads all the data that comes from BL, VAL and CNT module and extracts the SQL query and triggers it to the database.
 - **Scheduler** – Is a module which schedules all the reports based on the user selection (monthly, quarterly, semiannually & annually)
 - **DB** – Is the Database.
- Now, having seen the architecture of the entire web application, as a single unit, Integration testing, in this case, will focus on the flow of data between the modules.:

Example of Integration Testing



- In this example, the communication of data is done in an XML format. So whatever data the user enters in the UI, it gets converted into an XML format.
- In our scenario, the data entered in the UI module gets converted into XML file which is interpreted by the 3 modules BL, VAL and CNT. The EN module reads the resultant XML file generated by the 3 modules and extracts the SQL from it and queries into the database. The EN module also receives the result set and converts it into an XML file and returns it back to the UI module which converts the results in user readable form and displays it.
- In the middle we have the scheduler module which receives the result set from the EN module , creates and schedules the reports.

Example of Integration Testing



- So where Integration testing does comes into the picture?
- Well, testing whether the information/data is flowing correctly or not will be your integration testing, which in this case would be validating the XML files. Are the XML files generated correctly? Do they have the correct data? Are the data is being transferred correctly from one module to another? All these things will be tested as part of Integration testing.

System Testing



- System testing is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements
- System testing is most often the final test to verify that the system to be delivered meets the specification and its purpose
- It is usually carried out by a team that is independent of the development process
- System testing should investigate both functional and non-functional requirements of the testing
- Functional requirements provide detail on what the application being developed will do
- Non-functional system testing looks at those aspects that are important but not directly related to what functions the system performs. For example, Load Handling test , Performance test etc.

System Testing



➤ **When is it performed?**

- ❖ System Testing is the third level of software testing performed after Integration Testing and before Acceptance Testing.

➤ **Who performs it?**

- ❖ Normally, independent software testers perform System Testing.

➤ **Why System Testing?**

- ❖ #1) It is very important to complete a full test cycle and ST (System Testing) is the stage where it is done.
- ❖ #2) ST is performed in an environment that is similar to the production environment and hence stakeholders can get a good idea of the user's reaction.
- ❖ #3) It helps to minimize after-deployment troubleshooting and support calls.
- ❖ #4) In this STLC stage Application Architecture and Business requirements, both are tested.



Acceptance Testing

➤ **The next step after system testing is often acceptance testing**

- ❖ The purpose is to provide the end users with confidence that the system will function according to their expectations
- ❖ Usually done by users or customers, other stakeholders may be included

➤ **Types of acceptance testing**

- ❖ Alpha Testing: It takes place at the developer's site before release to external customers
 - ❑ Done by potential users/customers or an independent test team at the developer's site
- ❖ Beta Testing: It takes place at the customer's site. Tested by a group of customers, who use the product at their own locations
 - ❑ Provide the feedback, before the system is released. Often known as “field testing”



Categories of Test Techniques

❖ Three Categories:

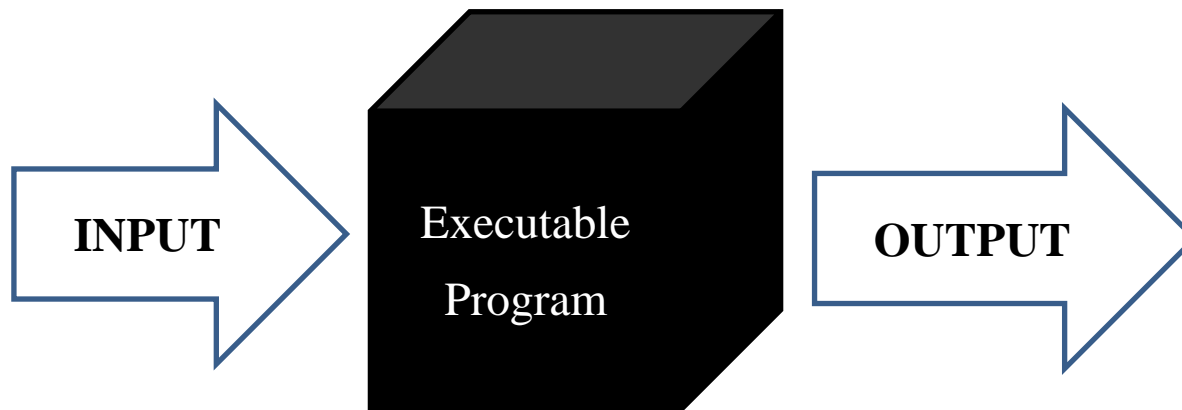
- Black-Box Testing or Specification-Based Testing
- White-Box Testing or Structure-Based Testing
- Grey –Box Testing



BLACK-BOX TESTING TECHNIQUES

- ❖ Black Box testing refers to a software testing method where the SUT (Software Under Test) functionality is tested without worrying about its details of implementation, internal path knowledge and internal code structure of the software
- ❖ Black Box testing techniques are also known as Specification-Based Techniques
- ❖ In specification-based techniques, Test Cases are derived directly from the specification or from some other kind of model of what the system should do.
- ❖ The source of information on which to base testing is known as the ‘test basis’. It includes both functional and non-functional aspects

BLACK-BOX TESTING TECHNIQUES





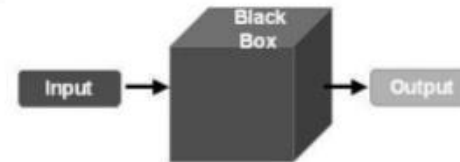
White Box Testing

- White Box testing is based on specific knowledge of the source code to define the test cases and to examine outputs
- These kinds of techniques help to identify which line of code is actually executed and which is not
- Which may indicate that there is either missing logic or a typo
- exercise a path of code
- This kind of testing is also known as Structure-based Testing or Glass-box Testing

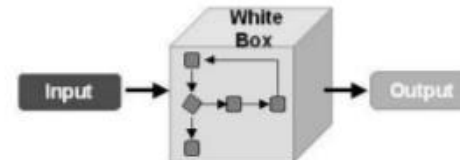
White Box Testing

White box testing Vs. Black box testing

In **Black box** testing, we test the software from a user's point of view.



In **White box** testing, we evaluate the code and internal structure of the program.



Grey Box Testing

- Gray Box Testing is a software testing method, which is a combination of both White Box Testing and Black Box Testing method.
 - ❑ In White Box testing internal structure (code) is known
 - ❑ In Black Box testing internal structure (code) is unknown
 - ❑ In Grey Box Testing internal structure (code) is partially known





Grey Box Testing

- Example of Gray Box Testing: While testing websites feature like links or orphan links, if tester encounters any problem with these links, then he can make the changes straightaway in HTML code and can check in real time.
- Why Gray Box Testing
 - ☐ It provides combined benefits of both black box testing and white box testing both
 - ☐ It combines the input of developers as well as testers and improves overall product quality
 - ☐ It reduces the overhead of long process of testing functional and non-functional types
 - ☐ It gives enough free time for a developer to fix defects
 - ☐ Testing is done from the user point of view rather than a designer point of view

Example



Welcome

Username

Password

Login

➤ Almost every web application requires its users/customers to log in. For that, every application has to have a “Login” page which has these elements:

Account/Username

Password

Login/Sign in Button

➤ For Unit Testing, the following may be the test cases:

- ☐ Field length – username and password fields.
- ☐ Input field values should be valid.
- ☐ The login button is enabled only after valid values (Format and lengthwise) are entered in both the fields.

➤ For Integration Testing, the following may be the test cases:

- ☐ The user sees the welcome message after entering valid values and pushing the login button.
- ☐ The user should be navigated to the welcome page or home page after valid entry and clicking the Login button.

Example



- Now, after unit and integration testing are done, let us see the additional test cases that are considered for functional testing:
 - ❖ The expected behavior is checked, i.e. is the user able to log in by clicking the login button after entering a valid username and password values.
 - ❖ Is there a welcome message that is to appear after a successful login?
 - ❖ Is there an error message that should appear on an invalid login?
 - ❖ Are there any stored site cookies for login fields?
 - ❖ Can an inactivated user log in?
 - ❖ Is there any 'forgot password' link for the users who have forgotten their passwords?
- There are much more such cases which come to the mind of a functional tester while performing functional testing. But a developer cannot take up all cases while building Unit and Integration test cases.
- Thus, there are a plenty of scenarios that are yet to be tested even after unit and
- integration testing.