IACSD

Part III - Software Engineering Concepts

Session 1

Developing an application in a team

Establishing a clear software development team structure is an important first step into an overall success of your project or any application, and then next is deciding the team size.

Decide on the software development team size:

when it comes to assembling the team, it all depends on the following key factors:

- Complexity of your project
- Budget
- Deadline
- Available resources

The general structure of a development team looks and includes the following roles:

- **Project manager** is a person responsible for managing and leading the whole team. Their role is to efficiently optimize the work of the team, ensure the product is meeting the requirements and identify the goals for the team.
- **Software architect** is a highly-skilled software developer that has to think through all the aspects of the project and is responsible for making high level design choices, as well as select technical standards
- **Developers or software engineers** are team members that apply their knowledge of engineering and programming languages in software development.
- **Experience designers** ensure that the product is easy and pleasant to use. They conduct user interviews, market research, and design a product with end-users in mind.
- **QA or tester** is responsible for the Quality Assurance and makes sure the product is ready to use.

- **Business Analyst's** role is to uncover the ways to improve the product. They interact with stakeholders to understand their problems and needs, and later document and analyze them to find a solution.

Issues developers face when working in a team

1. Getting used to the environment and team
2. Understanding Customers and end user
3. Coding and Programming
4. Testing and Debugging
5. Keeping up with technology
6. Managing Delivery Schedules
7. Long working hours
8. Handling Data Security Threats
9. Using understanding another person's code
10. Multi-Tasking

The following can be the solution to deal with the above issues for the software developer

- **Communication**

Communication skills are as important in a Software Developer's life as technical skills**.** Your team will judge you based on how proactive you are and how you communicate with them.
By mingling with your teammates, you can get accustomed to the workplace faster. You learn from other's experiences. This will give you an opportunity, to understand the customer and his need, in a better way. Communicate with those members of your team who have experience in handling customers.

- **Practice with Simple Target**

When it comes to difficulties faced, when you write your first code, there is no need to panic or feel defeated. Failures are stepping stones for success. Instead of writing full code, try breaking down the project into simple parts. Distribute your final program into small targets. Completing your first target is an achievement by itself. To get a more confident feel, test your programs, as soon as you finish reaching your target. Testing will throw errors. Fixing one error will guide you in fixing the next and so on. There is no harm in requesting for help when you are not able to decide how to go about testing.

- **Being Updated with New Technology**

It is always a good idea to keep in touch with newer technologies as and when they arise. Customers will also appreciate your work if they get to see something new and different.

- **Find Smart Way to Work**

It is always good to work smart and not work long to meet your delivery schedules. Hence, it is good to assess the situation and work in a smart way, then putting in too many long hours at work. Try not to get glued down to your workplace. It is good to have regulated time off from work and come back with a fresh mind and fresh ideas

- **Detain the Access**

The life of a Software developer can get easier if he/she can keep their work stations secure. There is always a chance of stealing data and project code or it misusing it. Hence, it is always good to limit the access of the data to you alone and let no one access it, apart from you, as a developer.

- **Involve Deeper Analysis**

If there is a situation, where you have to work on a working program, as a developer, take time to understand the code. Analyze what the expected end result is and then start attempting to work on the code.

Introduction to code versioning system

What is Version Control?

As the name implies, Version Control is about the management of multiple versions of a project. To manage a version, each change (addition, edition, or removal) to the files in a project must be tracked. Version Control records each change made to a file (or a group of files) and offers a way to undo or roll back each change.
For an effective Version Control, you have to use tools called Version Control Systems. They help you navigate between changes and quickly let you go back to a previous version when something isn't right.

One of the most important advantages of using Version Control is teamwork. When more than one person is contributing to a project, tracking changes becomes a nightmare, and it greatly increases the probability of overwriting another person's changes. With Version Control, multiple people can work on their copy of the project (called branches) and only merge those changes to the main project when they (or the other team members) are satisfied with the work.

Why do you need one?

Have you ever worked on a text project or on a code that requires you to recall the specific changes made to each file? If yes, how did you manage and control each version?

Following are the different approaches that were used for Version Control.

- Duplicate and rename the files with suffixes like “review,” “fixed,” or “final”?
  (But with this approach it is very easy to forget which file is which and what has changed between them)

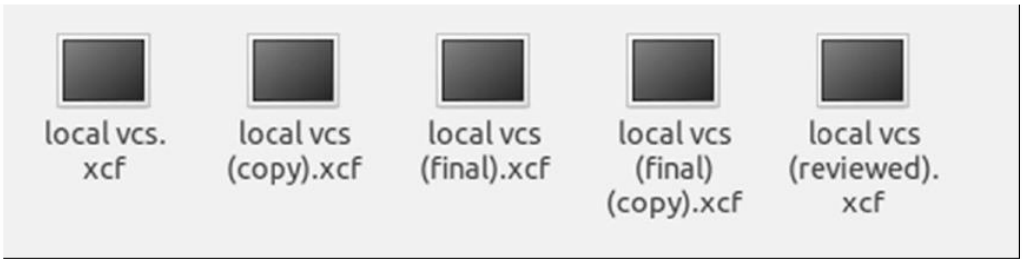


Figure 1-1. Gimp files with suffixes like “final,” “final (copy),” and “reviewed”

- To track versions, one idea is to compress the files and append timestamps to the names so that the versions are arranged by date of creation.

(But with this approach there is no way to know what are the contents and descriptions of each version.)

Figure 1-2. Compressed version files sorted by dates

- Separate History file accompanies the project folder with a short description of the change made. Also note the many compressed files which contain the previous versions of the project.

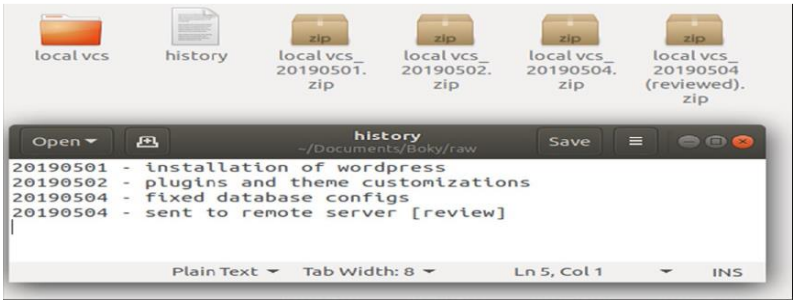


Figure 1-3. A separate file where each version is tracked

Not a perfect approach again you would still need a way to compare each version and every file change. There is no way to do this in that system; you just need to memorize everything you did. And if the project gets big, the folder just gets bigger with each version. What happens when another developer or writer joins your team? Would you email each other the files or versions you edited? Or work on the same remote folder? In the last case, how would you know who is working on which file and what changed? And lastly, have you ever felt the need to undo a change you made years ago without breaking everything in the process?

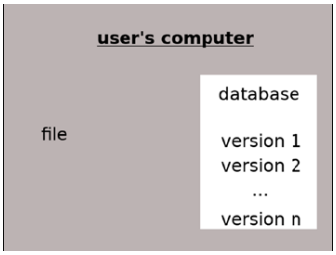
All those problems are solved by using a Version Control System or VCS.
A VCS tracks each change you made to every file of your project and provides a simple way to
Compare and roll back those changes. One popular example of VCS is Git.

Flavors of Version Control Systems

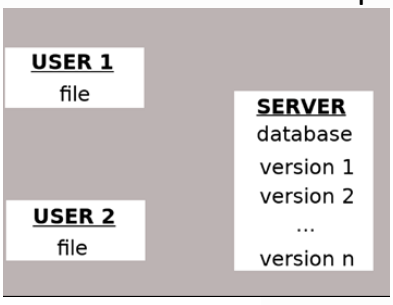
A VCS can be:

- Local
- Centralized
- Distributed.

Local VCS: everything is on the user's computer, and only one file is tracked. The versioning is stored in a database managed by the local VCS. No way to track an entire project with them.

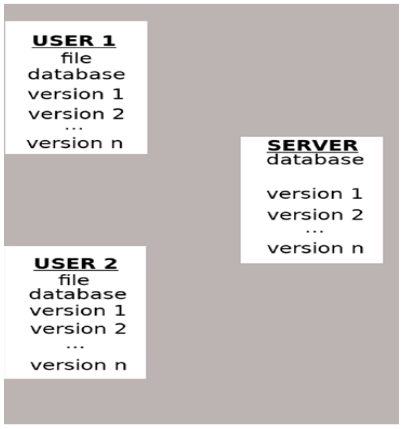


Centralized VCS: works by storing the change history on a single server that the clients (authors) can connect to. This offers a way to work with a team and also a way to monitor the general pace of a project.
The main problem is that, a server error can cost the team all their work. A network connection was also required since the main project was stored in a remote server.



Distributed VCS: works nearly the same as centralized VCS but with a big difference: there is no main server that holds all the history. Each client has a copy of the repository (along with the change history) instead of checking out a single server. This greatly lowers the chance of losing everything as each client has a clone of the project.
There is also a slight difference with how it works: instead of tracking the changes between versions, it tracks all changes as "patches." This means that those patches can be freely exchanged between repositories, so there is no "main" repository to keep up with.

IACSD



Introduction to git

What is Git?

Git is a distributed Version Control System, but it is faster and works better with large projects. The Git community is very active, and there are many contributors involved in its development.

What can Git do?

First, it works great with tracking changes. You can

- Go back and forth between versions
- Review the differences between those versions
- Check the change history of a file
- Tag a specific version for quick referencing

Git is also a great tool for teamwork. You can

- Exchange “changesets” between repositories
- Review the changes made by others

One of the main features of Git is its Branching system. A branch is a copy of a project which you can work on without messing with the repository. This concept has been around for some time, but with Git, it is way faster and more efficient. Branching also comes along with Merging, which is the act of copying the changesets done in a branch back to the source.

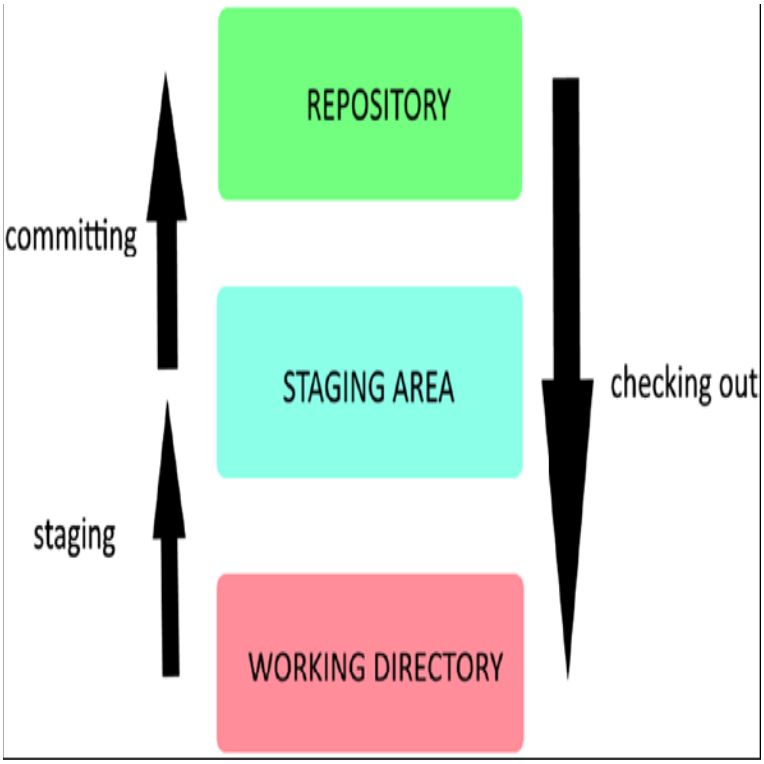
The typical Git workflow

The main feature of Git is its “Three States” system.

The states are: the working directory, the staging area, and the git directory

• The working directory is just the current snapshot that you are working on.
• The staging area is where modified files are marked in their current version, ready to be stored in the database.
• The git directory is the database where the history is stored.

So, basically Git works as follows: you modify the files, add each file you want to include in the snapshot to the staging area (git add), then take the snapshot and add them to the database (git commit). For the terminology, we call a modified file added to the staging area "staged" and a file added to the database "committed." So, a file goes
from "modified" to "staged" to "committed."



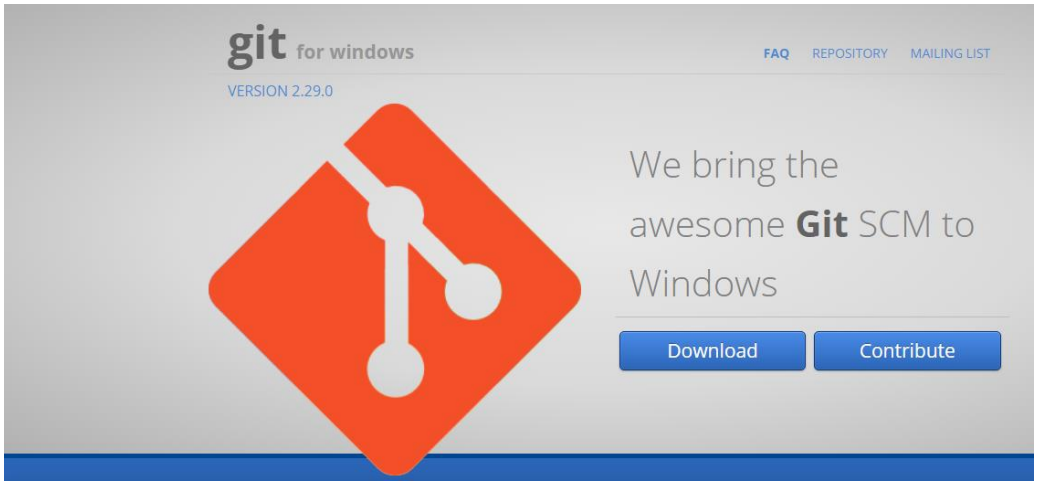
IACSD

Git Installations:

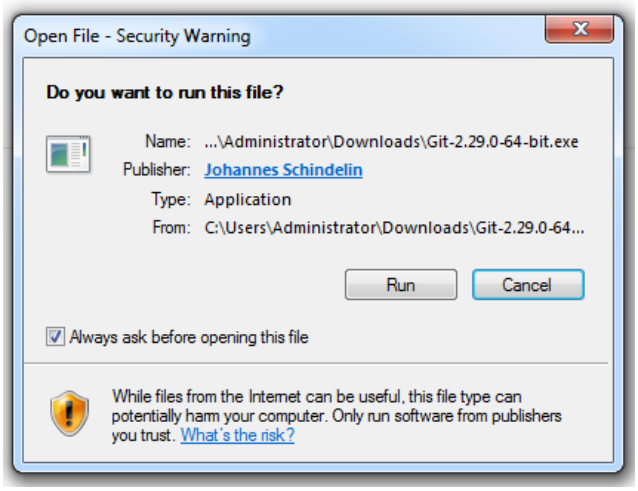
The files necessary to install Git are on https://git-scm.com/downloads for all systems.
Just follow the link and choose your Operating System.

GUI clients for Git also available there , You need to familiarize yourself with Git commands before using GUI clients

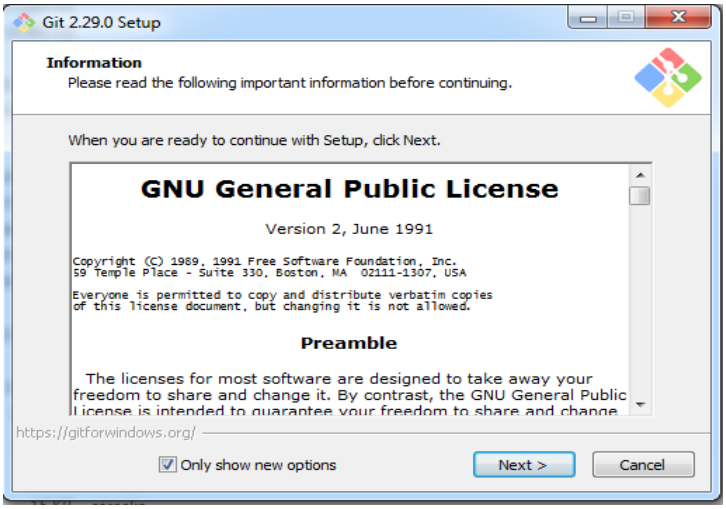
Installation and setup for Windows:

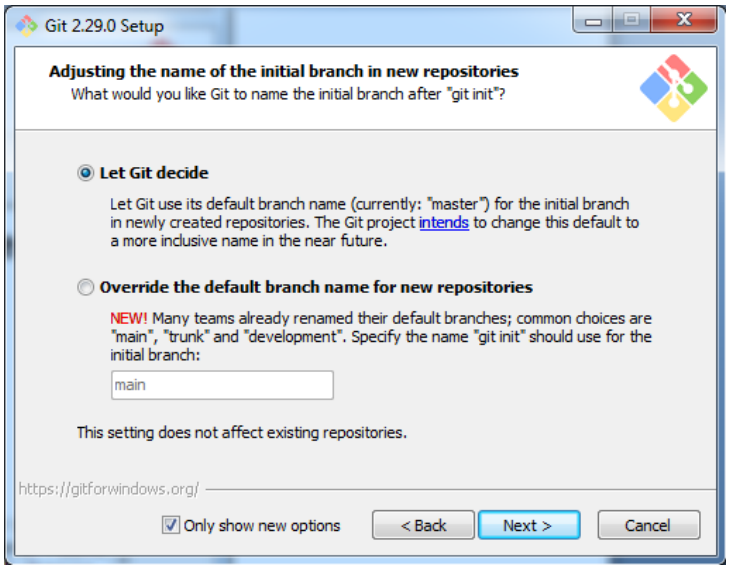


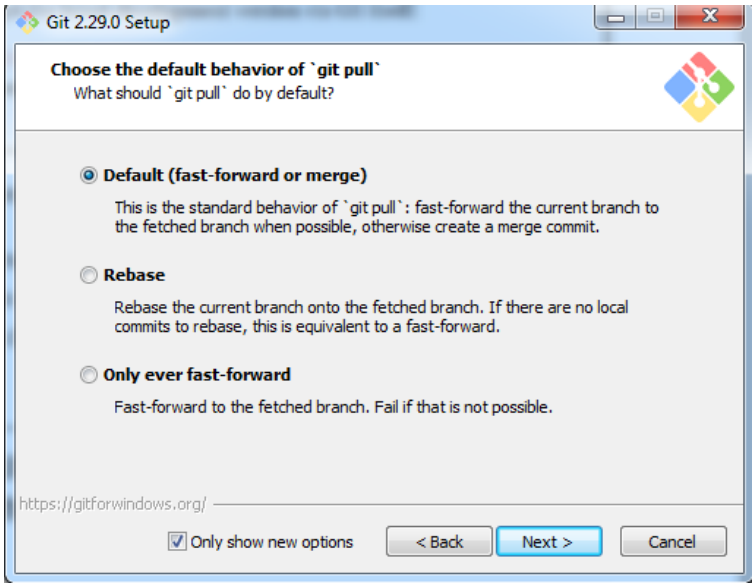
Download exe for windows

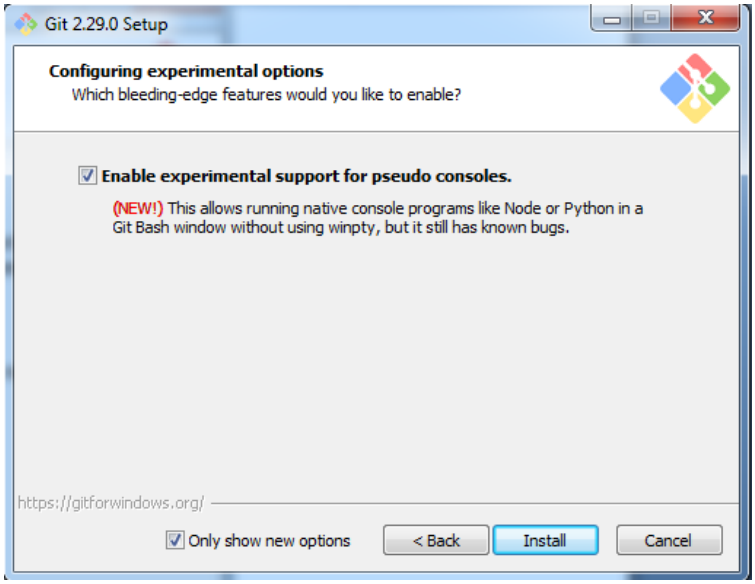


Then just simple click run and follow the simple wizards as follows:

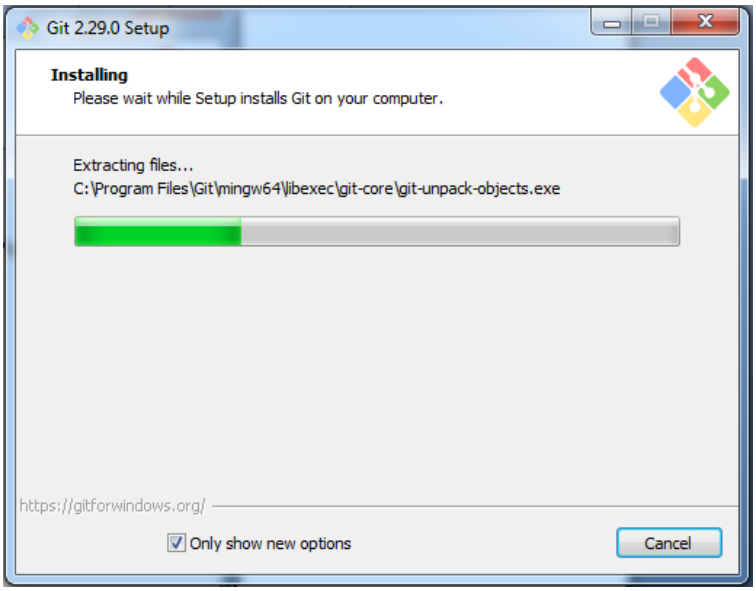




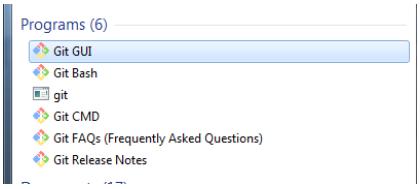

Git 2.29.0 Setup
Choose the default behavior of `git pull`
What should `git pull` do by default?
Default (fast-forward or merge)
This is the standard behavior of `git pull`: fast-forward the current branch to the fetched branch when possible, otherwise create a merge commit.
Rebase
Rebase the current branch onto the fetched branch. If there are no local commits to rebase, this is equivalent to a fast-forward.
Only ever fast-forward
Fast-forward to the fetched branch. Fail if that is not possible.
https://gitforwindows.org/
Only show new options
< Back
Next >
Cancel


Git 2.29.0 Setup
Configuring experimental options
Which bleeding-edge features would you like to enable?
Enable experimental support for pseudo consoles.
(NEW!) This allows running native console programs like Node or Python in a Git Bash window without using winpty, but it still has known bugs.
https://gitforwindows.org/
Only show new options
< Back
Install
Cancel

IACSD



Once finished you with installation you will get a shortcut icon on desktop and also the GUI You can check in program list.

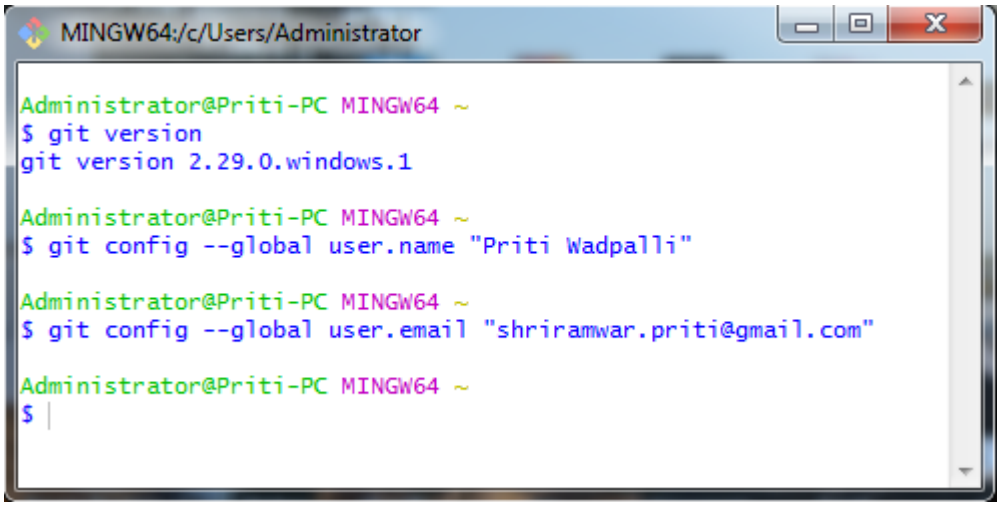


Before beginning to use Git, you need a little bit of setup first. You will probably only do this once since all the setup is stored on an external global file, meaning that all your projects will share the same configs

Since Git is a distributed Version Control System, you will one day need to connect to other remote repositories.

To set up Git, open Git Bash
In the command prompt, just tell Git your name and email address:

$ git config --global user.name "Prit Wadpalli"
$ git config --global user.email "shriramwar.priti@gmail.com "



Notice the “global” argument; it means that the setup is for all future Git repositories, so you don’t have to set this up again in the future.

Introduction to git repository and git structure

Repositories

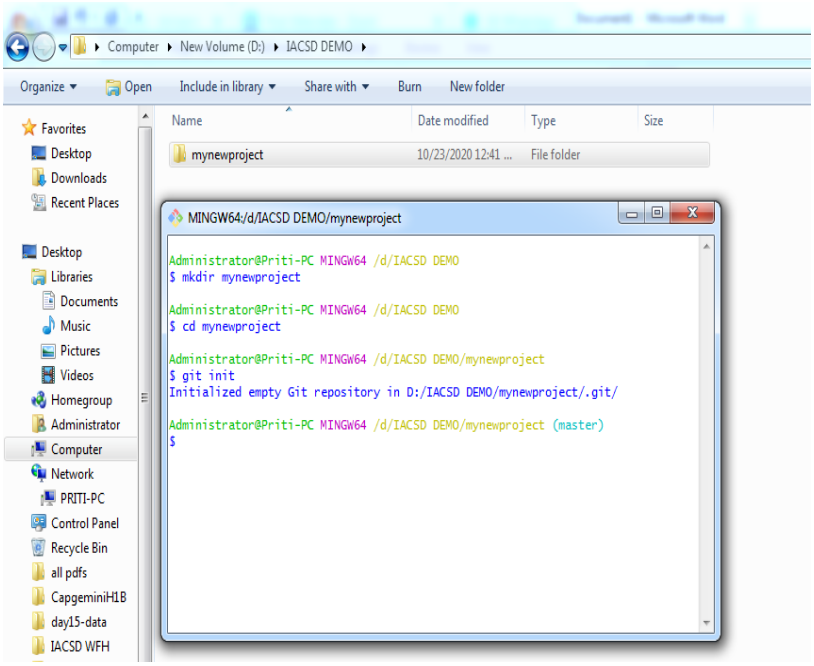
A repository is a storage where all your project and all the changes made to it are kept. You can think of it as a “change database.” it is only a normal folder on your system, so it is very easy to manipulate.

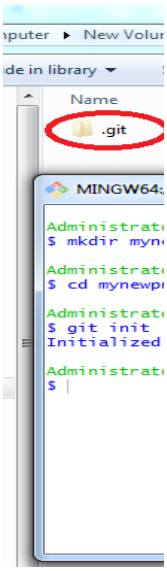
For each project you want to manage with Git, you have to set up a repository for it. Setting up a repository is very easy. Just navigate to the folder you want to track and tell Git to initiate a repository there.

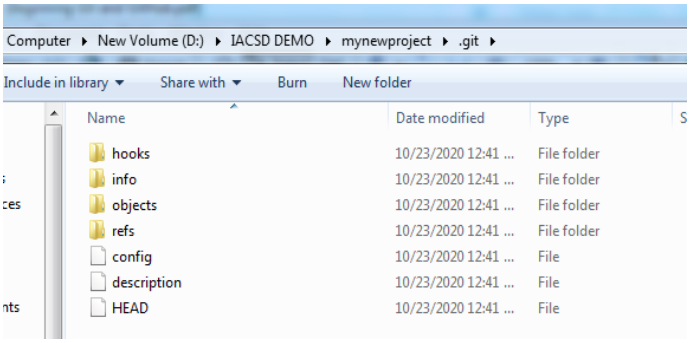
So for each project you want to start, you should

- Create the directory containing your project
- Navigate into the directory
- Initialize a Git repository

To get started right click in your workspace where you want to keep your projects and say git bash here
And then follow the commands on git bash
$ mkdir mynewproject
$ cd mynewproject/
$ git init







Git will create a directory called ".git" that will contain all your changesets and snapshots.

all those snapshots are stored in the ".git" directory. Each snapshot is called "commit," and we'll look into that shortly after this section.The HEAD file in this ".git" directory points to the current "branch" or subversion of the project that you are working on. The default branch is called "master," You should also know that initializing is the only way to get a repository. You can copy an entire repository with all its history and snapshots. It is called "cloning,"

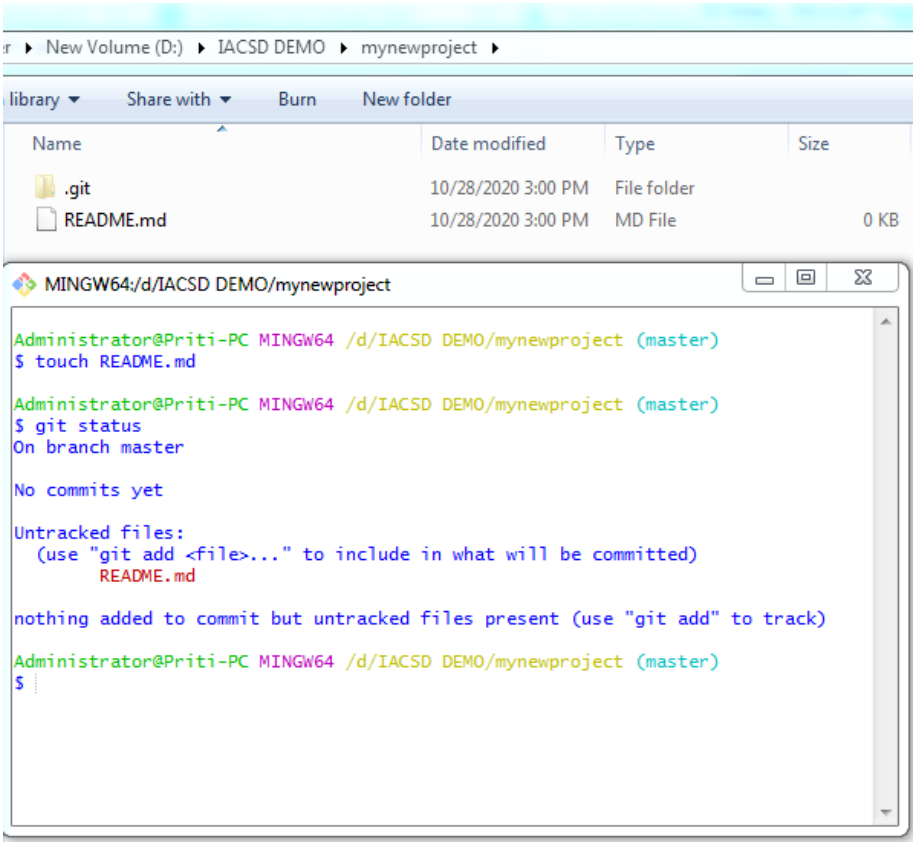
Working Directory

What about the empty area outside the ".git" directory? It is called the Working Directory, and the files you will be working on will be stored there. Generally, your most recent version will be on the Working Directory. Each file you work on is on the Working Directory. There is nothing particular about this place except the fact that you will only manipulate the files here directly. Never modify the files inside the ".git" directory! Git will detect any new file you will place in the Working Directory. And you check the status of the directory by using the Git command "status."

$ git status

For example, if we create a new file called README.md in the Working Directory, we will see that Git will know that the project has changed

Figure: *The status of the Working Directory*



So essentially, that is the Working Directory: the area where you directly interact with your project files.

Adding code to git

Staging Area

The Staging Area is where your files go before the snapshots are taken. Not every file you modified on the Working Directory should be taken into account when taking a snapshot of the current state of the project. Only the files placed in the Staging Area will be snapshotted.

So, before taking a snapshot of the project, you select which changed files to take account of. A change in a file can be creating, deleting, or editing. Think of it as designating which files get to be in the family photo. To add a file to the Staging Area, we use the Git command “add.”

$ git add nameofthefile

It's that simple. If we wanted to stage the README.md that we created earlier, we would use "git add README.md." Or if you created multiple files, you can add them one after another or together like "git add file1 file2 file3." Let's stage our new file by using the command:

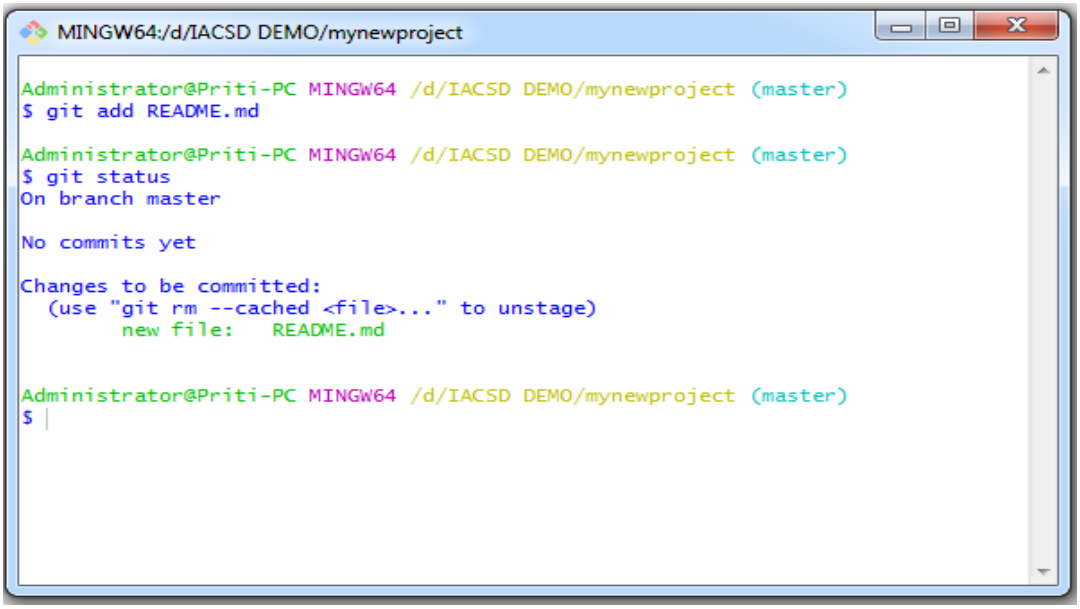
$ git add README.md

Then let's check the status with git status command.

$ git status

Adding a file to the staging area won't produce any visible result, but checking the status will get you a result similar to Figure

Figure: *Staging a file*



you will notice that after staging the file, the Working Directory is clean again. That's because "git status" only keeps track on "unstaged" files.

you can unstage a file using the Git command "git rm" with the option "--cached."

$ git rm --cached README.md

Don't forget the option "--cached" when unstaging a file. If you forget it,

you could lose your file!

After you stage all the files that you want the changes to be taken into account, you are now ready to take your first snapshot!

Commits

A commit is just a snapshot of the entire project at a certain time. Git doesn't record the individual changes done to the files; it takes a picture of the entire project. In addition to the snapshot, a commit also contains information about the "author" of the content and the "commiter" or who put the changeset into the repository.

(Note : "author" and "commiter" are usually the same person, unless the commiter took the changeset from another team member. Remember that Git commits are exchangeable since it is a distributed VCS)

Since a commit is a snapshot from the state of the project, the previous state of the project is another commit called "parent." The very first commit is created by Git when the repository is created, and it's the one commit that has no parents. All future commits are then linked to each other via parentage. The ensemble of those commits that are parents to each other is called "branch."

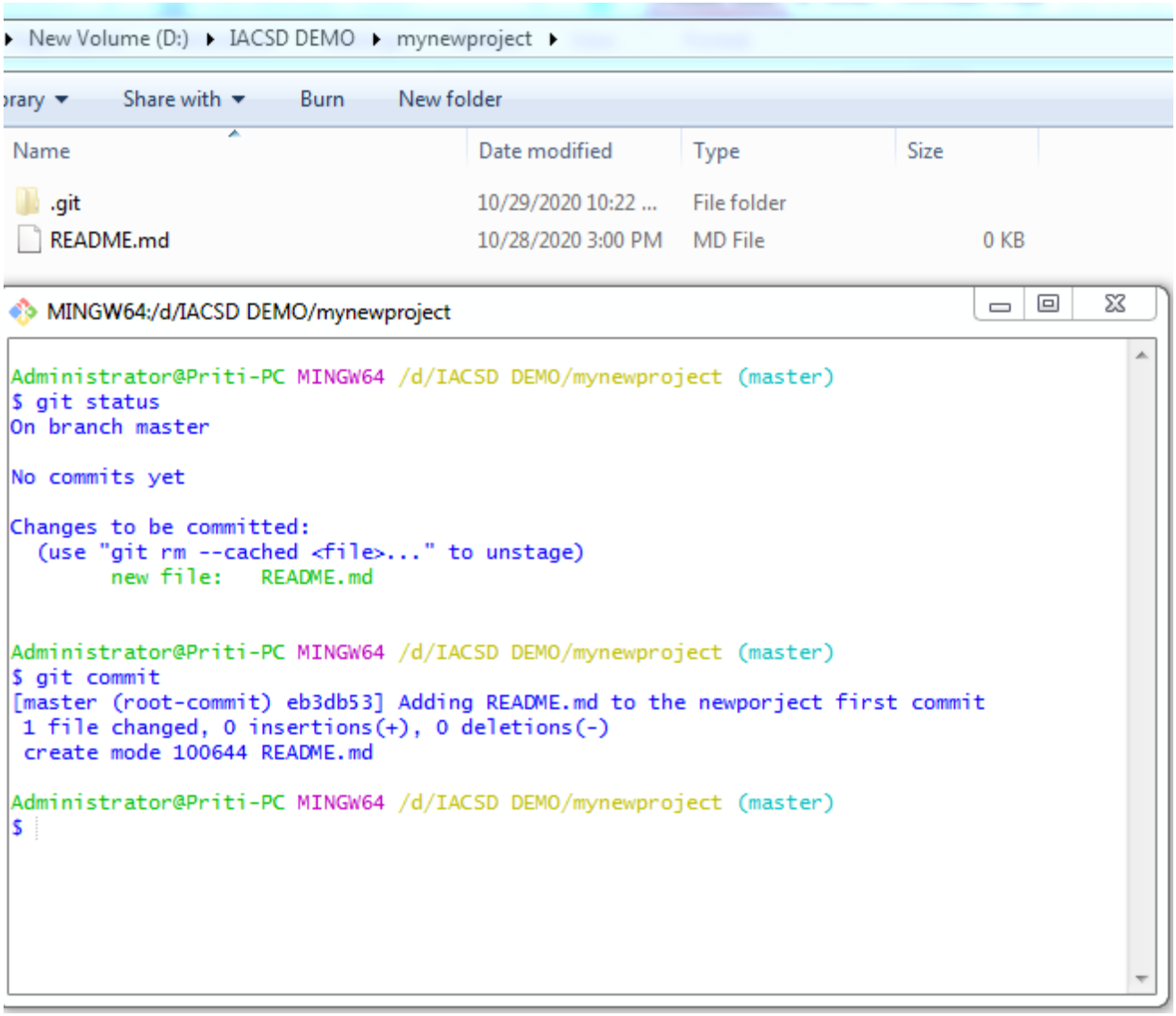
(Note If a commit has two parents, that means that it was created by merging two branches.)

A reference to a specific commit is called "head," and it also has a name. And the head you are currently working on is called "HEAD" (see the previous section).We can now commit the files we staged earlier. Before each commit, you should check the status of the Working Directory and the Staging Area. If all the files you want to commit are in the Staging Area (under the phrase "Changes to be committed"), you can commit. If not, you have to stage them with "git add." To commit all the changes we made, we use "git commit." This will take a snapshot of our current state of the project.

$ git commit

If we execute this command, it will open our default editor and ask us for a commit message. A commit message is a short description of what has changed in the commit compared to the previous one.

IACSD



After commit if you check

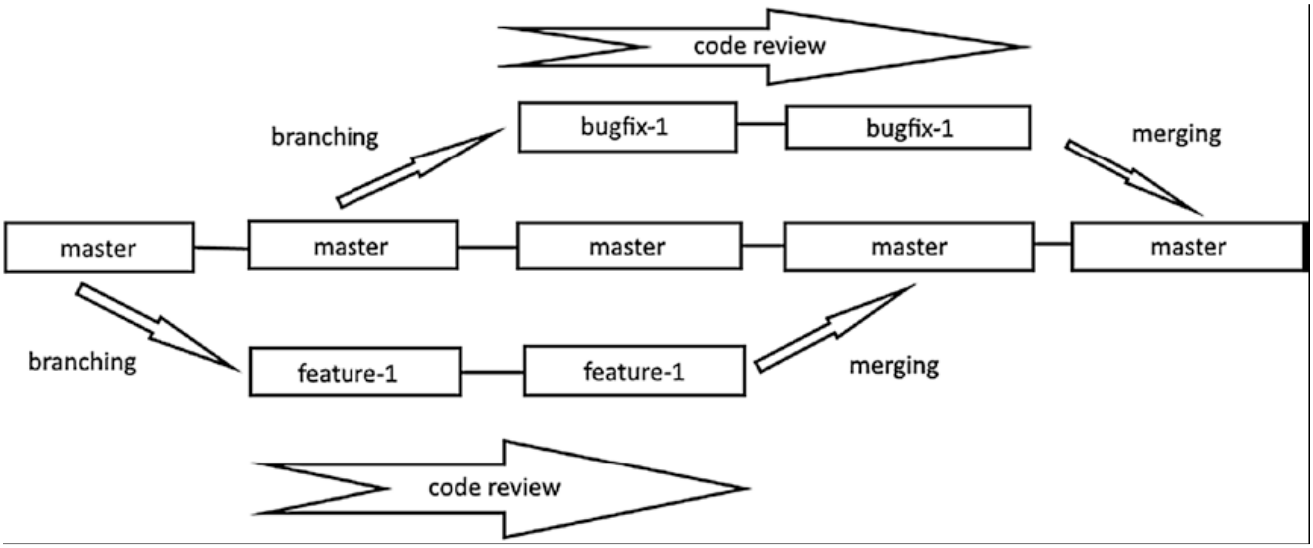
$ git status

```
Administrator@Priti-PC MINGW64 /d/IACSD DEMO/mynewproject (master)
$ git status
On branch master
nothing to commit, working tree clean
```

IACSD

Creating and merging different git branches

As you can see, we can create branch from any branch in our project. Git created a branch called master for us at the initialization of the repository. We then can create more branches (e.g. a bugfix branch or a feature branch) to introduce changes in the master branch.



Branches

As we said earlier, branches are the main feature behind code reviews. You have to work on your own branch before publishing your work, so that it won't be bothered by other people's changes. Put simply, a branch is just your own independent copy of the project at a certain time.

The logic behind branches is simple: take the current state of the project and make a copy of it. In this copy, you can make your changes without impacting other people. You can use branches to have distinct channels of distributions or just to try new things with the project.


IACSD

When creating a repository, you get a branch by default: master. When working on very small projects, this branch is enough; but most projects need more branches to get the best results.

- First, they need a **production branch**, where clients can get the last stable version of the software; this is the master branch. The production branch is only updated when the project is sure to be stable as this is the release branch.
- Then, there is the **development branch**, where all the progress is recorded and all the commits tested. You will mostly work on the development branch as it is where most of the fun is.
- Finally there are the short-lived **patching branches** which you will create to hold your commits before merging them to the development branch. Those patching branches live and die with a pull request; you create one when you are solving an issue and delete it afterward.

(Summary Note:
• Production branch, where you will release stable versions of your project
• Development branch, where you will test your latest version
• Patching branch, where you will work on your issues)

Creating a branch

You just need to use the "git branch" command followed by the branch name. Keep in mind that the
branch name should only contain alphanumeric values and dashes or underscores; no spaces allowed.

$ git branch <name>

let's create a development branch for our project. Let's name it "develop."

After you execute that command, you will notice that nothing has changed in your project. That's because creating a branch is just about creating a reference to the last commit of the current branch and nothing else. To begin working with a branch, you have to switch to it

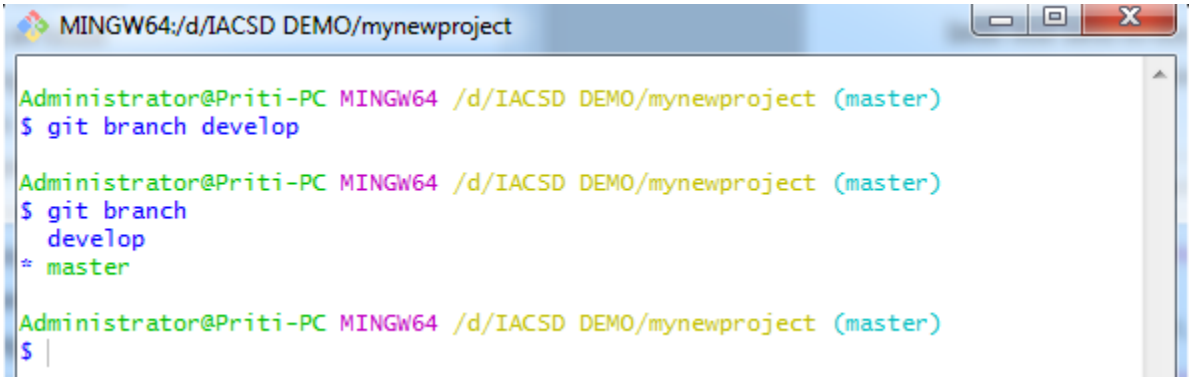


Figure: List of branches in our project

Switching to another branch

To work on a particular branch you need to switch into it first. To check the list of available branches in the project use git branch command without any parameter .

$ git branch

This command will give you the list branches you currently have and will put a little star next to the one you're currently on (the HEAD). Check out above figure.

You will notice that we still are on the master branch because we haven't made anything other than creating a branch.
Simply use "git checkout" with the name of the branch as parameter.

$ git checkout <name>

So, if we want to switch to the develop branch, we will have to execute:

$ git checkout develop

After checking out the new branch, you will get a confirmation message from Git and you can also check the result of git status to make sure. Figure 11-3 shows the result of those commands.

```
Administrator@Priti-PC MINGW64 /d/IACSD DEMO/mynewproject (master)
$ git checkout develop
Switched to branch 'develop'

Administrator@Priti-PC MINGW64 /d/IACSD DEMO/mynewproject (develop)
$ git status
On branch develop
nothing to commit, working tree clean

Administrator@Priti-PC MINGW64 /d/IACSD DEMO/mynewproject (develop)
$
```

(Note : Like when we navigated between versions, you can't switch branches if you have uncommitted changed files. Commit before you move. Or use a technique called "stashing" that we will see in later chapters.)

Deleting a branch

To delete a branch, simply use the same command as to create one but with the option "-d."

$ git branch -d <name>

So, to delete our testing branch, we will use

$ git branch -d testing

Just like a real tree branch, you don't cut the Git branch you are currently standing on. Check out another branch before deleting the branch; and for this reason, you can't have less than one branch in a project.

```
Administrator@Priti-PC MINGW64 /d/IACSD DEMO/mynewproject (develop)
$ git checkout master
Switched to branch 'master'

Administrator@Priti-PC MINGW64 /d/IACSD DEMO/mynewproject (master)
$ git branch -d develop
Deleted branch develop (was eb3db53).

Administrator@Priti-PC MINGW64 /d/IACSD DEMO/mynewproject (master)
$
```

If you try to delete a current branch you will get an error like this

```
Administrator@Priti-PC MINGW64 /d/IACSD DEMO/mynewproject (develop)
$ git branch -d develop
error: Cannot delete branch 'develop' checked out at 'D:/IACSD DEMO/mynewproject'

Administrator@Priti-PC MINGW64 /d/IACSD DEMO/mynewproject (develop)
$
```

So always checkout to some other branch before you delete the current one.

Merging branches

Let's imagine that you want to improve the README file of the project by adding a few information. This The next step is to create a new branch from the development branch so we can merge them later. You have to create a new branch from the develop branch instead of the master because we won't touch the master branch until everything is properly tested. If everything is clear and clean, we will merge the development branch into the master branch.

Ex for Merger:

create the new branch where we will work on. Let's name it "improve-readme-description." Don't forget to checkout out the develop branch before creating a new branch from it.

$ git checkout develop

$ git branch improve-readme-description

Now that the branch has been created, switch to it so we can begin to work.

$ git checkout improve-readme-description

Open the README.md file and change its content to

```
# TODO list
A simple app to manage your daily tasks.
It uses HTML5 and CSS3.
## Features
* List of daily tasks
```

Now, stage the file and get ready to commit.

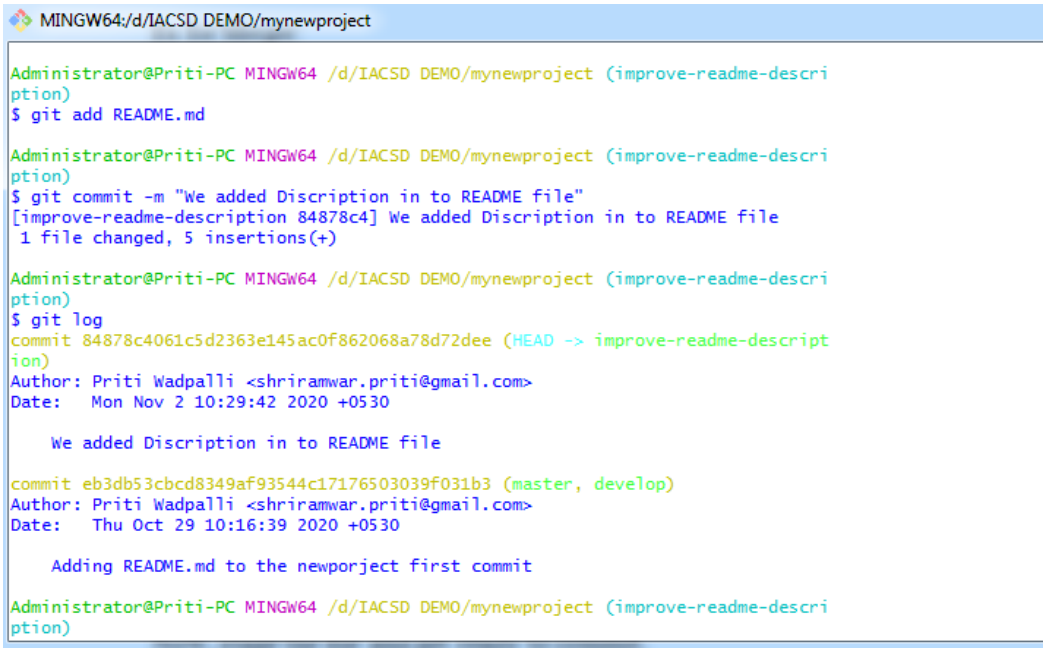
$ git add README.md

$ git commit

After you made the commit, check the Git history to put all of we did in perspective.
Execute the git log command to see our project history.

$ git log

As you can see in the figure, HEAD now points to the last commit of our new branch; it means that every commit we will create will have that as a parent. You will also notice that the master and develop branch didn't change; that's because we only worked on our newly created branch.



Now that we are satisfied with our fix, let's merge the branch to the develop branch so we can test it. To merge our branch into develop, we first have to check it out. So, navigate there by using the git checkout command.

$ git checkout develop

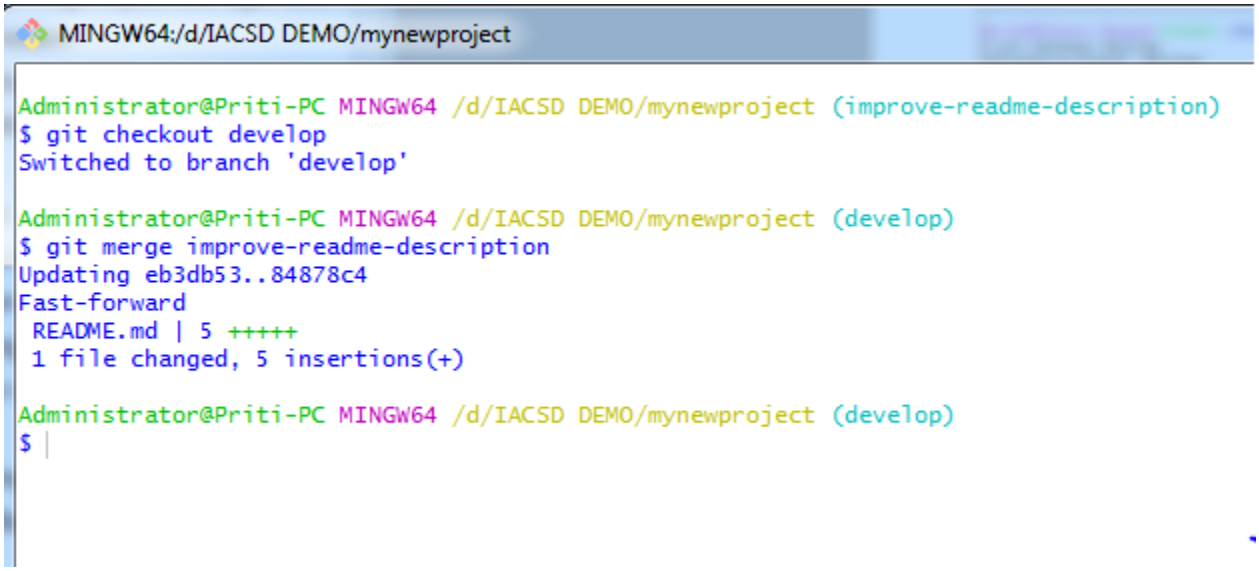
Now let's try to merge the branch into the develop one. Merging just means reproducing all the commits on one branch on another. To do so, we will use the git merge command followed by the name of the branch be merged.

$ git merge <name>

Since we are looking to merge "improve-readme-description" into "develop," our command to execute on the develop branch is

$ git merge improve-readme-description

This command will recreate your commits from “improve-readme-description” into “develop.”



Let’s recheck the git log to have a clearer idea of what happened. As you can see, HEAD now points to develop because it’s the checked-out branch. You can also notice that develop and improve-readme-description now point to the same commit; that’s because of the merge.

IACSD

Session 2

Introduction to software engineering

What is Software Engineering?

The application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is, the application of engineering to software. (IEEE (Institute of Electrical and Electronics Engineers.)Standard Computer Dictionary, 610.12, ISBN 1-55937-079-3, 1990)

In Simple Words

- Software Engineering It is nothing but a set of best practices
- Where so best practices come from? From experience of excellent people
- Helps fresher or less experienced people perform much better like excellent experienced people

Software Programming ≠ Software Engineering

Software programming i.e. What we did in academics: the process of translating a problem from its physical environment into a language that a computer can understand and obey.

(Webster's New World Dictionary of Computer Terms) say Programming is having following features

- Single developer
- "Toy" applications
- Short lifespan
- Single or few stakeholders
- Architect = Developer = Manager = Tester = Customer = User
- One-of-a-kind systems
- Built from scratch
- Minimal maintenance

Whereas Software engineering i.e. what we need to do in Industry has following features

- Teams of developers with multiple roles
- Complex systems
- Indefinite lifespan

- Numerous stakeholders
- Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠ User
- Maintenance accounts for over 60% of overall costs

Three key Challenges

Software engineering in the 21st century faces three key challenges:

- **Legacy systems**

Old, valuable systems must be maintained and updated

- **Heterogeneity**

Systems are distributed and include a mix of hardware and software

- **Delivery**

There is increasing pressure for faster delivery of software

Software Myths from Different Perspectives

Understanding the myths of different people involved in software development process is important

In order to evolve into a solutions of these myths

Software Myths (Customer Perspectives)

- A general statement of objectives is sufficient to get started with the development of software. Missing/vague requirements can easily be incorporated/detailed out as they get concretized.
  Reality: Application requirements can never be stable; software can be and has to be made flexible enough to allow changes to be incorporated as they happen.

Software Myths (Developer Perspectives)

- Once the software is (Designed, Developed, Tested and then) deployed, the job is done.
  Reality: Usually, the problems just begin!
- Until the software is coded and is available for testing, there is no way for assessing its quality.
  Reality: Usually, there are too many tiny bugs inserted at every stage that grow in size and complexity as they progress thru further stages!
- The only deliverable for a software development project is the tested code.

Reality: The code is only the externally visible component of the entire software complement!

Software Myths (Management Perspectives)

- As long as there are good standards and clear procedures in my company, I shouldn't be too concerned.
  Reality: However, they are frequently ignored by developers because they are irrelevant and incomplete, and sometimes incomprehensible.
- As long as my software engineers have access to the fastest and the most sophisticated computer environments and state-of-the art software tools, I shouldn't be too concerned.
  Reality: Tools may help, but there is no magic. Problem solving requires more than tools, it requires great understanding.
- When my schedule slips, what I have to do is to start a fire-fighting operation: add more software specialists, those with higher skills and longer experience - they will bring the schedule back on the rails!
  Reality: Unfortunately, increasing team size increases communication overhead. New workers must learn project details taking up the time of those who are already immersed in the project.
  ``adding people to a late project makes it later.''

Unique Characteristics of Software

- Software is malleable (flexible)
- Software construction is human-intensive
- Software is intangible and hard to measure
- Software problems are usually complex
- Software directly depends upon the hardware
- Software doesn't wear out but will deteriorate
- So Software solutions require unusual thoroughness

Importance of Software Engineering

- **1. Reduces complexity**

- **2. To minimize software cost**
- **3. To decrease time**
- **4. Handling big projects**
- **5. Reliable software**
- **6. Effeteness**

Software Engineering is very important, as it is the backbone of all software systems. It links technologies and practices.

Software Development Life Cycle SDLC

The entire SDLC process divided into the following stages:

- Phase 1: Requirement collection and analysis
- Phase 2: Feasibility study:
- Phase 3: Design:
- Phase 4: Coding:
- Phase 5: Testing:
- Phase 6: Installation/Deployment:
- Phase 7: Maintenance:

Phase 1: Requirement collection and analysis:

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognition of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

Phase 2: Feasibility study:

Once the requirement analysis phase is completed the next step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

There are mainly five types of feasibilities checks:

- **Economic:** Can we complete the project within the budget or not?
- **Legal:** Can we handle this project as cyber law and other regulatory framework/compliances?
- **Operation feasibility:** Can we create operations which is expected by the client?
- **Technical:** Need to check whether the current computer system can support the software
- **Schedule:** Decide that the project can be completed within the given schedule or not.

Phase 3: Design:

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design (LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

Phase 4: Coding:

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

Phase 5: Testing:

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Phase 6: Installation/Deployment:

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

Phase 7: Maintenance:

Once the system is deployed, and customers start using the developed system, following 3 activities occur

- Bug fixing - bugs are reported because of some scenarios which are not tested at all
- Upgrade - Upgrading the application to the newer versions of the Software
- Enhancement - Adding some new features into the existing software

The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

Requirements Engineering

What is the Requirements Analysis?

- It is the process of defining the expectations of the users for an application that is to be built or modified.
- Identify the needs of different stakeholders.
- Requirements analysis means to analyze, document, validate and manage software or system requirements.
- High-quality requirement analysis helps to identify business opportunities, and are defined to a facilitate system design.

Requirements analysis process

- **Eliciting requirements**

The process of gathering requirements by communicating with the customers is known as eliciting requirements.

- **Analyzing requirements**

This step helps to determine the quality of the requirements. It involves identifying whether the requirements are unclear, incomplete, ambiguous, and contradictory. These issues resolved before moving to the next step.

- **Requirements modeling**

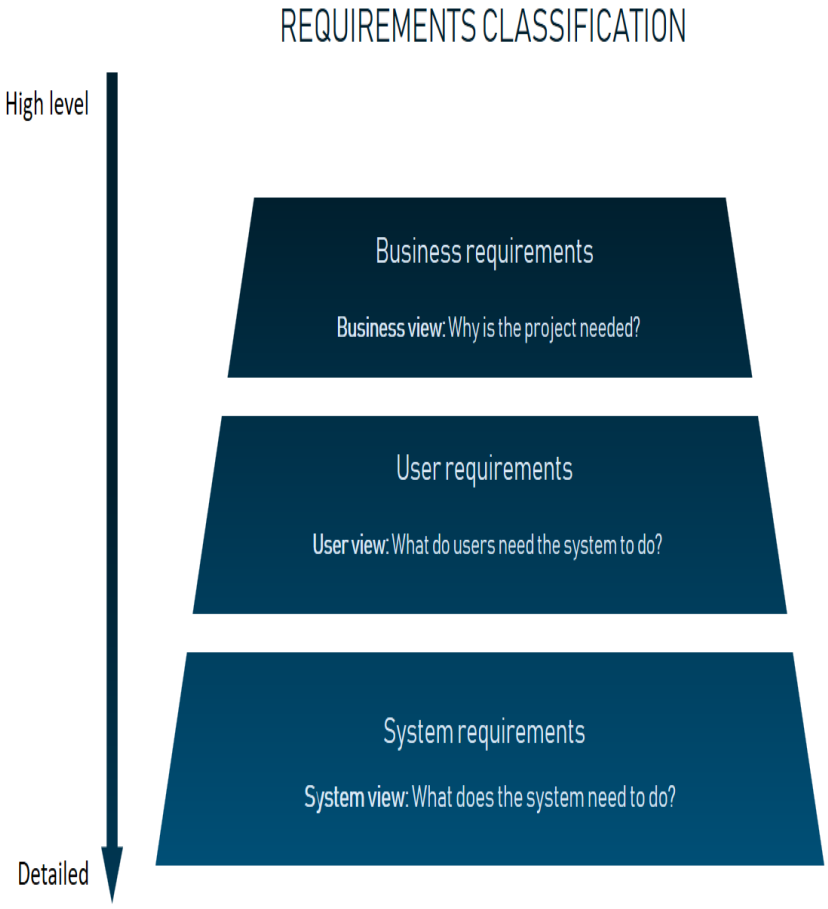
In Requirements modeling, the requirements are usually documented in different formats such as:

1. **Use cases,**
2. **User stories,**
3. **Natural-language documents,**

4. Process specification.

- **Review and retrospective**

This step is conducted to reflect on the previous iterations of requirements gathering in a bid to make improvements in the process going forward.



Types of Requirements

- Business **Requirements** (BR)
- Market **Requirements** (MR)
- Functional **Requirements** (FR)
- Non-Functional **Requirements** (NFR)
- UI **Requirements** (UIR)

- **Business requirements.** These include high-level statements of goals, objectives, and needs.
- **Stakeholder requirements.** The needs of discrete stakeholder groups are also specified to define what they expect from a particular solution.
- **Nonfunctional** requirements describe the general characteristics of a system. They are also known as ***quality attributes***.
- **Functional** requirements describe how a product must behave, what its features and functions.

Functional Requirements:

- These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract
- Ex: in a hospital management system, a doctor should be able to retrieve the information of his patients.

Non-functional requirements:

- These are basically the quality constraints that the system must satisfy according to the project contract.
- Ex They basically deal with issues like:
  Portability, Security, Maintainability, Reliability, Scalability,
  Performance Reusability Flexibility

Once the requirements are gathered, we document the requirements in a Software Requirements Specification (**SRS**) document**, use cases** or as **User Stories**, which are shared with the stakeholders for approval. This document is **easy to understand** for both **normal users** and **developers**.

User Stories Template

IACSD

Sample User Story

Title:	Customer Inter Account Transfer
Value Statement:	As a bank customer, I want to transfer funds between my linked accounts, So that I can fund my credit card.
Acceptance Criteria:	*Acceptance Criterion 1:* Given that the account is has sufficient funds When the customer requests an inter account transfer Then ensure the source account is debited AND the target account is credited. *Acceptance Criterion 2:* Given that the account is overdrawn, When the customer requests an inter account transfer Then ensure the rejection message is displayed And ensure the money is not transferred.
Definition of Done:	• Unit Tests Passed • Acceptance Criteria Met • Code Reviewed • Functional Tests Passed • Non-Functional Requirements Met • Product Owner Accepts User Story
Owner:	MR I Owner
Iteration:	Unscheduled
Estimate:	5 Points

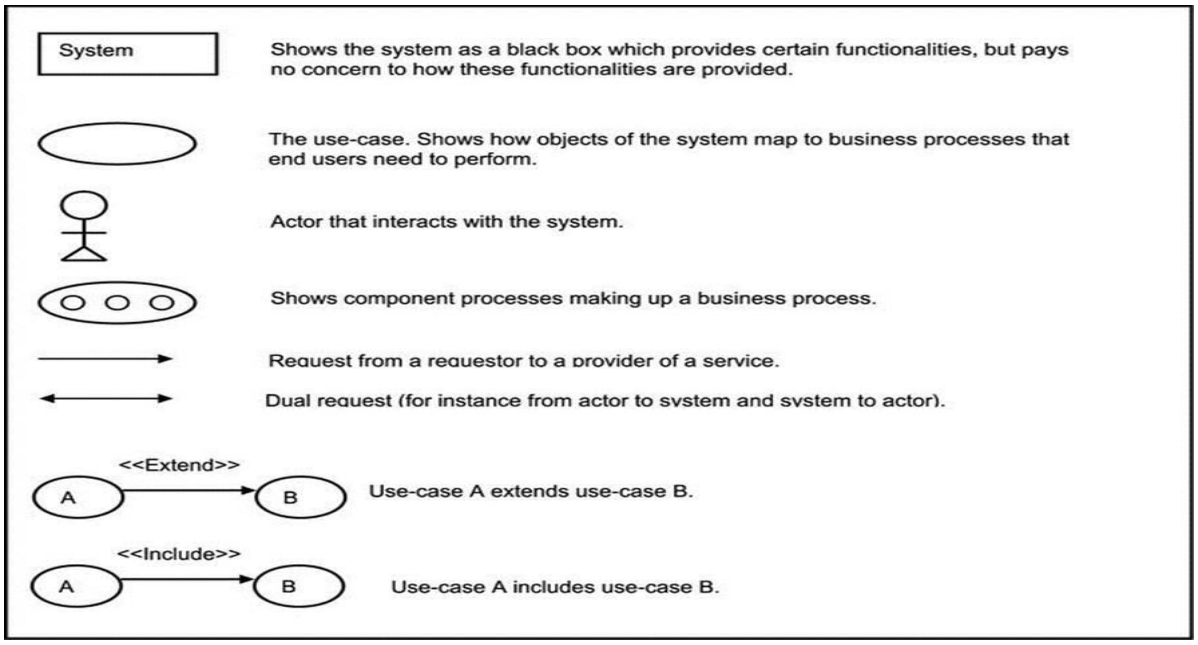
Use Case Approach

- It's a methodology **used** in system analysis to identify, clarify, and organize system requirements.
- The **use case** is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.
- A **use case specification** represents the sequence of events along with other information that relates to this use case.
- A typical use case specification template includes the following information:
  - Description
  - Pre- and Post- interaction condition
  - Basic interaction path
  - Alternative path
  - Exception path

Use Case Diagrams:

A use case diagram contains four components.

- The **boundary**, which defines the system of interest in relation to the world around it.
- The **actors**, usually individuals involved with the system defined according to their roles.
- The **use cases**, which the specific roles are played by the actors within and around the system.
- The **relationships** between and among the actors and the use cases.



Note:

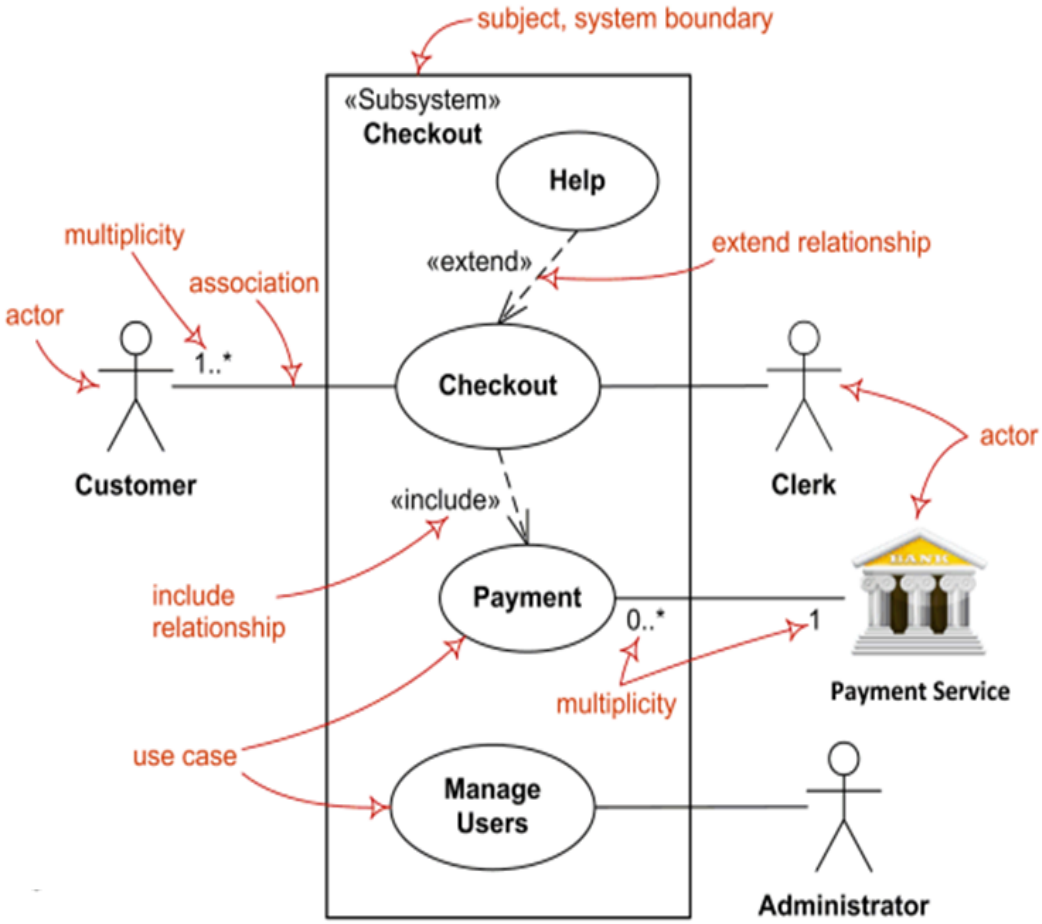
extend:

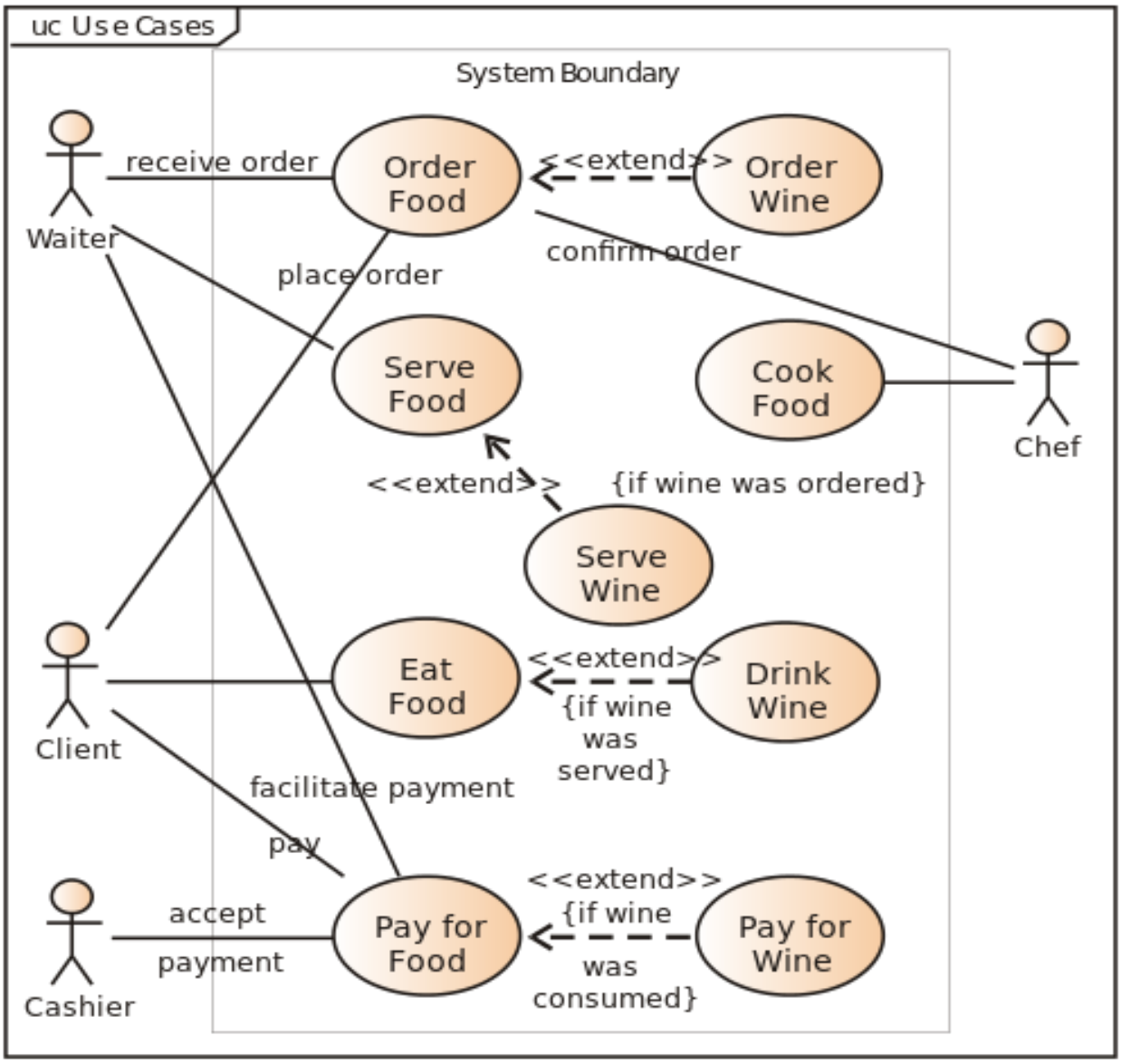
- The extending use case is dependent on the base use case; it literally extends the behavior described by the base use case. The base use case should be a fully functional use case in its own right

include

- A base use case is dependent on the included use case(s); without it/them the base use case is incomplete as the included use case(s) represent sub-sequences of the interaction that may happen always OR sometimes

Sample Diagrams:


subject, system boundary
«Subsystem»
Checkout
Help
«extend»
extend relationship
multiplicity
association
actor
1..*
Checkout
Customer
Clerk
actor
«include»
include
relationship
Payment
0..*
1
Payment Service
multiplicity
use case
Manage
Users
Administrator


uc Use Cases
System Boundary
Waiter
receive order
Order Food
<<extend>>
Order Wine
confirm order
place order
Serve Food
Cook Food
Chef
<<extend>>
{if wine was ordered}
Serve Wine
Client
Eat Food
<<extend>>
Drink Wine
{if wine was served}
facilitate payment
pay
Cashier
accept payment
Pay for Food
<<extend>>
{if wine was consumed}
Pay for Wine

Usage Scenarios

The basic strategy is to identify a path though a use case, or through a portion of a use case, and then write the scenario as an instance of that path.
Ex:

- **Scenario: A successful withdrawal attempt at an automated teller machine (ATM).**
  John Smith presses the "Withdraw Funds" button
- The ATM displays the preset withdrawal amounts ($20, $40, and so on)
- John chooses the option to specify the amount of the withdrawal
- The ATM displays an input field for the withdrawal amount
- John indicates that he wishes to withdraw $50 dollars
- The ATM displays a list of John's accounts, a checking and two savings accounts
- John chooses his checking account
- The ATM verifies that the amount may be withdrawn from his account
- The ATM verifies that there is at least $50 available to be disbursed from the machine
- The ATM debits John's account by $50
- The ATM disburses $50 in cash
- The ATM displays the "Do you wish to print a receipt" options
- John indicates "Yes"
- The ATM prints the receipt

Benefits of Use Cases

- **Use case** help to capture the functional requirements of a system.
- **Use cases** are traceable.
- **Use cases** can serve as the basis for the estimating, scheduling, and validating effort.

IACSD

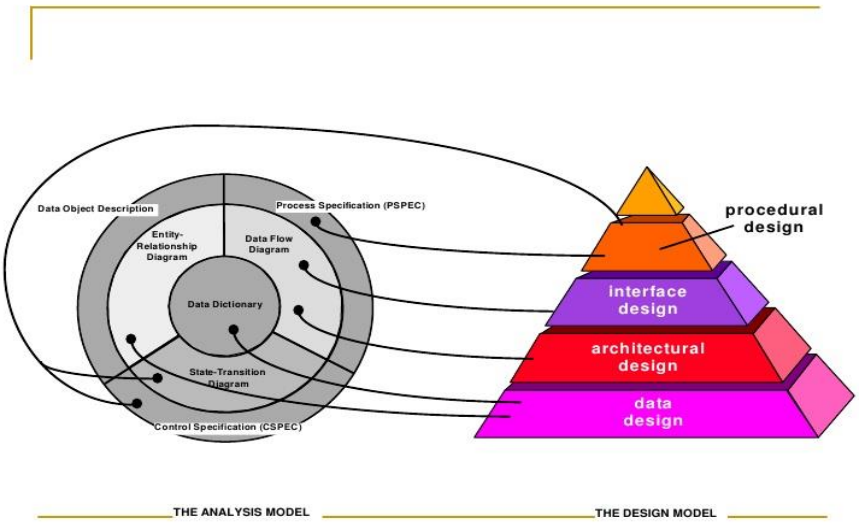
Design and Architectural Engineering

Design process

Aim of design engineering is to generate a model which shows firmness
Software design is an iterative process through which requirements are translated into the blueprint for building the software.

Process of Design Engineering

- During the design process the software specifications are transformed into design models
- Models describe the details of the data structures, system architecture, interface, and components.
- Each design product is reviewed for quality before moving to the next phase of software development.
- At the end of the design process a design model and specification document is produced.
- This document is composed of the design models that describe the data, architecture, interfaces and components.



Developing a Design Model

Data design:

- This specifies the data structures for implementing the software by converting data objects and their relationships identified during the analysis phase. Various

studies suggest that design engineering should begin with data design, since this design lays the foundation for all other design models.(**data dictionary and ERD**)

Architectural design:

- This specifies the relationship between the structural elements of the software, design patterns, architectural styles, and the factors affecting the ways in which architecture can be implemented.(**DFD**)

Component-level design:

- This provides the detailed description of how structural elements of software will actually be implemented.

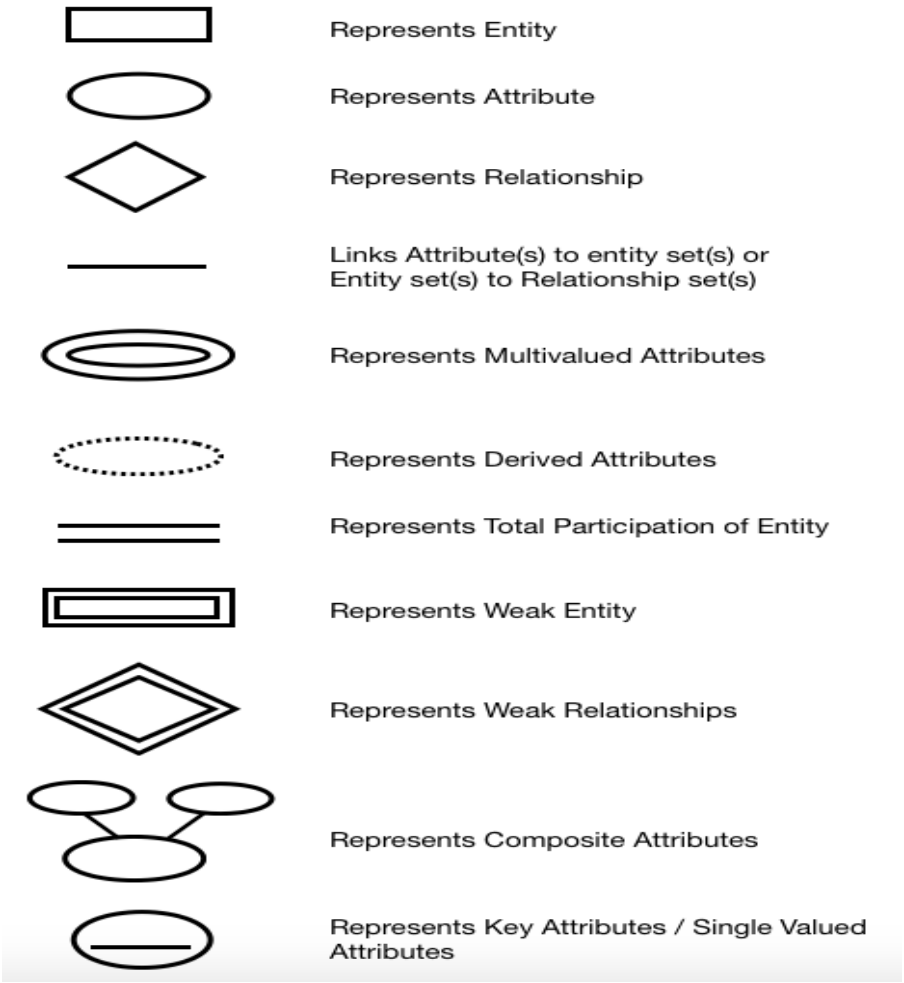
Interface design:

- This depicts how the software communicates with the system that interoperates with it and with the end-users.

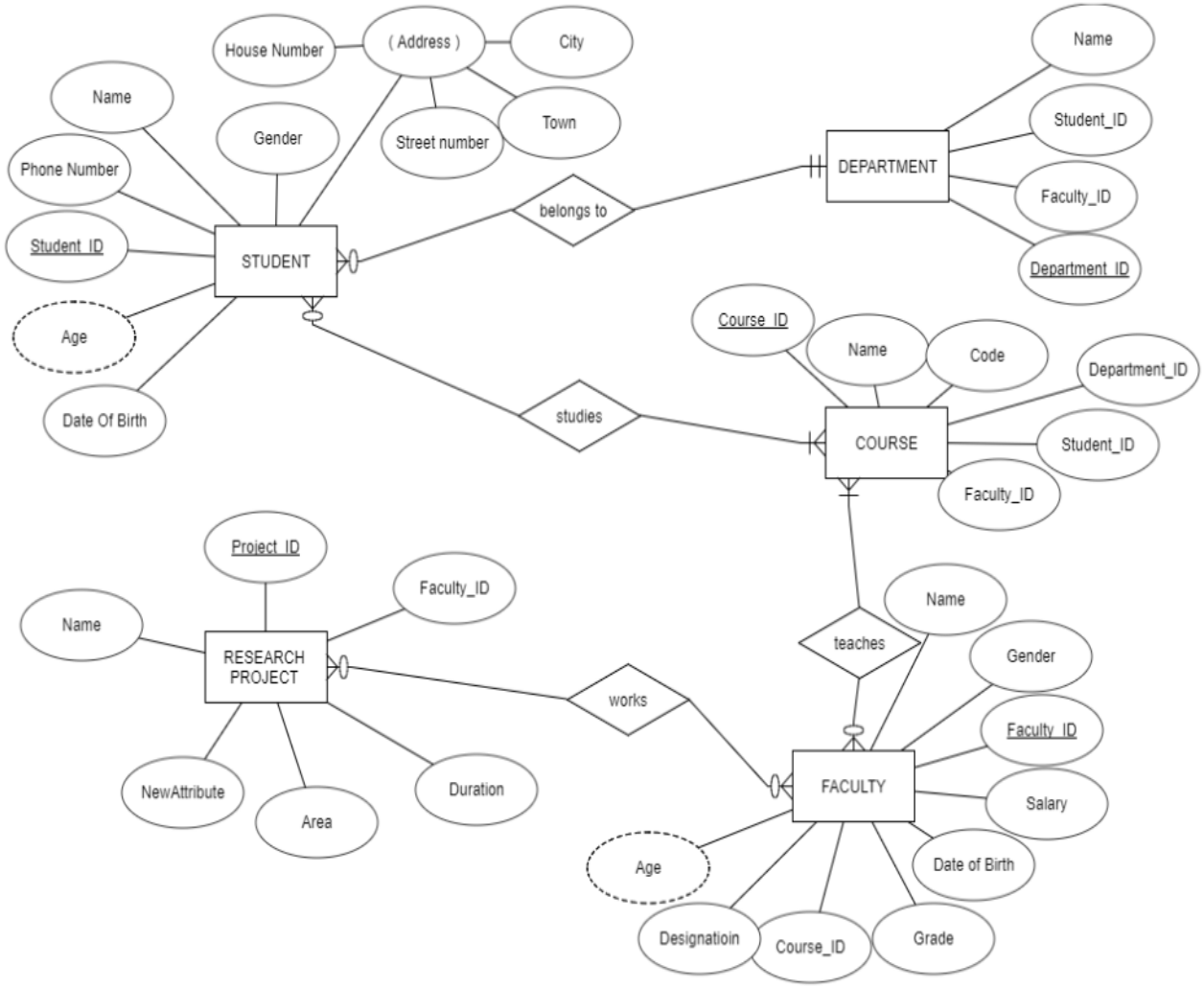
ERD: --relational modeling

IACSD

ER Diagram is a graphical representation of a data model using *entities*, their *attributes* and *relationships* between those entities

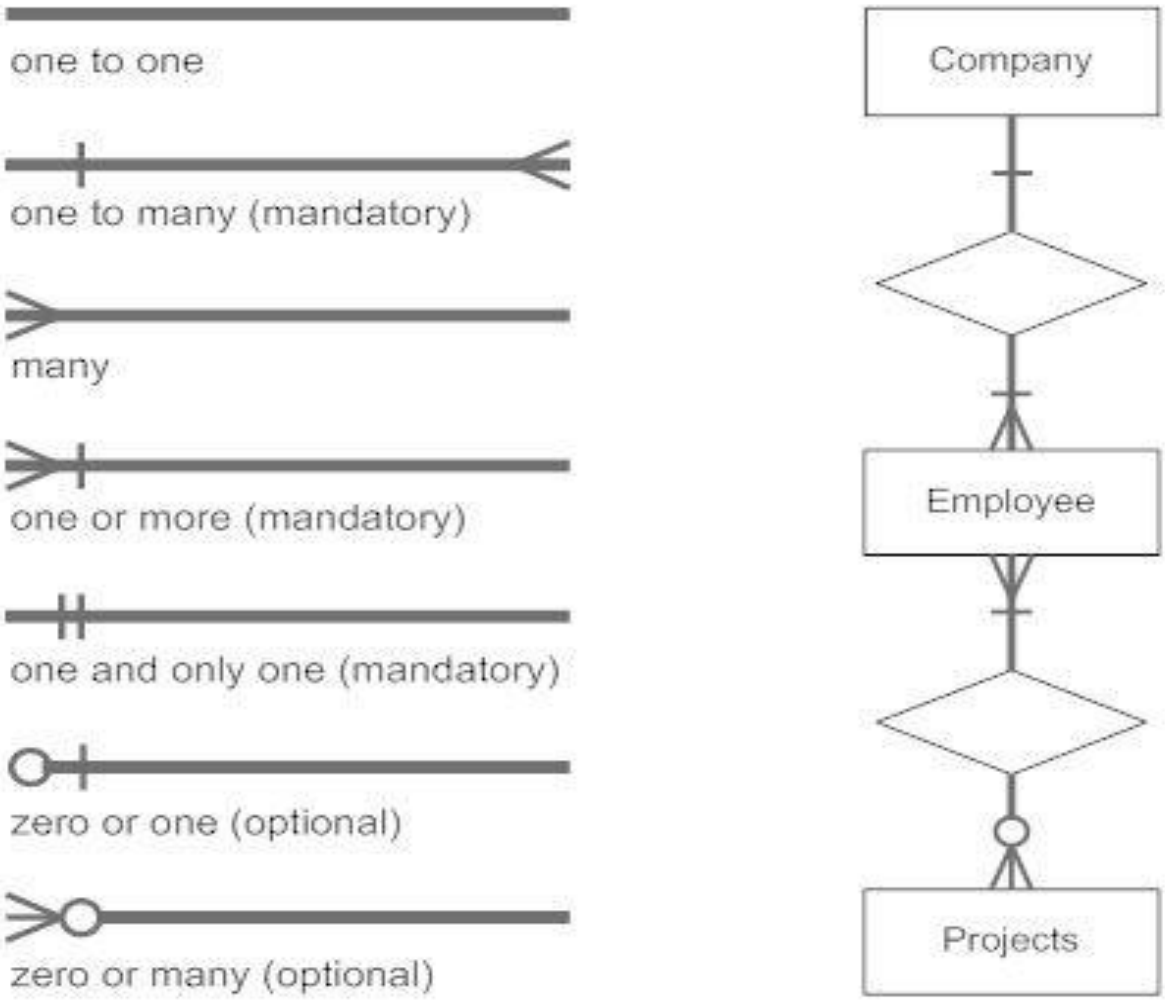


IACSD


House Number
(Address)
City
Name
Gender
Street number
Town
Phone Number
Student_ID
STUDENT
Age
Date Of Birth
belongs to
DEPARTMENT
Name
Student_ID
Faculty_ID
Department_ID
Course_ID
Name
Code
Department_ID
studies
COURSE
Student_ID
Faculty_ID
Project_ID
Faculty_ID
Name
RESEARCH PROJECT
works
NewAttribute
Area
Duration
teaches
Name
Gender
Faculty_ID
FACULTY
Salary
Age
Date of Birth
Designatioin
Course_ID
Grade

IACSD

Information Engineering Style



IACSD

- **Data Dictionary**

Is a list of data elements (entity/table and attribute/column) with their attributes and descriptions. It has a form of a set of **tables**.

Table Name
tbl_persons_mst

Description
This table will have all the details of person entity

Dictionary

Field Name	Datatype	Description	Constraint
pk_person_id	bigint	Unique key of the table	Not Null (auto_increment)
firstname	varchar(50)	Person's first name	Non Null
lastname	varchar(50)	Person's last name	Non Null
createddate	datetime	UTC Date and time on which entry is created	Non Null
modifieddate	datetiime	UTC Date and time on which entry gets modified	Non Null
createdby	integer	Unique id of user who has created this entry	Non Null
modifiedby	integer	Unique id of user who as modified this entry	Non Null
status	tinyint(1)	Status (0-Inactive,1-Active)	Non Null

Entity-Relationship Diagram

Pros

- Easier to see the big picture
- Easier to understand table relations
- Possible to use visual cues to communicate information (e.g. location, proximity, color, shape)

Cons

- Doesn't work with large data models due to space constraints and clutter
- Supports limited amount of details
- May contain very little descriptions (as notes on a diagram)
- Requires careful layout and fitting into canvas
- Hard to search

Data Dictionary

Pros

- May include many data attributes (e.g. list of values, default values, owner, etc.)
- Includes detailed descriptions of each element (table, column)
- Easily searchable

Cons

- Less visually appealing
- More difficult to read

Characteristics of Good Design

- Correctness
- Understandability
- Efficiency
- Maintainability

Basic Design Concepts:

There are seven main principles to keep in mind in the **design model** in **object-oriented programming (OOP)**:

- Abstraction
- Patterns
- Separation of Data
- Modularity

IACSD

- Data Hiding
- Functional Independence
- Refactoring

Abstraction

Abstraction refers to a powerful design tool, which allows software designers to consider components at an abstract level, while neglecting the implementation details of the components.

- **Functional abstraction:** This involves the use of parameterized subprograms. Functional abstraction can be generalized as collections of subprograms referred to as 'groups'.
- **Data abstraction:** This involves specifying data that describes a data object. For example, the data object *window* encompasses a set of attributes (window type, window dimension)
- **Control abstraction:** This states the desired effect, without stating the exact mechanism of control. For example, if and while statements in programming languages (like C and C++)

Patterns

We use **patterns** to identify solutions to design problems that are recurring and can be solved reliably. A pattern must be guaranteed to work so that it may be reused many times over, but it also must be relevant to the current project at the same time.

There are three main patterns:

- **Architectural** - High-level pattern type that can be defined as the overall formation and organization of the software system itself.
- **Design** - Medium-level pattern type that is used by the developers to solve problems in the design stage of development. Can affect how objects or components interact with one another.
- **Idioms** - Low-level pattern type, often known as **coding patterns**, they are used as a workaround means of setting up and defining how components will be interacting with the software itself without being dependent on the programming language

Separation of Data

Known as **separation of concerns**,

- To ensure proper implementation, the two sections must have little to no overlap between them and must have a defined purpose for each component.
- This principle allows each component to be developed, maintained, and reused independently of one another.
- It allows the code to be modified without needing to know the specifics of other components.

IACSD

Modularity

- Modularity is achieved by dividing the software into uniquely named and addressable components, which are also known as **modules.**
- A complex system (large program) is partitioned into a set of discrete modules in such a way that each module can be developed independent of other modules.
- After developing the modules, they are integrated together to meet the software requirements. Note that larger the number of modules a system is divided into, greater will be the effort required to integrate the modules.

Data Hiding

- data hiding allows modules to pass only the required information between themselves without sharing the internal structures and processing
- Data hiding leads to following benefits:
- Leads to low coupling
- Emphasizes communication through controlled interfaces
- Restricts the effects of changes in one component on others

Functional independence

Functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.

- The functional independence is accessed using two criteria i.e Cohesion and coupling.

Cohesion: The degree to which a module performs one and only one function.
Coupling: The degree to which a module is "connected" to other modules in the system

Refactoring

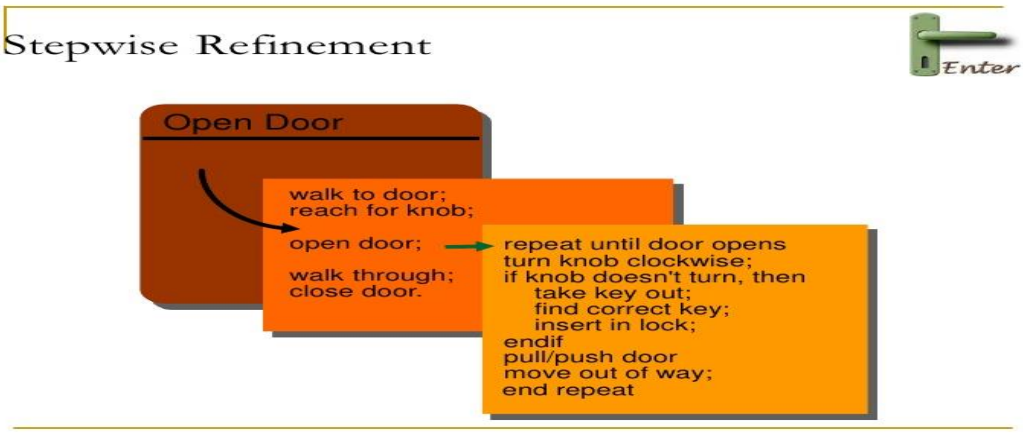
Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure.
When software is refactored, the existing design is examined for

- Redundancy.
- Unused design elements
- Inefficient or unnecessary algorithms.

- Poorly constructed or inappropriate data structures
- Or any other design failure that can be corrected to yield a better design.

Also Termed as Refinement

- Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.
- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.



Functional independence in Details

Ways Components can be dependent

References made from one to another

- Component A invokes B
- A depends on B for completion of its function or process

Amount of data passed from one to another

- Component A passes to B : a parameter ,contents of an array, block of data

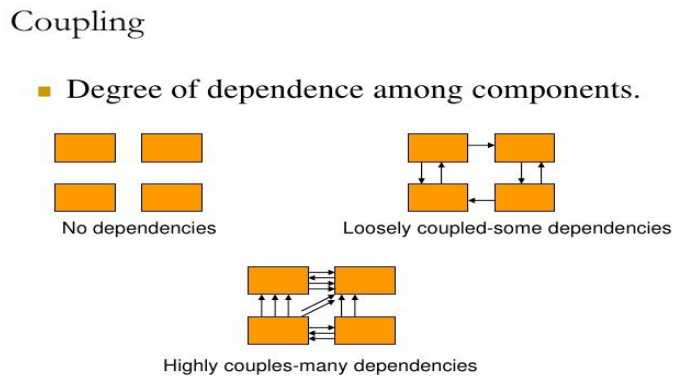
Amount of control one has over the other

- Component passes a control flag to B

- Value of flag tells B the state of some resource or subsystem, process to invoke or whether to invoke a process

Degree of Complexity in the interface between components

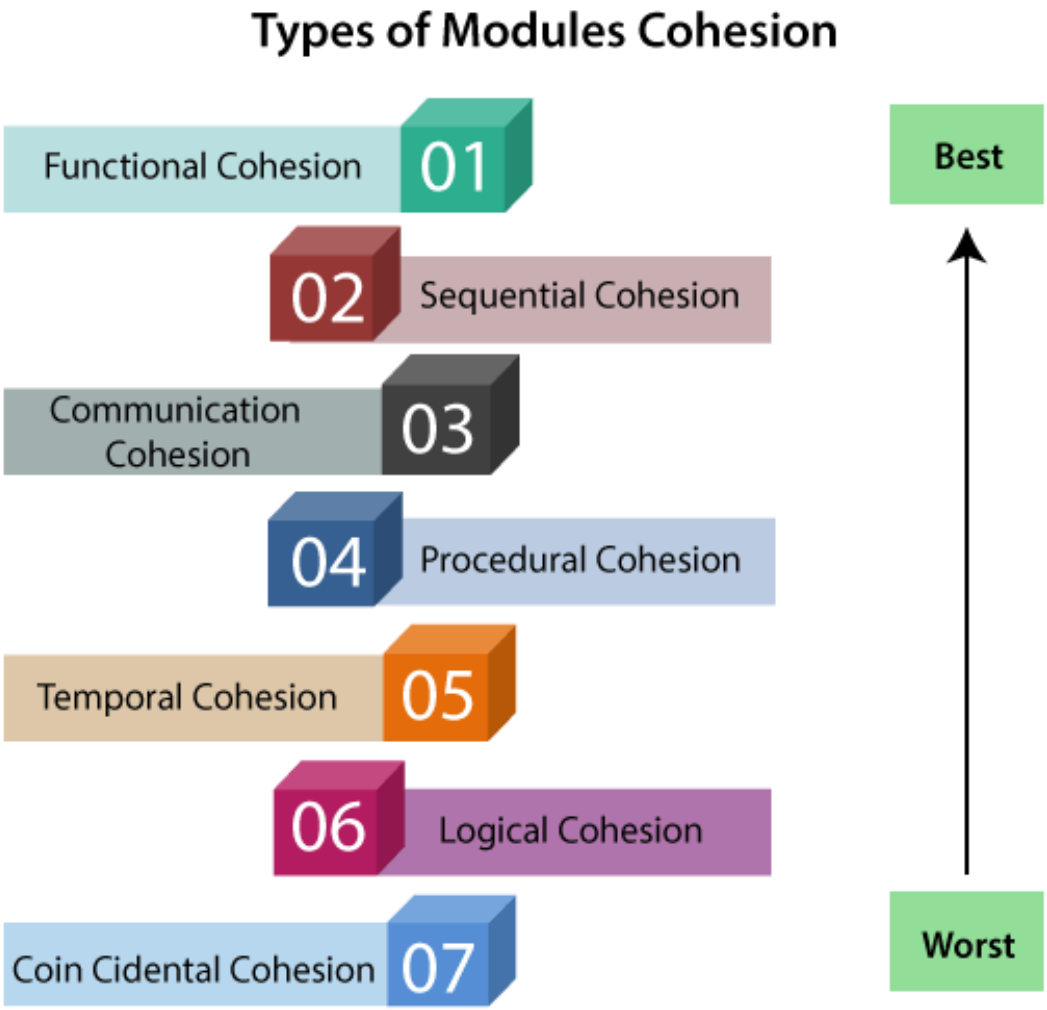
- Component C and D exchange value before D can complete execution



1. **Data Coupling –**
   Data coupling simply means the coupling of data i.e. interaction between data when they are passed through parameters using or when modules share data through parameters. When data of one module is shared with other modules or passed to other modules, this condition is said to be data coupling.
2. **Control Coupling –**
   Control coupling simply means to control data sharing between modules. If the modules interact or connects by sharing controlled data, then they are said to be control coupled. The controlled coupling means that one module controls the flow of data or information by other modules by them the information about what to do.
3. **Common Coupling –**
   Common coupling simply means the sharing of common data or global data between several modules. If two modules share the information through global data items or interact by sharing common data, then they are said to be commonly coupled.
4. **Content Coupling –**
   Content coupling simply means using of data or control information maintained in other modules by one module. This coupling is also known as pathological coupling. In these coupling, one module relies or depends upon the internal workings of another module. Therefore, if any changes have to be done in the

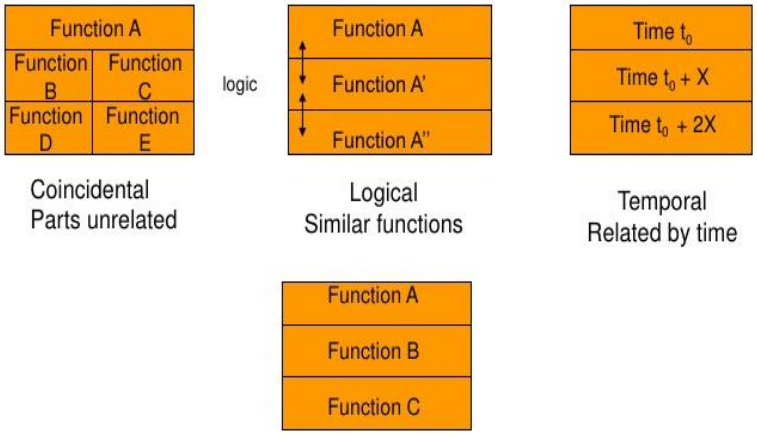
inner working of a module then this will lead to the need for change in the dependent module.

5. **Stamp Coupling –**
   Stamp coupling simply means the sharing of composite data structure between modules. If the modules interact or communicate by sharing or passing day structure that contains more information than the information required to perform their actions, then these modules are said to be stamp coupled.
6. **External Coupling –**
   The external coupling means the sharing of data structure or format that are imposed externally between the modules. External coupling is very important but there should be a limit also. It should be limited to less number of modules with structures.



IACSD

Examples of Cohesion-1



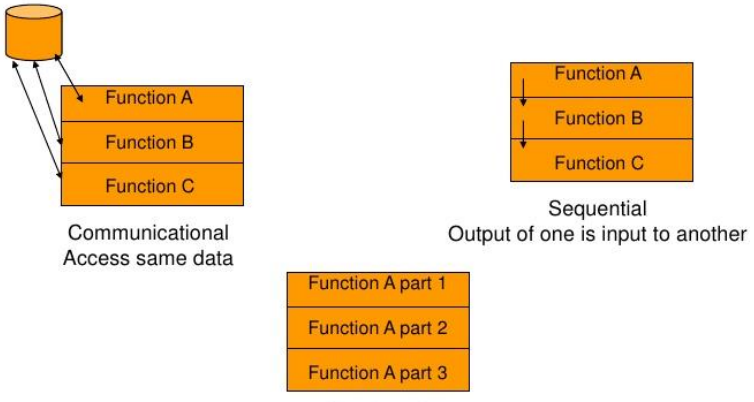
Coincidental
Parts unrelated

Logical
Similar functions

Temporal
Related by time

Procedural
Related by order of functions

Examples of Cohesion-2

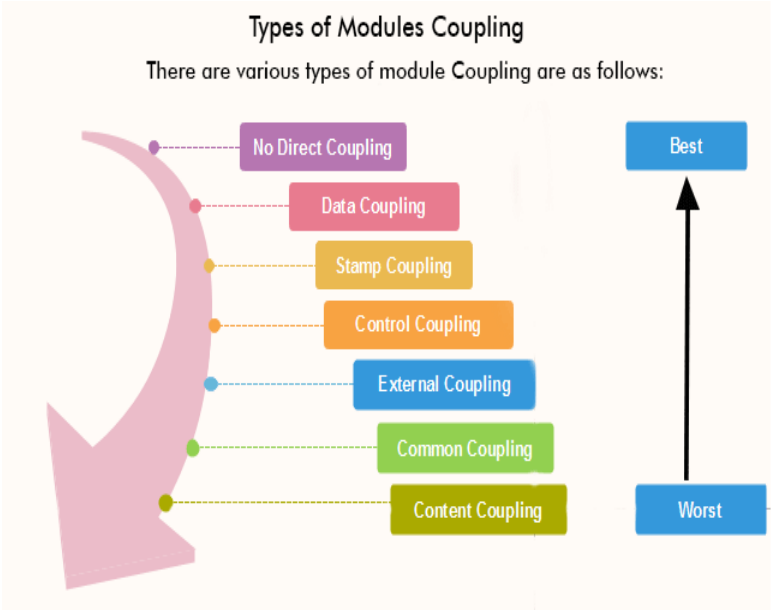


Communicational
Access same data

Sequential
Output of one is input to another

Functional
Sequential with complete, related functions

IACSD



- **Functional Cohesion:** Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.
- **Sequential Cohesion:** A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.
- **Communicational Cohesion:** A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure.
- **Procedural Cohesion:** A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal
- **Temporal Cohesion:** When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.
- **Logical Cohesion:** A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.
- **Coincidental Cohesion:** A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at all.

COHESION	COUPLING
The measure of strength of the association of elements within a module	The measure of interdependence of one module to another
It is the degree to which the responsibility of a single component form a meaningful unit	It describes the relationship between software components
It is a property or characteristic of an individual module	It is a property of a collection of modules

IACSD

UML

- What is UML
- UML building blocks
- Common mechanisms of UML

What is UML

(*A picture is worth a thousand words*)

- UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.
- UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.
- UML stands for **Unified Modeling Language**.
- UML is different from the other common programming languages such as C++, Java.
- UML is a pictorial language used to make software blueprints.
- UML can be described as a general purpose visual modeling language to visualize, specify, construct, and document software system.

Object Oriented Analysis and Design

OO can be defined as an investigation and to be more specific, it is the investigation of objects. Design means collaboration of identified objects.
Purpose:

- Identifying the objects of a system.
- Identifying their relationships.

Making a design, which can be converted to executable using OO languages.

IACSD

UML building blocks

- Things
- Relationships
- Diagrams

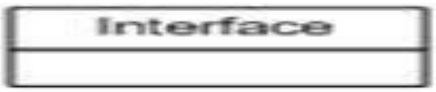
Things are the most important building blocks of UML. Things can be –

- Structural
- Behavioral
- Grouping
- Annotational

Structural Things:

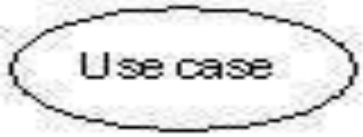
Class – Class represents a set of objects having similar responsibilities.

Interface – Interface defines a set of operations, which specify the responsibility of a class.



Collaboration –Collaboration defines an interaction between elements.

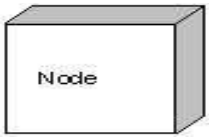
Use case –Use case represents a set of actions performed by a system for a specific goal.



Component –Component describes the physical part of a system.

IACSD

Node – A node can be defined as a physical element that exists at run time.

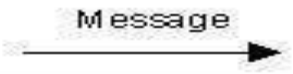


Behavioral Things

- **A behavioral thing** consists of the dynamic parts of UML models.

Following are the behavioral things −

- **Interaction –** Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.



- **State machine –** State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change

Grouping Things

- **Grouping things** can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available −
- **Package –** Package is the only one grouping thing available for gathering structural and behavioral things.

Annotational Things

- Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements.
- Note - It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element.

IACSD

note

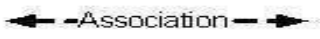
Relationship

- **Relationship** is another most important building block of UML. It shows how the elements are associated with each other and this association describes the functionality of an application.

There are four kinds of relationships available.

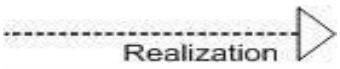
- **Dependency** is a relationship between two things in which change in one element also affects the other.

- **Association** is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.



- **Generalization** can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the world of objects.

- **Realization** can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces.



Diagrams

UML diagrams are organized into two distinct groups:

- structural diagrams
- behavioral or interaction diagrams

Structural UML diagrams

- Class diagram
- Package diagram
- Object diagram
- Component diagram
- Composite structure diagram
- Deployment diagram

Behavioral UML diagrams

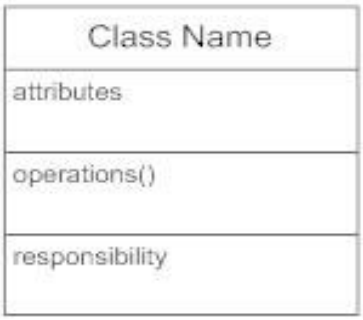
- Activity diagram
- Sequence diagram
- Use case diagram
- State diagram
- Communication diagram
- Interaction overview diagram
- Timing diagram

Class Diagram Symbols and Notations

- **Classes**

  Classes represent an abstraction of entities with common characteristics.

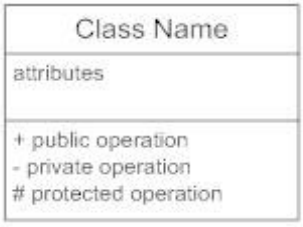
IACSD



Class

- **Visibility**

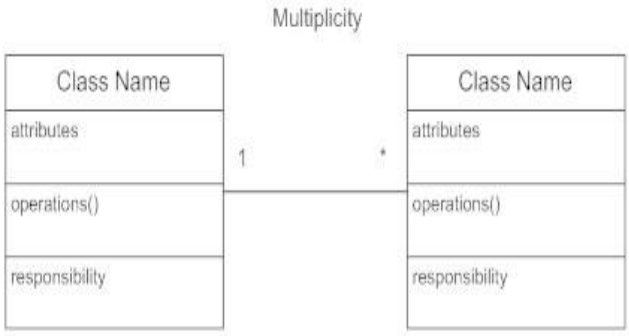
  Use visibility markers to signify who can access the information contained within a class.



Visibility

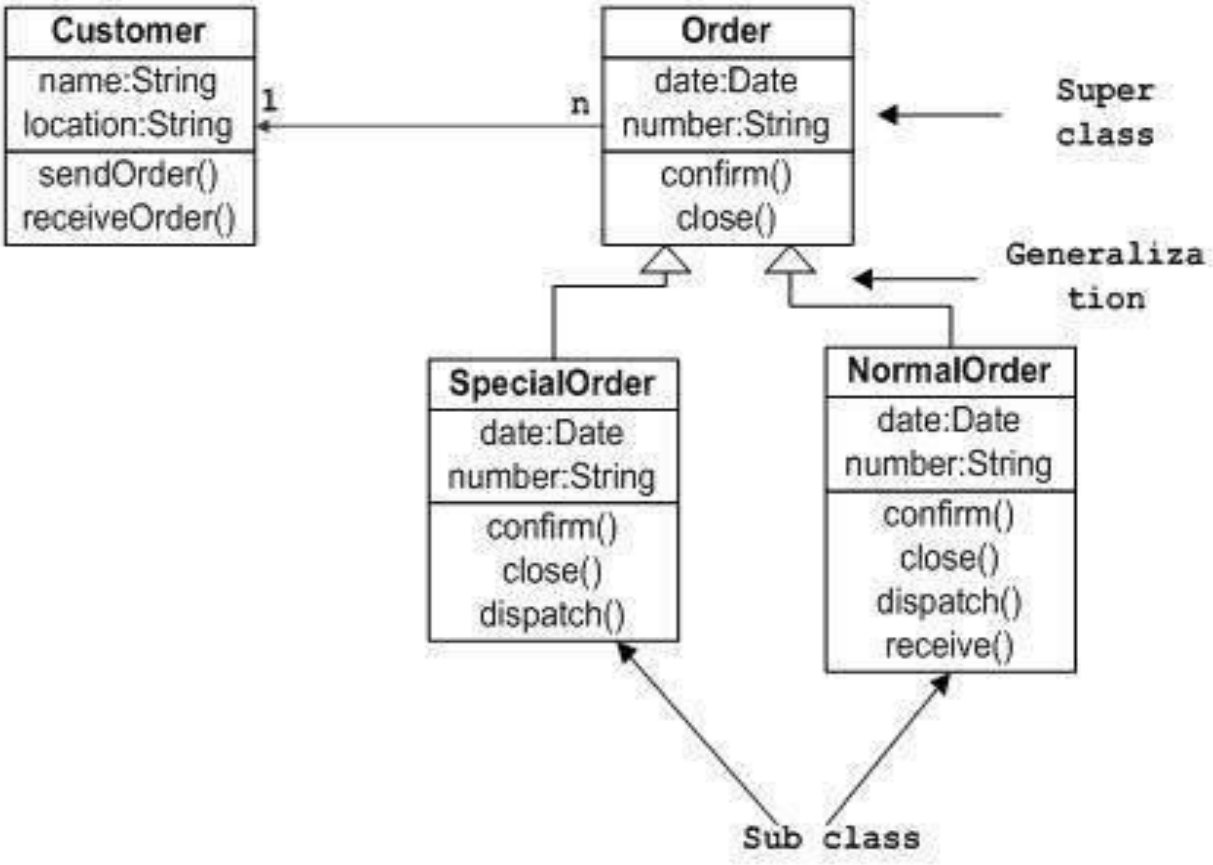
Marker	Visibility
+	public
-	private
#	protected
~	package

- **Associations**
- Associations represent static relationships between classes. Place association names above, on, or below the association line.
- **Constraint**
- Place constraints inside curly braces {}.



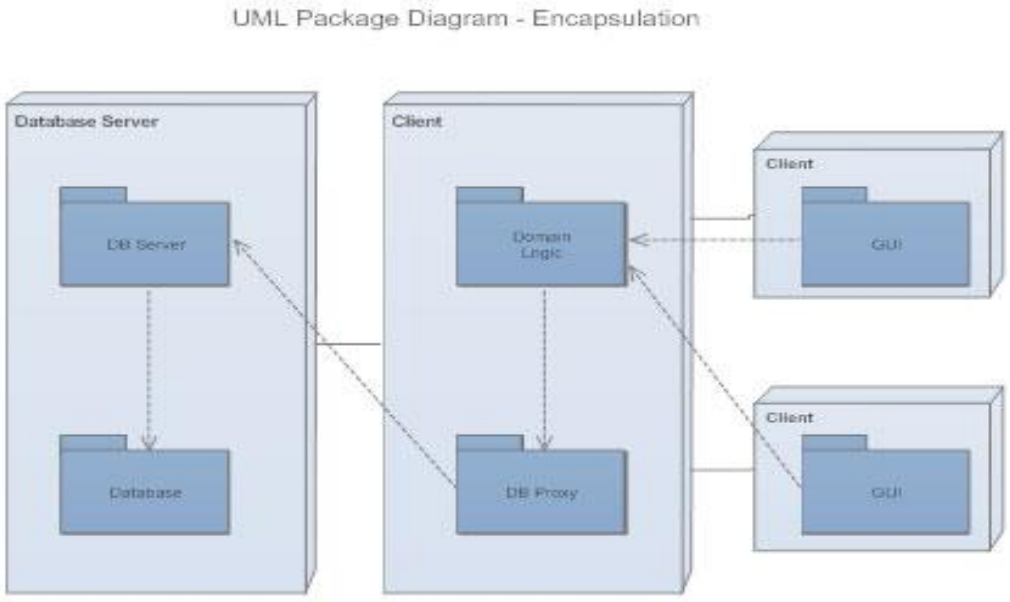
Indicator	Meaning
0..1	Zero or one
1	One only
0..*	0 or more
1..* *	1 or more
n	Only n (where n > 1)
0..n	Zero to n (where n >1)
1..n	One to n (where n > 1)

Sample Class Diagram



Package Diagram

Package diagrams are a subset of class diagrams, but developers sometimes treat them as a separate technique. Package diagrams organize elements of a system into related groups to minimize dependencies between packages.

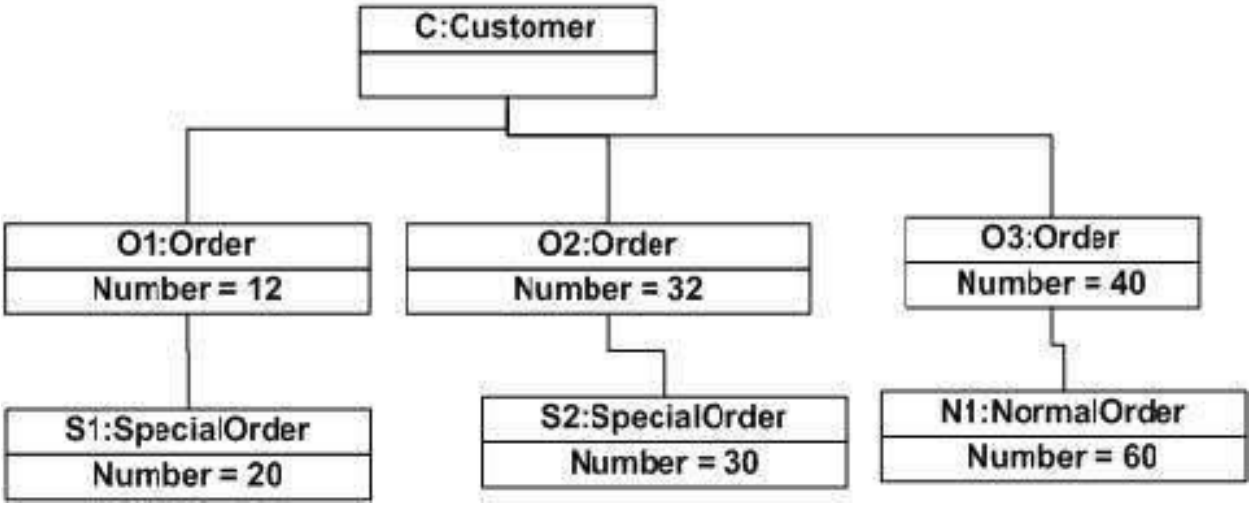


Object Diagram

Object diagrams describe the static structure of a system at a particular time. They can be used to test class diagrams for accuracy.

object diagram is an instance of a class diagram.

Object diagram of an order management system



Use Case Diagram

Use case diagrams model the functionality of a system using actors and use cases.

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Show the interaction among the requirements are actors.

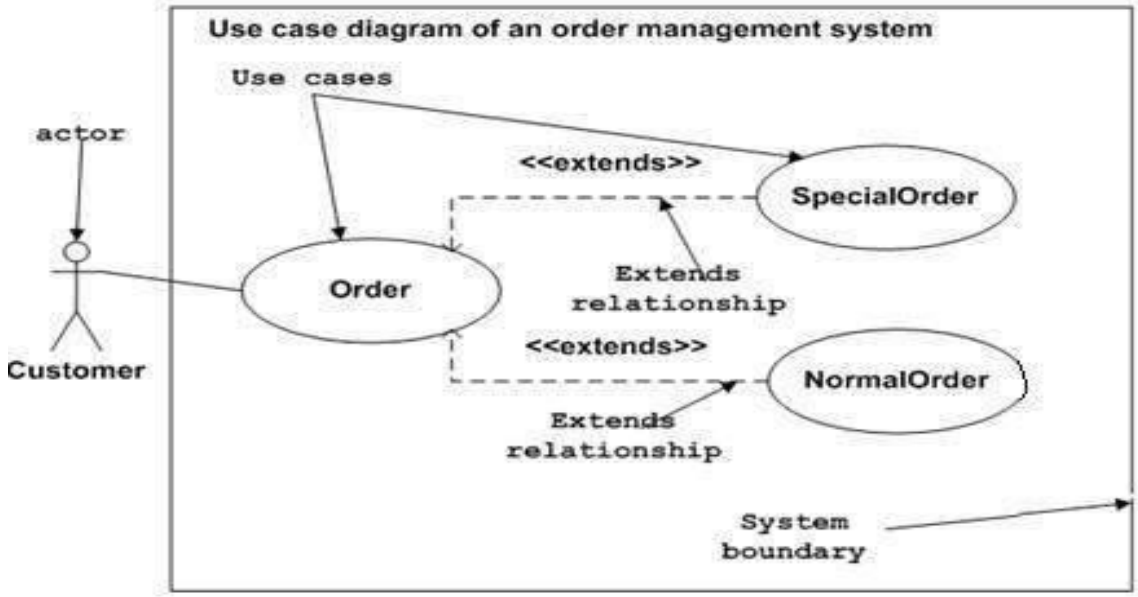
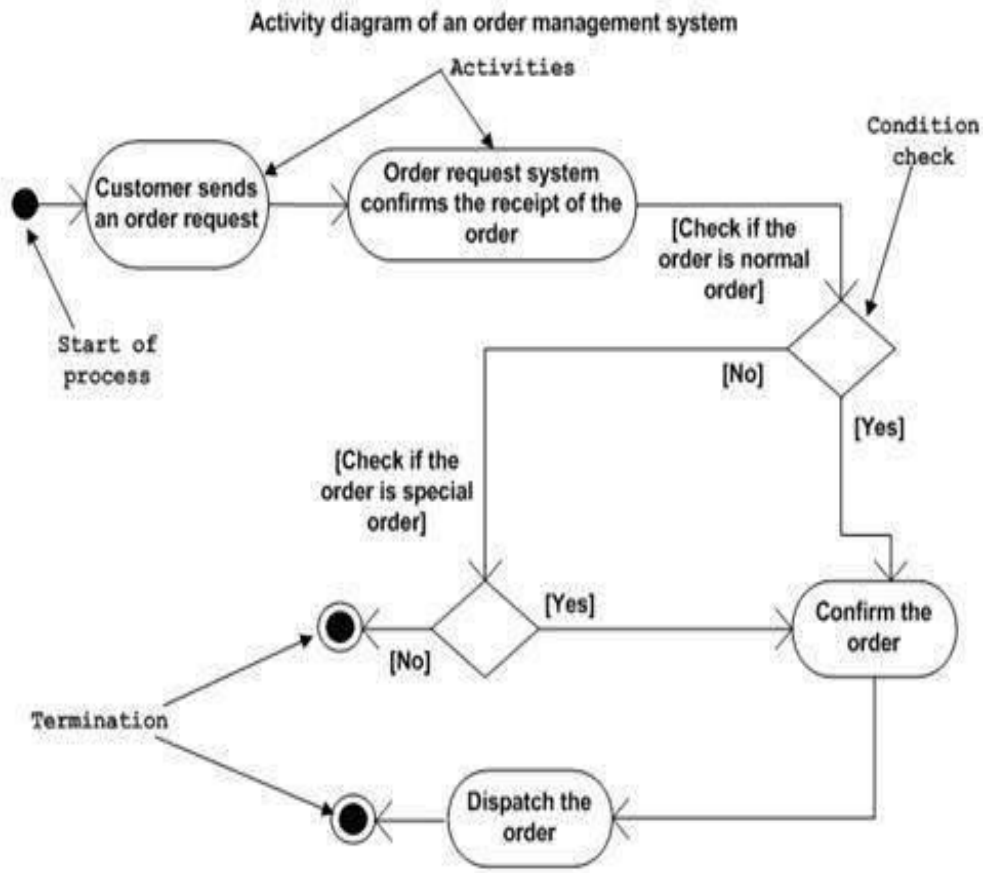


Figure: Sample Use Case diagram

Activity Diagram

- Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity.
- An activity represents an operation on some class in the system that results in a change in the state of the system.
- Typically, activity diagrams are used to model workflow or business processes and internal operation



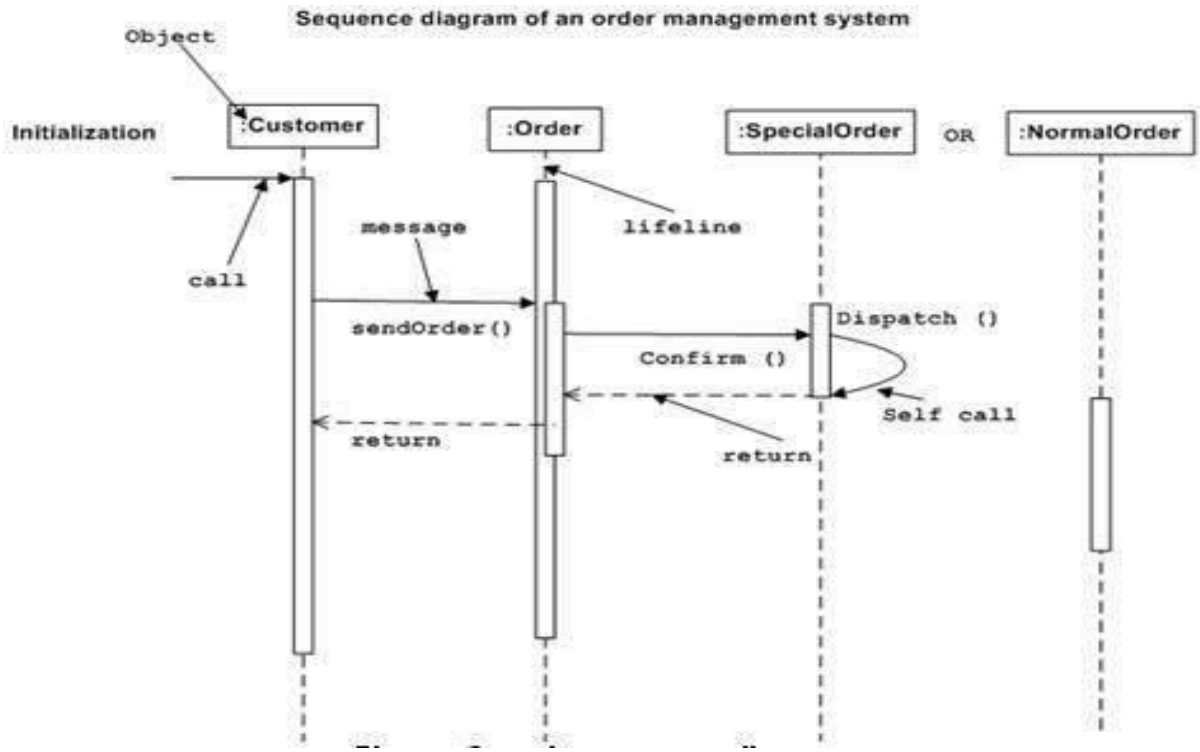
Sequence Diagram

- Sequence diagrams describe interactions among classes in terms of an exchange of messages over time.

It Represents:

- Objects taking part in the interaction.
- Message flows among the objects.

- The sequence in which the messages are flowing.
- Object organization.



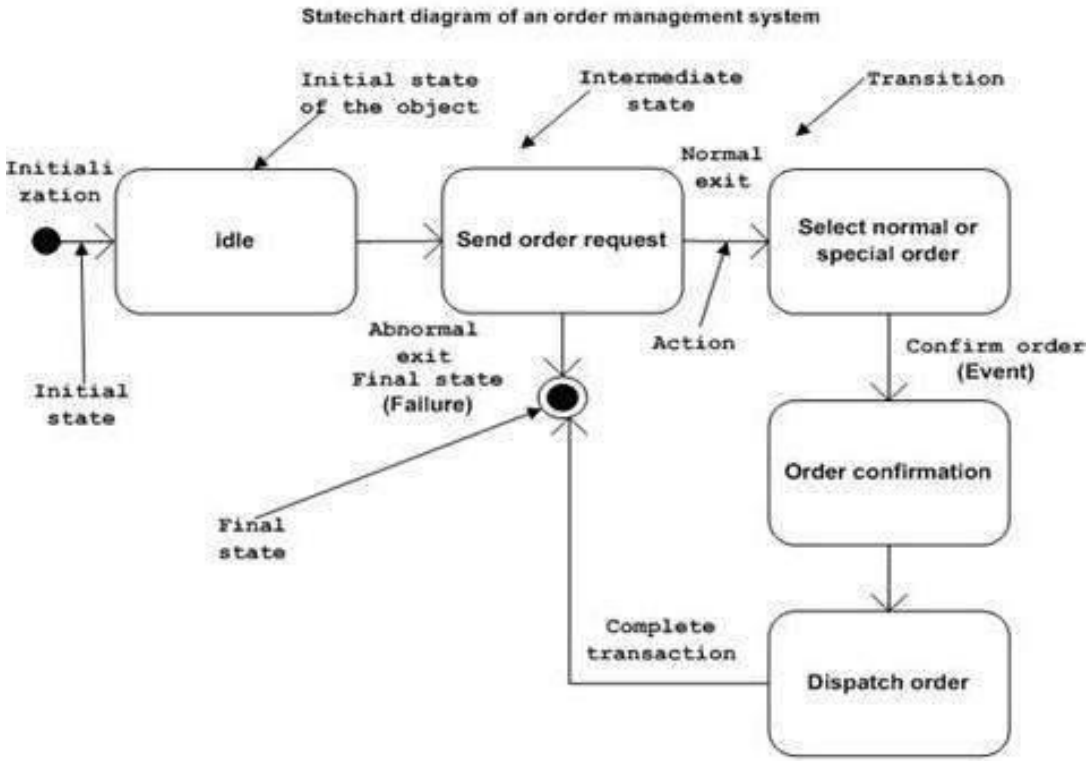
- **Interaction Overview Diagram**
  Interaction overview diagrams are a combination of activity and sequence diagrams.
- **Timing Diagram**
  A timing diagram is a type of behavioral or interaction UML diagram that focuses on processes that take place during a specific period of time. They're a special instance of a sequence diagram, except time is shown to increase from left to right instead of top down.
- **Communication Diagram**
  Communication diagrams model the interactions between objects in sequence. They describe both the static structure and the dynamic behavior of a system.

State Diagram

- Statechart diagrams, now known as state machine diagrams and state diagrams describe the dynamic behavior of a system in response to external stimuli. State diagrams are especially useful in modeling reactive objects whose states are triggered by specific events.

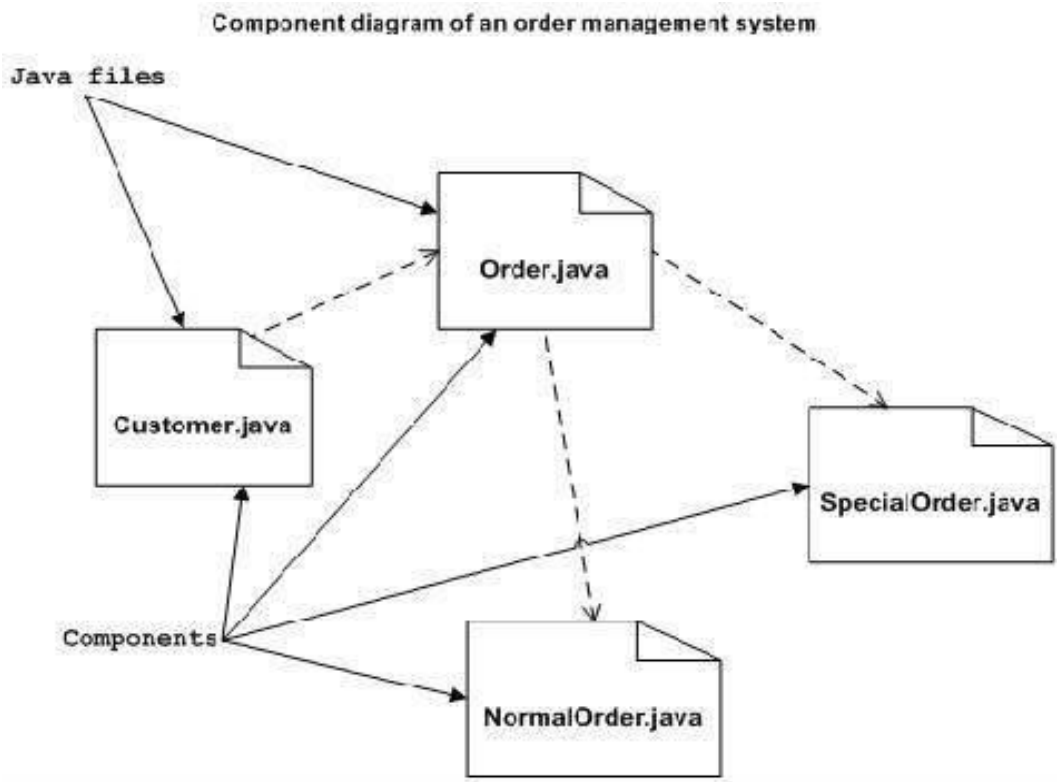
To Draw This:

- Identify the important objects to be analyzed.
- Identify the states.
- Identify the events.



- **Component Diagram**

  Component diagrams describe the organization of physical software components, including source code, run-time (binary) code, and executables.

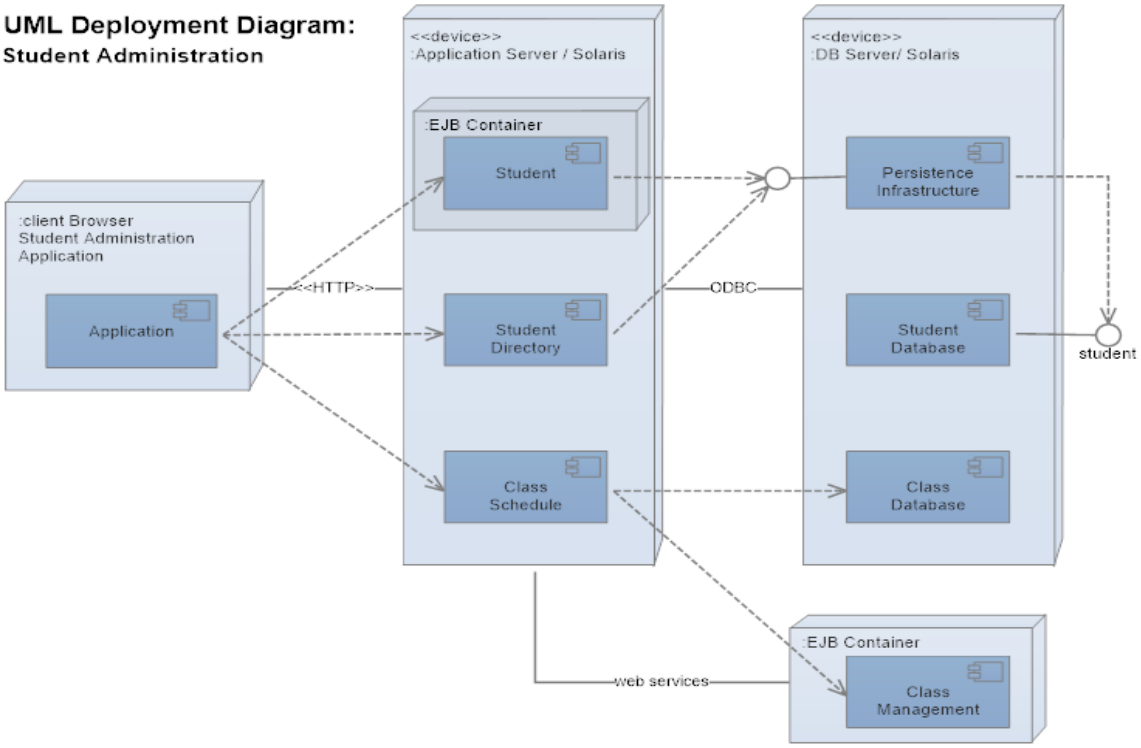


- **Deployment Diagram**
  Deployment diagrams depict the physical resources in a system, including nodes, components, and connections.

The purpose of deployment diagrams −

- Visualize the hardware topology of a system.

- Describe the hardware components used to deploy software components.
- Describe the runtime processing nodes.



Session 3

Introduction to Agile development model

➨ **WHAT IS AGILE?**

IACSD

Agile software development refers to software development methodologies centered round the idea of iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. The ultimate value in agile development is that it enables teams to deliver value faster, with greater quality and predictability, and greater aptitude to respond to change.

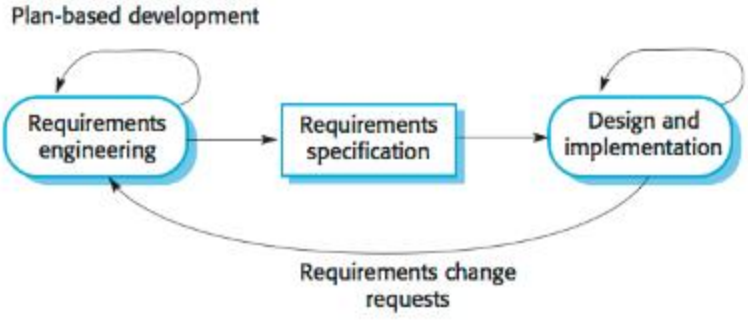
Plan-driven development

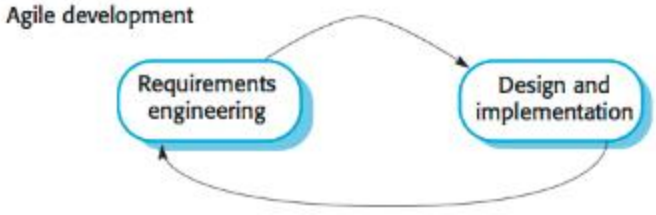
A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance. Not necessarily waterfall model: plan-driven, incremental development is possible. Iteration occurs within activities.

Agile development

Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on many technical, human, and organizational issues.





Parameter	Traditional	Agile
Requirements	Fixed	Evolve
Time & People	May vary	Fixed
Customer Involvement	Before, After	During
Negotiable	Estimates	Schedule
Testing	After code	Integrated
Feedback	After	During
Concentration on	Processes; reviews	Workable software
Focus	Plan driven	Value driven
Stages	Requirements, Design, Code, Test, Feedback	(Plan-do-adapt)*

Figure: plan-driven vs. Agile methodologies

Agile Manifesto

Better ways of developing software by doing it and helping others do it.

Agile Says value to:

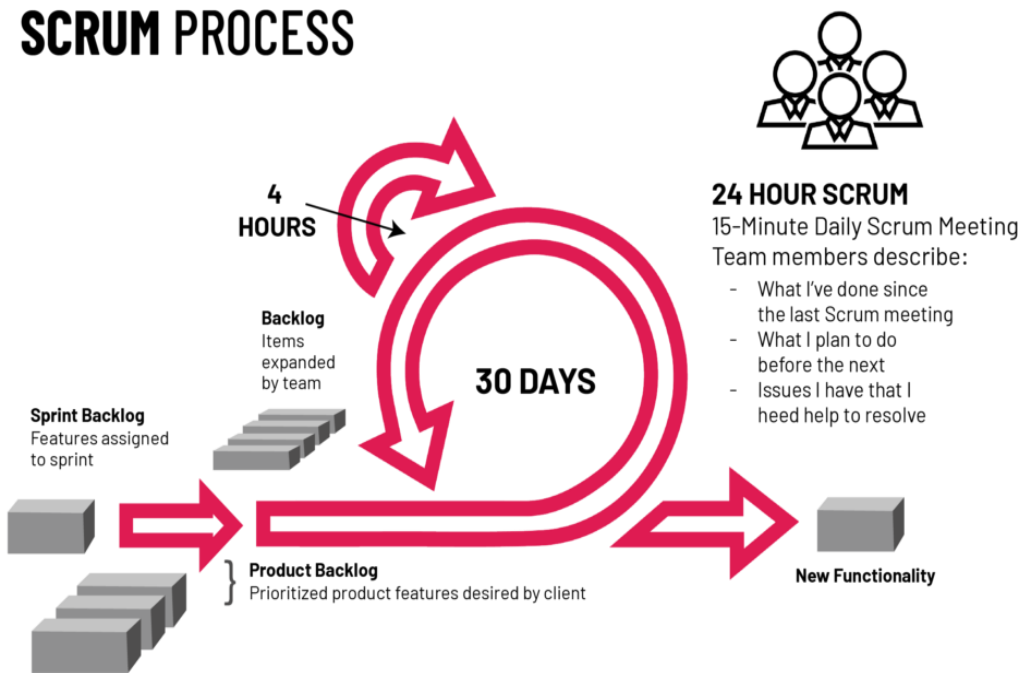
- **Individuals and interactions over processes and tools**
- **Working software over comprehensive documentation**
- **Customer collaboration over contract negotiation**
- **Responding to change over following a plan**

Agile Methodology: Types

- **Scrum**
- **eXtreme Programming (XP)**
- **Feature Driven Development(FDD)**
- **Dynamic Systems Development Method (DSDM)**

Agile Methodology: SCRUM

- **Scrum** is a **subset** of Agile. It is a lightweight process framework for agile development, and the most widely-used one.
- **A "process framework"** is a particular set of practices that must be followed in order for a process to be consistent with the framework.
- Scrum process framework requires the use of development cycles called **Sprints**
- **"Lightweight"** means that the overhead of the process is kept as small as possible, to maximize the amount of productive time available for getting useful work done.



WHAT ARE THE SCRUM REQUIREMENTS?

- Scrum does not define just what form requirements are to take, but simply says that they are gathered into the Product Backlog, and referred to generically as "Product Backlog Items," or "PBIs" for short
- Most Scrum projects borrow the "XP" (Extreme Programming) practice of describing a feature request as a "User Story,"

IACSD

User Stories

XP eXtream Programming

- **Definition - What does *Extreme Programming (XP)* mean?**
- Extreme Programming (XP) is an intense, disciplined and agile software development methodology focusing on coding within each software development life cycle (SDLC) stage
- These stages are: Continuous integration to discover and repair problems early in the development process Customer involvement and rapid feedback

  These XP methodology disciplines are derived from the following four key values of Kent Beck, XP's originator:

- **1. Communication:** Communication between team members and customers must occur on a frequent basis and result in open project discussion without fear of reprisal.
- **2. Simplicity:** This involves using the simplest design, technology, algorithms and techniques to satisfy the customer's needs for the current project iteration.
- **3. Feedback:** Feedback must be obtained at multiple, distinct levels, e.g., unit tests, code review and integration.
- **4. Courage:** Implement difficult but required decisions.

IACSD

WHAT ARE THE BENEFITS OF AGILE?

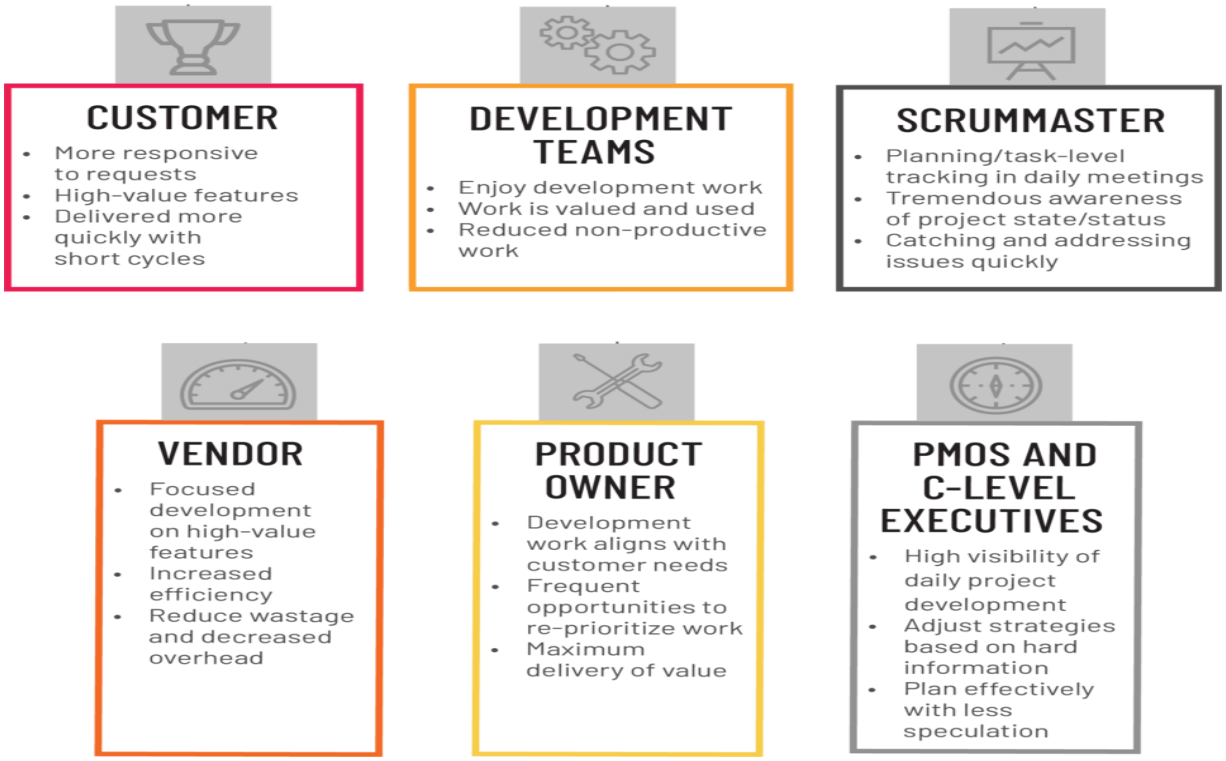


Figure: Benefits of Agile.

4 essential components of Agile Project

1. An agile project plan is divided into releases and sprints

Agile planners define a release, which involves creating a new product or substantially updating an existing product. Each release is broken down into several iterations, also called *sprints*. Each sprint has a fixed length, typically 1-2 weeks, and the team has a predefined list of work items to work through in each sprint. The work items are called *user stories*. The release plan is broken down into several iterations (sprints) that include user stories (items)


IACSD

2. Planning is based on user stories

A *user story* briefly describes a need experienced by your users. For example:

- "As a team member, I need to know which tasks are currently assigned to me"
- "As a team leader, I need to receive email notification when a task is stuck or behind schedule"

Unlike in traditional project management methodologies like *waterfall*, in which teams would create detailed technical specifications of exactly what they would build, in agile planning, the team only documents **what the user needs**. Throughout the sprint, the team figures out together how to address that specific need in the best way possible.

3. Planning is iterative and incremental

The agile process is focused on the concept of iteration. All sprints are of equal length, and an agile team repeats the same process over and over again in every sprint. Each sprint should result in working features that can be rolled out to end users.

An iterative process allows the team to learn what they are capable of, estimate how many stories they can complete in a given timeframe (the team's *velocity)* and learn about problems that impede their progress. These problems can be taken care of in subsequent sprints.

4. Estimation is done by team members themselves

A core ethic of agile planning is that development teams should participate in planning and estimation, and not have the work scope "dictated" to them by management.

In this spirit, agile planning allows teams to assign *story points* to user stories in the release plan.

IACSD

What is a story point?

In agile methodology, a story point is a number which reflects the complexity or amount of work involved in developing a user story. For example, a team can assign 1 point to a simple user story, 2-3 points for moderately complex and 4-5 points for a big story – based on their understanding of the work involved.

An alternative estimation unit for agile stories is *ideal time*: how long a user story should take to develop, assuming zero interruptions.

Sprint Planning Process

Here is how an agile team plans at the beginning of a new sprint, as part of an existing release plan:

1. **Do a retrospective meeting** to discuss the previous sprints and lessons learned.
2. **Run a sprint planning meeting** to analyze the release plan and update it according to velocity in recent sprints, changes to priorities, new features, or idle time that wasn't planned for in the release.
3. **Make sure user stories are detailed enough** to work on. Elaborate on tasks that are not well defined, to avoid surprises.
4. **Break down user stories into specific tasks**. For example, the user story "view tasks assigned to me" can be broken up into UX design of a "my tasks screen", back-end implementation, and front-end development of the interface. Keep size of tasks small, no more than one work day.
5. **Assign tasks to team members** and confirm that they are committed to performing them. In the agile/scrum framework this is done by the Scrum Master.
6. **Write the tasks on (physical) sticky cards** and hang them up on a large board visible to the entire team. All the user stories in the current sprint should be up on the board.
7. **Track progress of all the tasks** on a grid, by recording who is responsible for completing each task, estimated time to complete it, remaining hours, and actual hours used. This time tracking should be updated by all team members and visible to everyone.
8. **Track velocity using a burndown chart.** During the sprint, use the team's time tracking to calculate a chart showing the number of tasks or hours remaining, vs.

the plan. The slope of the burndown chart shows if we are on schedule, ahead, or behind schedule.

What is JIRA?

JIRA is a tool developed by Australian Company Atlassian. This software is used for **bug tracking, issue tracking,** and **project management**. The name "JIRA" is actually inherited from the Japanese word "Gojira" which means "Godzilla". The basic use of this tool is to track issue and bugs related to your software

It is also used for project management. The JIRA dashboard consists of many useful functions and features which make handling of issues easy. Some of the key features are listed below.

Important Points to Note

The following points explain some interesting details of JIRA. JIRA is an incident management tool.

- JIRA is developed by Atlassian Inc., an Australian Company.
- JIRA is a platform independent tool; it can be used with any OS.
- JIRA is multi-lingual tool – English, French, German, Japanese, Spanish, etc.
- JIRA supports MySQL, Oracle, PostgreSQL and SQL server in the backend.
- JIRA can be integrated with many other tools – Subversion, GIT, Clearcase, Team Foundation Software, Mercury, Concurrent Version System and many more.

Following are some of the most significant uses of JIRA:

- JIRA is used in Bugs, Issues and Change Request Tracking.
- JIRA can be used in Helpdesk, Support and Customer Services to create tickets and
- track the resolution and status of the created tickets.

JIRA is useful in Project Management, Task Tracking and Requirement Management.

- JIRA is very useful in Workflow and Process management.

JIRA – Project

A Project contains issues; a JIRA project can be called as a collection of issues. A JIRA Project can be of several types. For example –

- Software Development Project
- Migration to other platform project

IACSD

- Help Desk Tracking Project
- Leave Request Management System
- Employee Performance System
- Website Enhancement

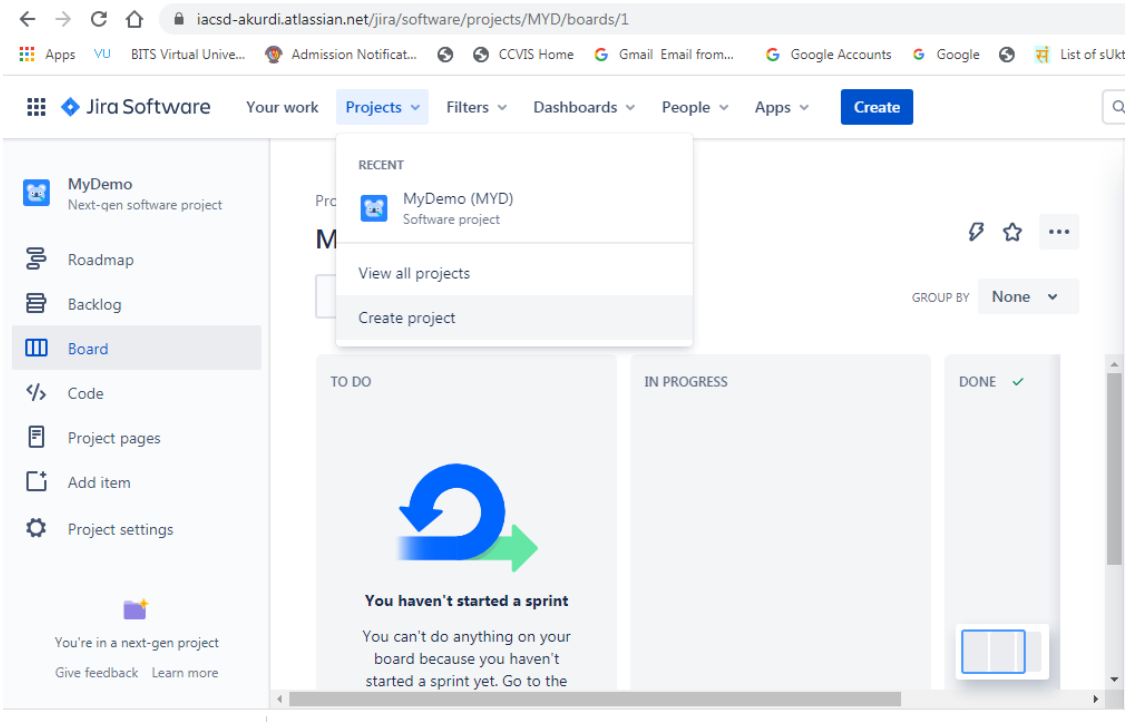
What is Sprint?

A sprint is a fixed time period in a continuous development cycle where teams complete work from their product backlog. At the end of the sprint, a team will typically have built and implemented a working product increment. Jira Software makes your backlog the center of your sprint planning meeting, so you can estimate stories, adjust sprint scope, check velocity, and re-prioritize issues in real-time.

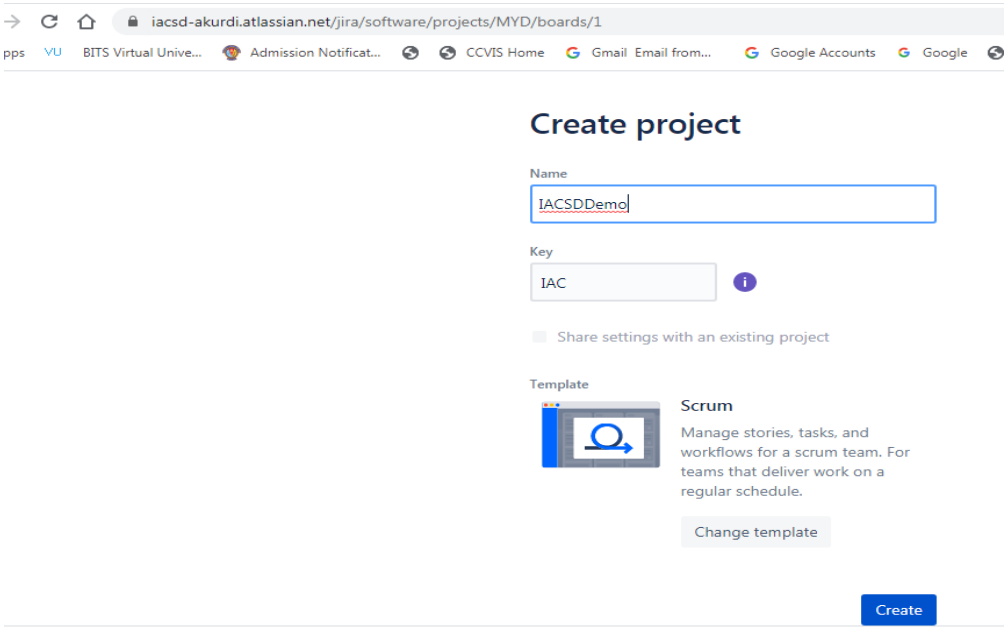
Step 1: Creating New Project.

To create a project, the user should login as a JIRA Service Desk Admin and then Click on Create Project.

The following screenshot shows how to reach to the Create Project button from the Dashboard.

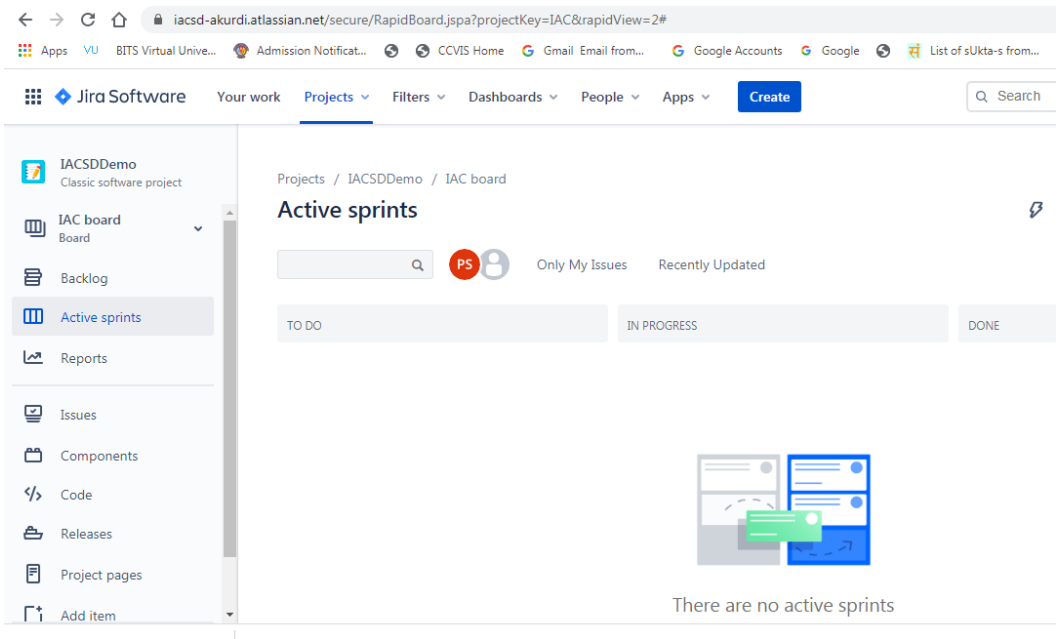


IACSD

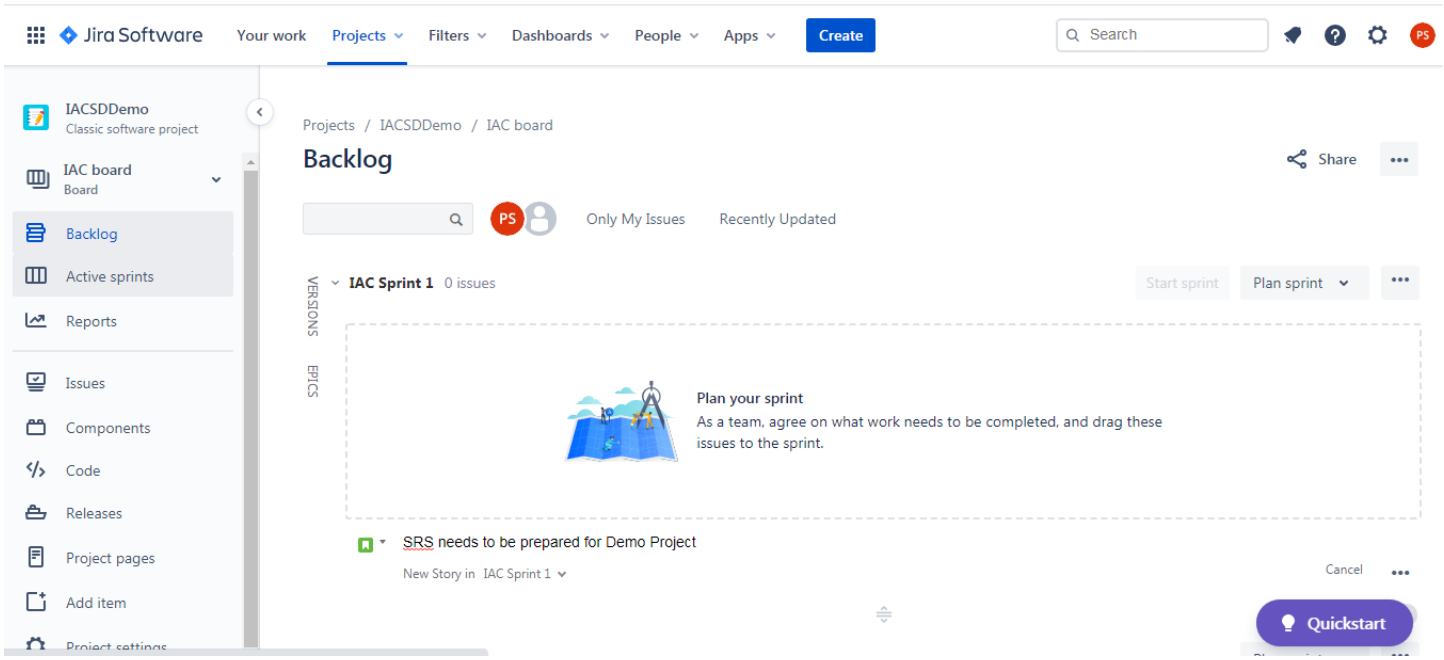


Step 2: Create a sprint

1. Go to the **Backlog** of your Scrum project.
2. Click the **Create Sprint** button at the top of the backlog.
   Note that you can create more than one sprint, if you want to plan work several weeks in advance.



IACSD



Step 3: Fill your sprint with stories from the backlog

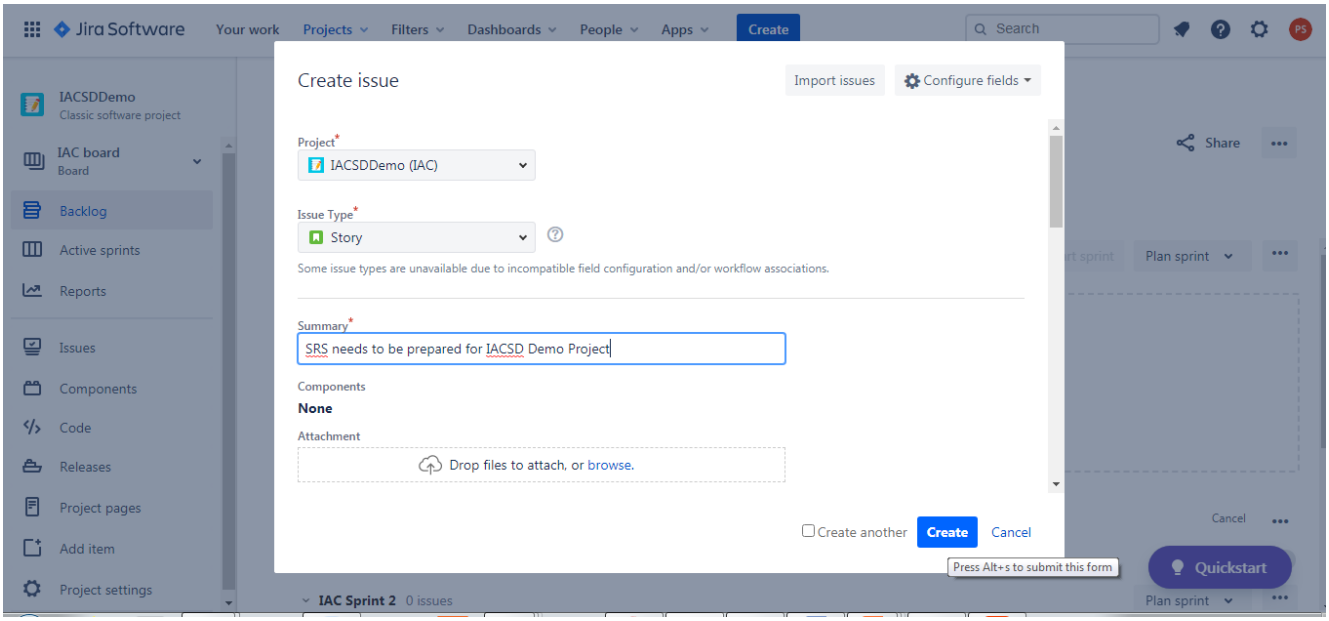
Once you've created your sprint, you'll need to fill it with issues. Before you do this, make sure you sit down with your team and discuss what work you'd like to commit to doing. Ensure you add enough work for everyone in the team.

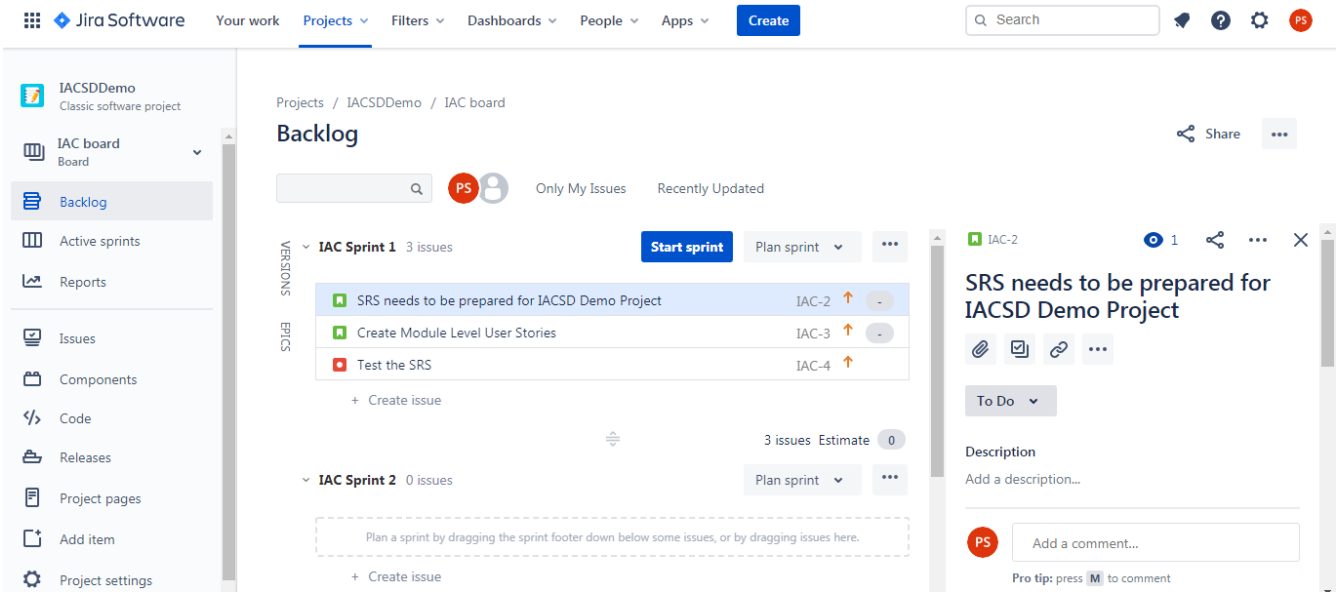
To add stories to your sprints

1. Navigate to the Backlog.
2. Drag and drop issues from the Backlog onto your sprint.

   Note that you can also add an issue to your sprint by editing the issue and updating the **Sprint** field.

IACSD





Step 4: Start sprint

Once you've added issues to your sprint and the team is ready to work, you'll need to start the sprint.

Note, you can only start a sprint, if:

- You haven't started one already. If you want to have more than one active sprint at a time, try the Parallel Sprints feature, and
- The sprint is at the top of the backlog. If you want to start a planned sprint that is lower down, you'll need to reorder your sprints to move it to the top.

Step 5: Monitor your team's progress.

During the sprint, you'll probably want to monitor the team's progress. One way of doing this is by viewing the Sprint Report.
During sprints, teams work together to complete the stories they committed to at the start of the sprint. This typically requires a lot of collaboration, so we recommend doing team standup meetings every day, so you know what everyone in the team is working on.

Step 6: Close the sprint

To close a sprint

1. Navigate to the **Active sprints** of your Scrum board.
2. If necessary, select the sprint you want to complete from the sprint drop-down.
   Note that if you have multiple sprints in the Active sprints of your board, the 'Complete Sprint' button will not appear until you select one of the sprints.

3. Click **Complete Sprint**. All completed issues will move out of Active sprints.
4. If the sprint has incomplete issues, you'll be asked to move them to one of the following:
   - The backlog
   - Any future sprint, or
   - A new sprint

IACSD

JIRA – Workflow

In JIRA, workflow is used to track the lifecycle of an Issue. Workflow is a record of statuses and transitions of an issue during its lifecycle. A status represents the stage of an issue at a particular point. An issue can be in only one status at a given point of time like Opened, To Do, Done, Closed, Assigned, etc. A transition is a link between two statuses when an issue moves from one status to another. For an issue to move between two statuses, a transition must exist. In a simple way, a transition is some kind of work done on the issue, while status is the impact of work on that issue.

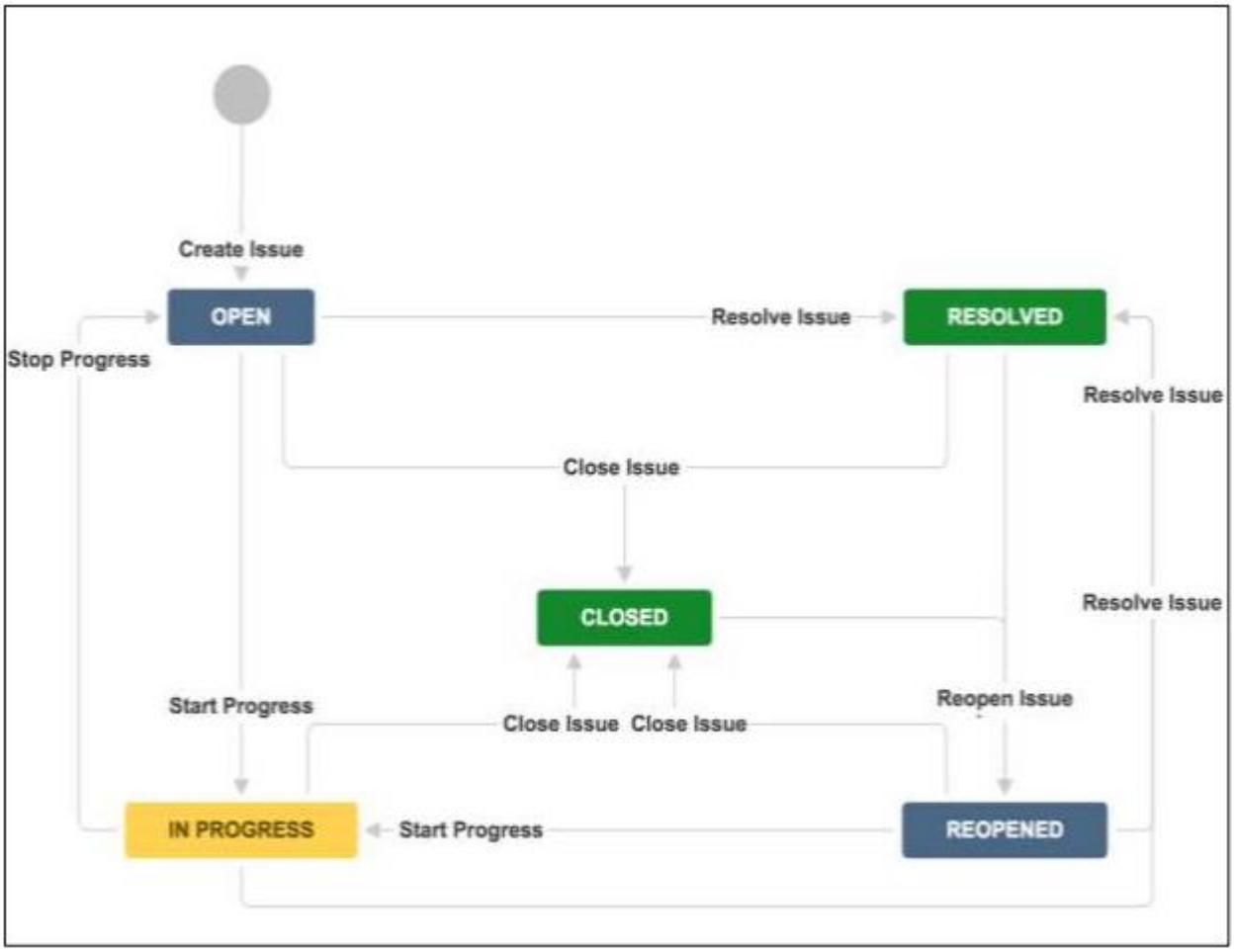
Example

As of now, an issue is created and opened. When the assignee starts working on the issue, the issue moves to the In Progress status. Here, the transition is starting the work, while the status of the issue is now progressive. JIRA workflow has the following stages to track as soon as an issue is created:

Open Issue: After creation, the issue is open and can be assigned to the assignee to

- Start working on it. In Progress Issue: The assignee has actively started to work on the issue.
- Resolved Issue: All sub-tasks and works of that Issue are completed. Now, the issue is waiting to be verified by the reporter. If verification is successful, it will be closed or re-opened, if any further changes are required.
- Reopened Issue: This issue was resolved previously, but the resolution was either incorrect or missed a few things or some modifications are required. From Reopened stage, issues are marked either as assigned or resolved.
- Close Issue: The issue is considered as finished, resolution is correct as of now. Closed issues can be re-opened later based on the requirement.

JIRA Workflow can be referred as a Defect Lifecycle. It follows the same concepts; the only difference is that it is generic for all issues rather than limited to Defects only.
The following diagram shows a standard workflow:



A transition is a one-way link, if an issue moves back and forth between two statuses; two transitions should be created. Example: There are two-way transitions between closed and re-opened statuses. A closed issue can be reopened if any modifications are required at any time until the project completes, while a re-opened issue can be closed directly if additional work is taken care in another issue and no specific work has been done on the re-opened issue.

IACSD

Session 4

Introduction to software testing

What is Software Testing?

- Software testing is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is Defect free
- Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements.

Done in two ways:
1. Manual
2. Automated

Why is Software Testing Important?

- Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, and history is full of such examples.

Ex:

- In April 2015, Bloomberg terminal in London crashed due to software glitch affected more than 300,000 traders on financial markets. It forced the government to postpone a 3bn pound debt sale.
- Nissan cars have to recall over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.

Principles of software testing

- **Testing shows presence of defects:**
  Testing shows presence of defects cannot prove absence of defects.
- **Exhaustive testing is not possible:**
  It is impossible to test all input combinations of data and scenarios. Smarter way of testing should be adopted.
- **Early testing:**
  Start testing earlier, it saves a lot of money rather than testing it later.

- **Defect clustering:**
  Equal distribution of the bugs across the modules is not possible; defects may be clustered in small piece of code/module
- **Pesticide paradox:**
  Executing same test case again and again will not help to find more bugs, Review them regularly and modify if changes required.
- **Testing is context dependent:**
  Different websites are tested differently. Ex: Banking sites are tested differently than Shopping websites
- **Absence of errors fallacy:**
  Finding and fixing many bugs does not help. If it fails to meet user's requirements. It is not useful.

Verification and validation

What is Verification?

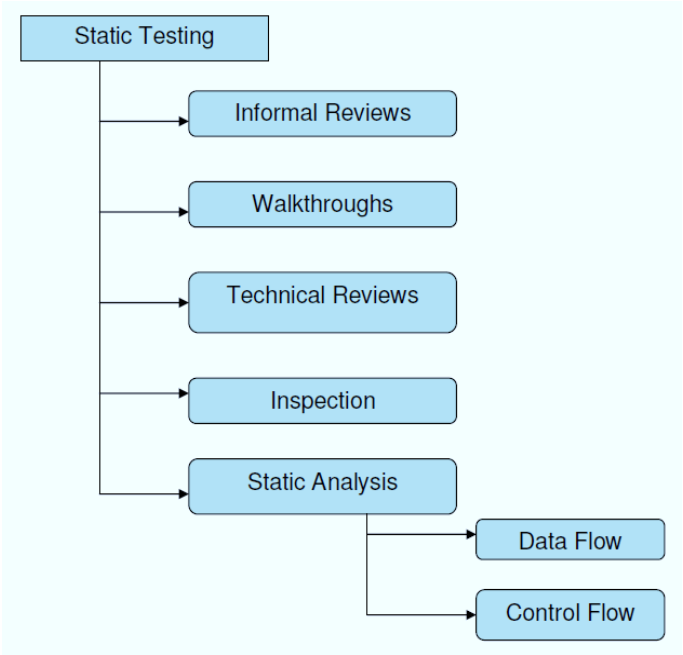
- ***Definition****: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.*
- Verification is a static practice of verifying documents, design, code and program. It includes all the activities associated with producing high quality software: inspection, design analysis and specification analysis. It is a relatively objective process.
- Verification will help to determine whether the software is of high quality, but it will not ensure that the system is useful. Verification is concerned with whether the system is well-engineered and error-free.
  Types of defects which can be easier to find during static testing are:
- Deviations from standards
- Non-maintainable code
- Design defects
- Missing requirements
- Inconsistent interface specifications

Methods of Verification: (Static Testing)

- *Walkthrough*
- *Inspection*
- *Review*

- **Walkthrough**: In this type of technique a meeting is led by author to explain the product. Participants can ask questions and a scribe is assigned to make notes.
- **Inspection**: Here the main purpose is to find defects. Code walkthroughs are conducted by moderator. It is a formal type of review where a checklist is prepared to review the work documents.
- **Technical reviews**: In this type of static testing a technical round of review is conducted to check if the code is made according to technical specifications and standards. Generally the test plans, test strategy and test scripts are reviewed here.

The overall static testing goes in the following manner:



IACSD

What is Validation?

- ***Definition***: *The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.*

- Validation is the process of evaluating the final product to check whether the software meets the customer expectations and requirements. It is a dynamic mechanism of validating and testing the actual product.

Testing is classified into three categories.

- Functional Testing
- Non-Functional Testing or Performance Testing
- Maintenance (Regression and Maintenance)

Functional Testing

Definition: Functional Testing is a type of Software Testing whereby the system is tested against the functional requirements/specifications.

Functions (or features) are tested by providing appropriate input and examining the output. The actual results are then compared with expected results.

Subtypes under this category:

Unit Testing
Integration Testing
Smoke
UAT (User Acceptance Testing)
Localization
Globalization
Interoperability
So on

- **Unit Testing**
  A ***unit*** is the smallest testable part of an application like functions, classes or procedures. ***Unit Testing*** is a ***Software Testing*** method by which individual units of source code, sets of one or more computer program modules are tested to determine whether they are fit for use. Unit tests are basically written and executed by software developers to make sure that code meets its design and requirements and behaves as expected.

- **Integration testing** is testing of a subsystem which comprises two or more integrating components. It is carried out once the individual components have been unit tested and they are working as expected. It is carried out with an objective to find defects in the interfaces and the interactions between the integrated components.

- ***Smoke Testing:*** It is performed after software build to ascertain that the critical functionalities of the program is working fine. It is executed "before" any detailed functional or regression tests are executed on the software build.

  What is build (After merging of several code file it turns into an executable file which is known as a ***Build***. In multiple agile cycles)
  "*Can tester able to access software application?*", "*Does user navigates from one window to other?*", "*Check that the GUI is responsive*"

- **Sanity (Stability)Testing:**
  The terminologies such as *Smoke Test* or ***Build Verification Test*** or ***Basic Acceptance Test*** or ***Sanity Test*** are interchangeably used, however, each one of them is used under a slightly different scenario. I

Sanity Testing is a *software testing technique* performed by the test team for some basic tests. Whenever a new build is received, after minor changes in code or functionality, *Sanity testing* is performed to ascertain that the bugs have been fixed.

Sanity is done by Tester.

Sanity is done for mature builds like build those are just going to hit production and have gone through multiple regression process.
Sanity can be removed from the testing process if regression is in the process of testing.

- **System testing** :
  is a testing level that evaluates the behavior of a fully integrated software system based on predetermined specifications and requirements. It is a solution to the question
  "***if the complete system works according to its predefined requirements?"***

- **Regression testing** :
  is the retesting of a software system to confirm that changes made to a few parts of the codes have not any side effects on existing system functionalities. It is to ensure that old codes are still working as they were before the introduction of the new change.

  The ideal process would be to create an extensive test suite and run it after each and every change.

  When to Perform Regression:
- *Any new feature or new functionality is added to the product.*
- *Any enhancement is done to previous functionality.*
- *Any defect is fixed*

- ***User Acceptance Testing – UAT***

- It is a type of testing performed by the Client to certify the system with respect to the requirements that were agreed upon. This testing happens in the final phase of testing before moving the software application to the Market or Production environment.
- ***User Acceptance Testing*** is also known as ***End-User Testing***, ***Acceptance Testing*** and ***Operational Acceptance Testing (OAT)***.

Types of User Acceptance Testing

- ***Alpha Testing*** *: Alpha Testing is done onsite therefore developers, as well as business analysts, are involved with the testing team.*
- ***Beta Testing :*** *Beta Testing is done at the client-side by the real users or customer, therefore developers and business analysts are not at all involved.*

What is Non-Functional Testing?

- Non-functional testing is defined as a type of Software testing to check non-functional aspects (performance, usability, reliability, etc) of a software application. It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.

Non Functional Testing Parameters:

Security, availability, efficiency, Integrity, Reliability Survivability, Usability, Flexibility, Scalability, Reusability, Interoperability Portability.

- **Security Testing**

  To test system is free from any vulnerabilities, threats, risks that may cause a big loss. Security testing of any system is about finding all possible loopholes and weaknesses of the system which might result into a loss of information, revenue, repute at the hands of the employees or outsiders of the Organization.
- **Reliability Testing**

Reliability testing is defined as a software testing type, that checks whether the software can perform a failure-free operation for a specified period of time in a specified environment.

Reliability means "yielding the same," in other terms, the word "reliable" mean something is dependable and that it will give the same outcome every time.

- **Usability Testing :**
  Is defined as a type of software testing where, a small set of target end-users, of a software system, "use" it to expose usability defects. This testing mainly focuses on the user's ease to use the application, flexibility in handling controls and the ability of the system to meet its objectives. It is also called User Experience (UX) Testing.

- There are many software applications/websites, which miserably fail, once launched, due to following reasons -
- Where do I click next?
- Which page needs to be navigated?
- Which Icon or Jargon represents what?
- Error messages are not consistent or effectively displayed
- Session time not sufficient.
- Software Engineering, Usability Testing identifies usability errors in the system early in the development cycle and can save a product from failure.

- **Scalability Testing:**
  Is defined as the ability of a network, system or a process to continue to function well when changes are done in the size or volume of the system to meet a growing need.

Scalability testing ensures that an application can handle the projected increase in user traffic, data volume, transaction counts frequency, etc.

Quality Assurance vs. Quality Control vs. Testing

Quality assurance

Is process oriented, it is all about preventing defects by ensuring the processes used to manage and create deliverables works. Not only does it work, but is consistently followed by the team. Moreover, QA is about engineering processes that assure quality is achieved in an effective and efficient way.

For instance, if a defect is found and fixed, there is no guaranteeing it won't pop back up. The role of QA is to identify the process that allowed the error to occur and re-engineer the system so that these defects won't appear for the second time. The QA process verifies that the product will continue to function as the customer expects.

Though QC is absolutely necessary, QA is perhaps more important. By the time you reach the QC stage, for instance, fixing bugs becomes an expensive issue. Because of that, focusing efforts on improved QA processes is one of the best investments an organization can make.

Examples of QA include process definition and implementation, training, audits and selection of tools.

Quality control, alternatively, is product oriented. It is the function of software quality that determines the ending result is what was expected. Whereas QA is proactive, QC is reactive. QC detects bugs by inspecting and testing the product. This involves checking the product against a predetermined set of requirements and validating that the product meets those requirements.

Examples of QC include technical reviews, software testing and code inspections.

Testing is a subset of QC. It is the process of executing a system in order to detect bugs in the product so that they get fixed. Testing is an integral part of QC as it helps demonstrate that the product runs the way it is expected and designed for.

To summarize, think of everything as an assembly line. QA can be thought of as the process to ensure the assembly line actually works, while QC is when the products coming off the assembly line are checked to verify they meet the required specifications.

Ultimately, both QA and QC are required for ensuring a successful product. When used together, they can help detect inefficient processes and identify bugs in the product. Moreover, QA and QC can help to develop and deliver a consistently high-quality product to your customers.

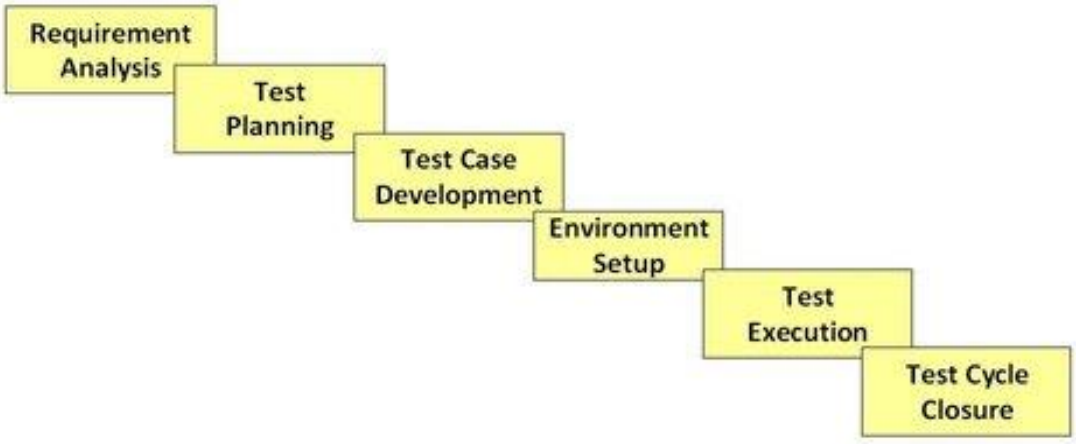
#	Quality Assurance	Quality Control	Testing
1	Activities which ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements.	Activities which ensure the verification of developed software with respect to documented (or not in some cases) requirements.	Activities which ensure the identification of bugs/error/defects in the Software.
2	Focuses on processes and procedures rather than conducting actual testing on the system.	Focuses on actual testing by executing Software with intend to identify bug/defect through implementation of procedures and process.	Focuses on actual testing.
3	Process oriented activities.	Product oriented activities.	Product oriented activities.
4	Preventive activities.	It is a corrective process.	It is a corrective process.
5	It is a subset of Software Test Life Cycle (STLC).	QC can be considered as the subset of Quality Assurance.	Testing is the subset of Quality Control.

IACSD

Introduction to STLC

Software Testing Life Cycle

Software Testing Life Cycle (STLC) is defined as a sequence of activities conducted to perform Software Testing.
Contrary to popular belief, Software Testing is not a just a single activity. It consists of a series of activities carried out methodologically to help certify your software product.



- Each of these stages has a definite Entry and Exit criteria, Activities & Deliverables associated with it.**What is Entry and Exit Criteria?**

- **Entry Criteria:** Entry Criteria gives the prerequisite items that must be completed before testing can begin.
- **Exit Criteria:** Exit Criteria defines the items that must be completed before testing can be concluded

Requirement Analysis:

- During this phase, test team studies the requirements from a testing point of view to identify the testable requirements.
- The QA team may interact with various stakeholders (Client, Business Analyst, Technical Leads and System Architects etc) to understand the requirements in detail.
- Requirements could be either Functional (defining what the software must do) or Non Functional (defining system performance /security availability)

Activities

- Identify types of tests to be performed.
- Gather details about testing priorities and focus.
- Prepare Requirement Traceability Matrix (RTM).
- Identify test environment details where testing is supposed to be carried out.
- Automation feasibility analysis (if required).
- **Deliverables**
- RTM
- Automation feasibility report. (if applicable)

Test Planning

- Typically, in this stage, a Senior QA manager will determine effort and cost estimates for the project and would prepare and finalize the Test Plan. In this phase, Test Strategy is also determined.

Activities

- Preparation of test plan/strategy document for various types of testing
- Test tool selection
- Test effort estimation
- Resource planning and determining roles and responsibilities.
- Training requirement
- Deliverables

- Test plan /strategy document.
- Effort estimation document

Test Environment Setup

- Test environment decides the software and hardware conditions under which a work product is tested. Test environment set-up is one of the critical aspects of testing process and *can be done in parallel with Test Case Development Stage*. *Test team may not be involved in this activity* if the customer/development team provides the test environment in which case the test team is required to do a readiness check (smoke testing) of the given environment.

Activities

- Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
- Setup test Environment and test data
- Perform smoke test on the build
- Deliverables
- Environment ready with test data set up
- Smoke Test Results.

Test Execution

- During this phase, the testers will carry out the testing based on the test plans and the test cases prepared. Bugs will be reported back to the development team for correction and retesting will be performed.

Activities

- Execute tests as per plan
- Document test results, and log defects for failed cases
- Map defects to test cases in RTM
- Retest the Defect fixes
- Track the defects to closure
- Deliverables
- Completed RTM with the execution status

- Test cases updated with results
- Defect reports

Test Cycle Closure

- Testing team will meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in the future, taking lessons from the current test cycle. The idea is to remove the process bottlenecks for future test cycles and share best practices for any similar projects in the future.

- **Activities**
- Evaluate cycle completion criteria based on Time, Test coverage, Cost, Software, Critical Business Objectives, Quality
- Prepare test metrics based on the above parameters.
- Document the learning out of the project
- Prepare Test closure report
- Qualitative and quantitative reporting of quality of the work product to the customer.
- Test result analysis to find out the defect distribution by type and severity.
- Deliverables
- Test Closure report
- Test metrics

What is Requirement Traceability Matrix?

- Requirement Traceability Matrix (RTM) is a document that maps and traces user requirement with test cases. It captures all requirements proposed by the client.
- The main purpose of Requirement Traceability Matrix is to validate that all requirements are checked via test cases such that no functionality is unchecked during Software testing.

IACSD

RTM parameters

- Requirement ID
- Requirement Type and Description
- Test Cases with Status

Template RTM:

Req No	Req Desc	Testcase ID	Status
123	Login to the application	TC01,TC02,TC03	TC01-Pass TC02-Pass
345	Ticket Creation	TC04,TC05,TC06, TC07,TC08,TC09 TC010	TC04-Pass TC05-Pass TC06-Pass TC06-Fail TC07-No Run
456	Search Ticket	TC011,TC012, TC013,TC014	TC011-Pass TC012-Fail TC013-Pass TC014-No Run

In typical software testing project, the traceability matrix would have more than these parameters. Sample

Sno	Req ID	Req Desc	TC ID	TC Desc	Test Design	Test Designer	UAT Test Req?	Test Execution			Defects?	Defect ID	Defect Status	Req Coverage Status
								Test Env	UAT Env	Prod Env				
1	Req01	Login to the Application	TC01	Login with Invalid Username and valid password	Completed	XYZ	No	Passed	No Run	No Run	None	None	N/A	Partial
2			TC02	Login with Valid Username and invalid password	Completed	YZA	No	Passed	No Run	No Run	None	None	N/A	Partial
3			TC03	Login with valid credentials	Completed	XYZ	Yes	Passed	Passed	No Run	Yes	DFCT001	Test OK	Partial

Sheet1 Sheet2 Sheet3

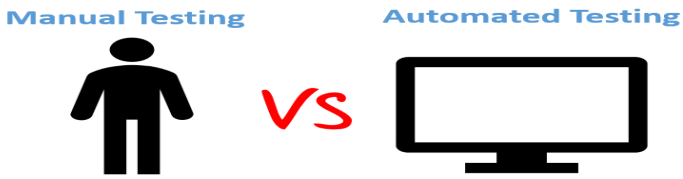
Manual Testing:

- Manual testing is testing of the software where tests are executed manually by a QA Analysts. It is performed to discover bugs in software under development.
- In Manual testing, the tester checks all the essential features of the given application or software. In this process, the software testers execute the test cases and generate the test reports without the help of any automation software testing tools.

Automation Testing:

- In Automated Software Testing, testers write code/test scripts to automate test execution. Testers use appropriate automation tools to develop the test scripts and validate the software. The goal is to complete test execution in a less amount of time.
- Automated testing entirely relies on the pre-scripted test which runs automatically to compare actual result with the expected results. This helps the tester to determine whether or not an application performs as expected. Automated testing allows you to execute repetitive task and regression test without the intervention of manual tester. Even though all processes are

performed automatically, automation requires some manual effort to create initial testing scripts.



KEY DIFFERENCE

- Manual Testing is done manually by QA analyst (Human) whereas Automation Testing is done with the use of script, code and automation tools (computer) by a tester.
- Manual Testing process is not accurate because of the possibilities of human errors whereas the Automation process is reliable because it is code and script based.
- Manual Testing is a time-consuming process whereas Automation Testing is very fast.
- Manual Testing is possible without programming knowledge whereas Automation Testing is not possible without programming knowledge.
- Manual Testing allows random Testing whereas Automation Testing doesn't allow random Testing.

WHITE BOX TESTING

- (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing)
- It is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential.
  Definition by ISTQB
  (International Software Testing Qualifications Board)

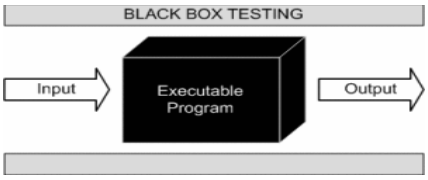
- **White-box testing:** Testing based on an analysis of the internal structure of the component or system.
- **White-box test design technique:** Procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.

- White Box Testing method is applicable to the following levels of software testing:

- Unit Testing: For testing paths within a unit.
- Integration Testing: For testing paths between units.
- System Testing: For testing paths between subsystems.
- However, it is mainly applied to Unit Testing.

BLACK BOX TESTING

- It is also known as Behavioral Testing, is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.



- This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:
- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behavior or performance errors
  Initialization and termination errors
  Definition by ISTQB
- **Black box testing:** Testing, either functional or non-functional, without reference to the internal structure of the component or system.
- **Black box test design technique:** Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.
- Black Box Testing method is applicable to the following levels of software testing:
- Integration Testing
- System Testing

- Acceptance Testing

GRAY BOX TESTING

- It is a software testing method which is a combination of Black Box Testing method and White Box Testing method
- In Gray Box Testing, the internal structure is partially know
- Levels Applicable To
- Though Gray Box Testing method may be used in other levels of testing, it is primarily used in Integration Testing.

Session 5

Introduction to Selenium

SELENIUM

Is a free (open-source) automated testing framework used to validate web applications across different browsers and platforms. You can use multiple programming languages like Java, C#, Python etc to create Selenium Test Scripts. Testing done using the Selenium tool is usually referred to as Selenium Testing.

Selenium Software is not just a single tool but a suite of software, each piece catering to different testing needs of an organization. Here is the list of tools

- Selenium Integrated Development Environment (IDE)
- Selenium Remote Control (RC)
- WebDriver

- Selenium Grid

Features of Selenium

- Selenium is an open source and portable Web testing Framework.
- Selenium IDE provides a playback and record feature for authoring tests without the need to learn a test scripting language.
- It can be considered as the leading cloud-based testing platform which helps testers to record their actions and export them as a reusable script with a simple-to-understand and easy-to-use interface.
- Selenium supports various operating systems, browsers and programming languages. Following is the list:
  - Programming Languages: C#, Java, Python, PHP, Ruby, Perl, and JavaScript
  - Operating Systems: Android, iOS, Windows, Linux, Mac, Solaris.
  - Browsers: Google Chrome, Mozilla Firefox, Internet Explorer, Edge, Opera, Safari, etc.
- It also supports parallel test execution which reduces time and increases the efficiency of tests.
- Selenium can be integrated with frameworks like Ant and Maven for source code compilation.
- Selenium can also be integrated with testing frameworks like TestNG for application testing and generating reports.
- Selenium requires fewer resources as compared to other automation test tools.
- WebDriver API has been indulged in selenium which is one of the most important modifications done to selenium.
- Selenium web driver does not require server installation, test scripts interact directly with the browser.
- Selenium commands are categorized in terms of different classes which make it easier to understand and implement.
- Selenium Remote Control (RC) in conjunction with WebDriver API is known as Selenium 2.0. This version was built to support the vibrant web pages and Ajax.

Selenium WebDriver

Selenium WebDriver is the most important component of Selenium Tool's Suite. The latest release "Selenium 2.0" is integrated with WebDriver API which provides a simpler and more concise programming interface.

In WebDriver, test scripts can be developed using any of the supported programming languages and can be run directly in most modern web browsers. Languages supported by WebDriver include C#, Java, Perl, PHP, Python and Ruby.

Selenium WebDriver performs much faster as compared to Selenium RC because it makes direct calls to the web browsers. RC on the other hand needs an RC server to interact with the browser.

WebDriver has a built-in implementation of Firefox driver (Gecko Driver). For other browsers, you need to plug-in their browser specific drivers to communicate and run the test. Most commonly used WebDriver's include:

- Google Chrome Driver
- Internet Explorer Driver
- Opera Driver
- Safari Driver
- HTML Unit Driver (a special headless driver)

Selenium WebDriver- Features

- **Multiple Browser Support**: Selenium WebDriver supports a diverse range of web browsers such as Firefox, Chrome, Internet Explorer, Opera and many more. It also supports some of the non-conventional or rare browsers like HTMLUnit.
- **Multiple Languages Support**: WebDriver also supports most of the commonly used programming languages like Java, C#, JavaScript, PHP, Ruby, Pearl and Python. Thus, the user can choose any one of the supported programming language based on his/her competency and start building the test scripts.
- **Speed**: WebDriver performs faster as compared to other tools of Selenium Suite. Unlike RC, it doesn't require any intermediate server to communicate with the browser; rather the tool directly communicates with the browser.

- **Simple Commands**: Most of the commands used in Selenium WebDriver are easy to implement. For instance, to launch a browser in WebDriver following commands are used:
  WebDriver driver = **new FirefoxDriver();** (Firefox browser)
  WebDriver driver = **new ChromeDriver();** (Chrome browser)
  WebDriver driver = **new InternetExplorerDriver();** (Internet Explorer browser)
- **WebDriver- Methods and Classes**: WebDriver provides multiple solutions to cope with some potential challenges in automation testing.
  WebDriver also allows testers to deal with complex types of web elements such as checkboxes, dropdowns and alerts through dynamic finders.

Installation help:

Selenium WebDriver installation process is completed in four basic steps:

1. Download and Install Java 8 or higher version.
2. Download and configure Eclipse or any Java IDE of your choice.
3. Download Selenium WebDriver Java Client (GecoDriver-Firefox,Chrome,Driver-Chrome)
4. Configure Selenium WebDriver:

   First 3 steps are easy and you might be having already Java and Eclipse, so simply last step you have to follow to configure Selenium WebDriver as follows:

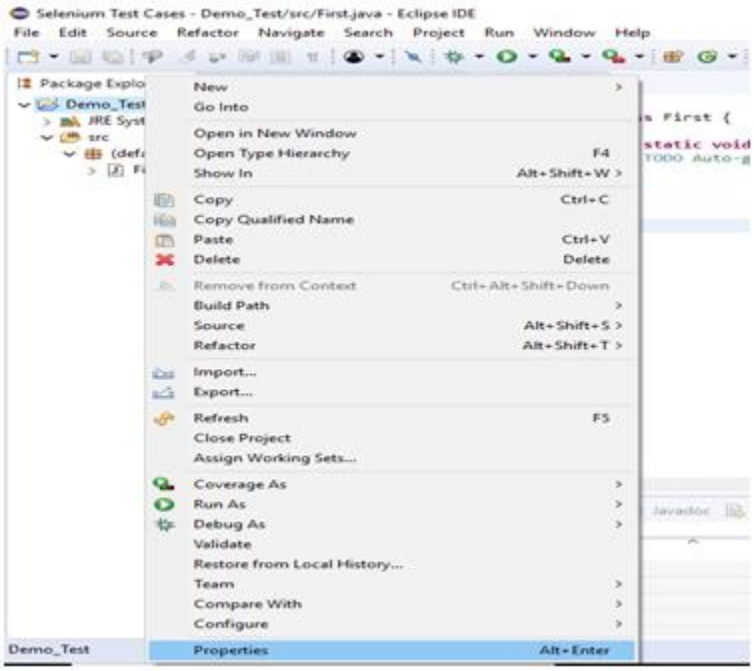
- Open URL: https://docs.seleniumhq.org/download/
  It will redirect you to the "downloads page" of Selenium official website.
- Scroll down through the web page and locate **Selenium Client & WebDriver Language Bindings**.
- Click on the "Download" link of Java Client Driver as shown in the image given below.

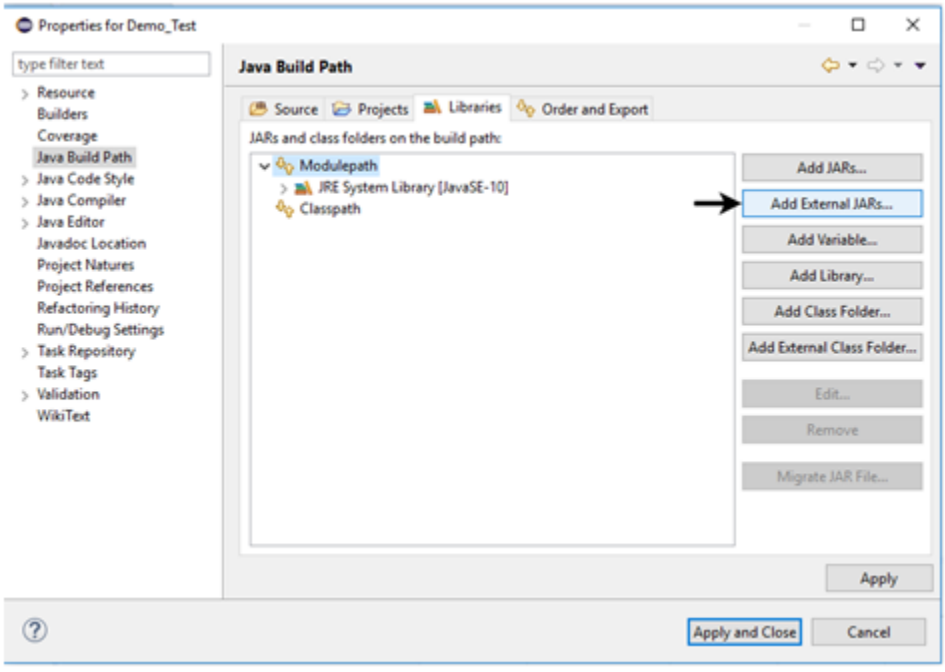
LANGUAGE	STABLE VERSION	RELEASE DATE	ALPHA VERSION	ALPHA RELEASE DATE	LINKS
Ruby	3.142.6	October 04, 2019	4.0.0alpha6	May 28, 2020	Download Alpha Download Changelog API Docs
Java	3.141.59	November 14, 2018	4.0.0-alpha-6	May 29, 2020	Download Alpha Download Changelog API Docs
Python	3.141.0	November 01, 2018	4.0.0a6.post1	May 28, 2020	Download Alpha Download Changelog API Docs
C#	3.14.0	August 02, 2018	4.0.0-alpha05	March 18, 2020	Download Alpha Download Changelog API Docs
JavaScript	3.6.0	October 06, 2017	4.0.0-alpha.7	March 05, 2020	Download Alpha Download Changelog API Docs

- The downloaded file would be in zipped format. Unpack the contents in a convenient directory. It contains the essential jar files required to configure Selenium WebDriver in Eclipse IDE.
- Launch Eclipse IDE
- Create a new Java Project from **File > New > Java Project**.

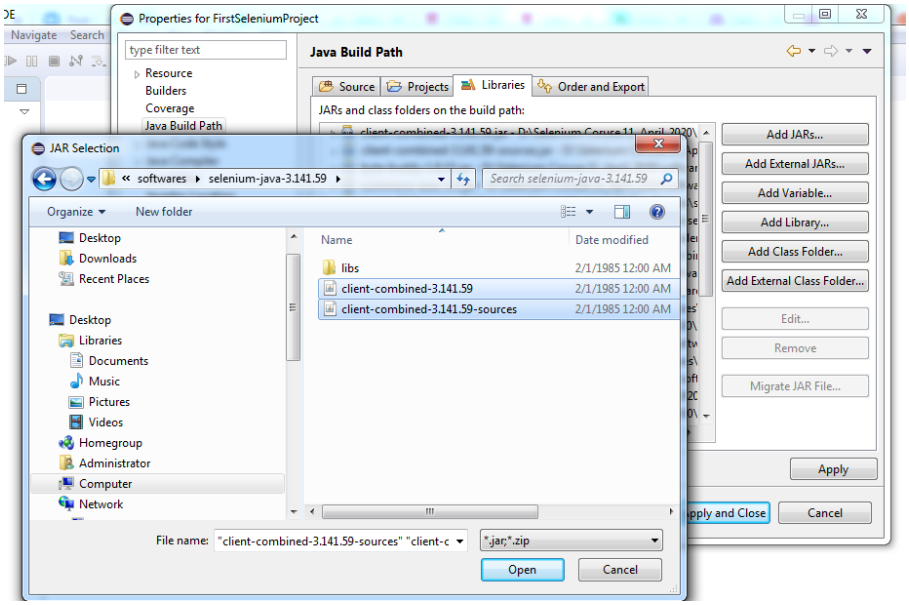
Now, we will add the Selenium jar files in our Test Suite (Demo_Test – Java Project).

- Right click on "Demo_Test" folder and select Properties.
- It will launch the Properties window for our "Demo_Test" Test Suite.
- Click on "Java Build Path" option from the left hand side panel.

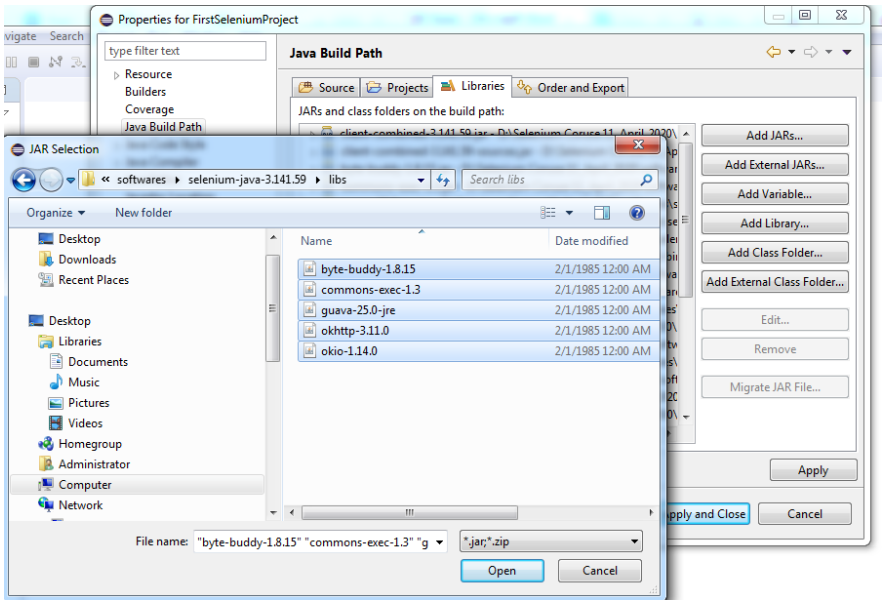




- Locate the directory where you have downloaded the Selenium jar files, select the respective jars and click on "Open" button.



- Repeat the same steps for the jars which are present under the "libs" folder.
- Open "libs" folder, select all of the respective jar files and click on "Open" button.

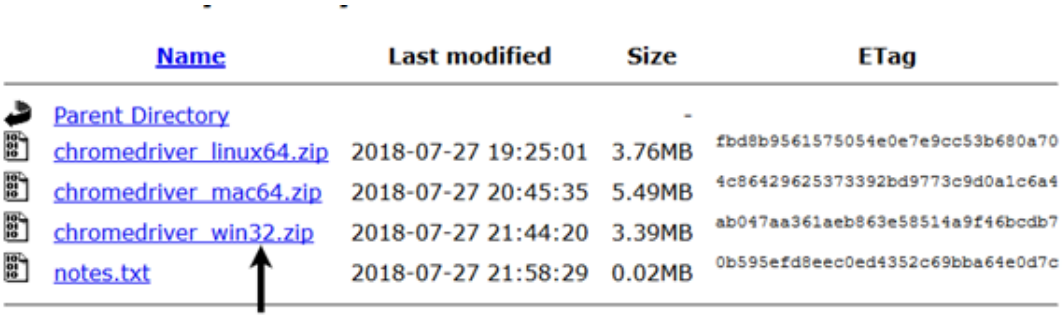


- Once you get all the Selenium jar files in your Libraries tab, click on Apply and Close button.

After That

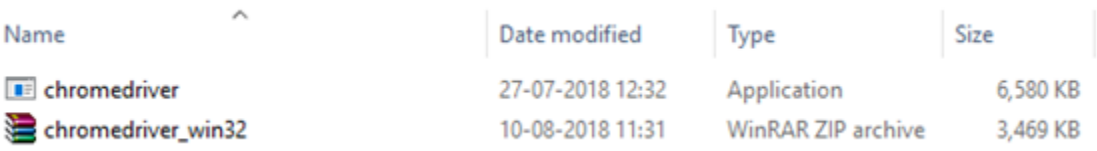
Open
URL: https://sites.google.com/a/chromium.org/chromedriver/downloads in your browser.

- Click on the "ChromeDriver 2.41" link. It will redirect you to the directory of ChromeDriver executables files. Download as per the operating system you are currently on.



Name	Last modified	Size	ETag
Parent Directory		-	
chromedriver_linux64.zip	2018-07-27 19:25:01	3.76MB	fbd8b9561575054e0e7e9cc53b680a70
chromedriver_mac64.zip	2018-07-27 20:45:35	5.49MB	4c86429625373392bd9773c9d0a1c6a4
chromedriver_win32.zip	2018-07-27 21:44:20	3.39MB	ab047aa361aeb863e58514a9f46bcdb7
notes.txt	2018-07-27 21:58:29	0.02MB	0b595efd8eec0ed4352c69bba64e0d7c

- For windows, click on the "chromedriver_win32.zip" download.



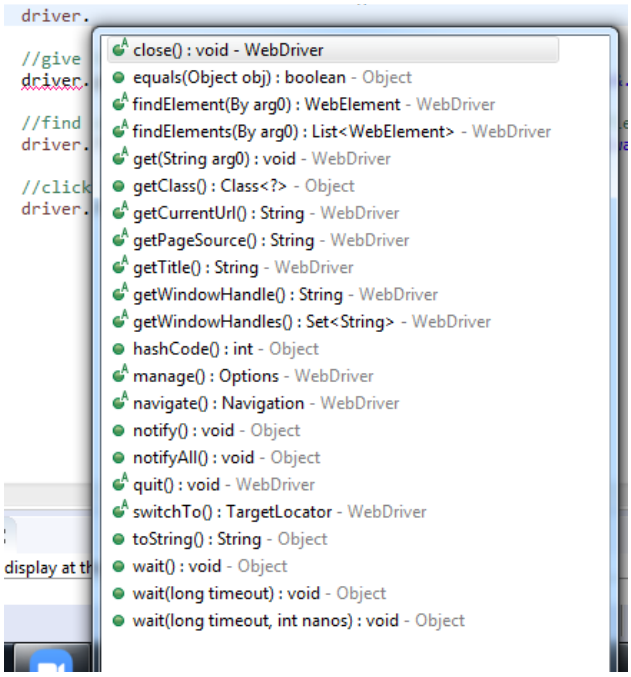
Name	Date modified	Type	Size
chromedriver	27-07-2018 12:32	Application	6,580 KB
chromedriver_win32	10-08-2018 11:31	WinRAR ZIP archive	3,469 KB

- The downloaded file would be in zipped format. Unpack the contents in a convenient directory.

Then We can Write The first Test Script in Java Class:
Ex:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class FirstSeleniumProgram {

	public static void main(String[] args) {

	//1. add jar of selenium
	//2. open browser,before that tell where is the chrome driver using
	 System.setProperty
System.setProperty("webdriver.chrome.driver","D:\\Selenium\\softwares\\chromedriver_win32\\chromedriver.exe");
		WebDriver driver=new ChromeDriver();
		//give the URL in get () to open the site
driver.get("https://login.yahoo.com/config/login?.src=fpctx&.intl=in&.lang=en-IN&.done=https://in.yahoo.com");
		//find any element from the web page and send text to the field
		driver.findElement(By.id("login-username")).sendKeys("pritiwadpalli");

		//click on next button
		driver.findElement(By.id("login-signin")).click();


	}

}
```

One possible way to view the methods provided by WebDriver is to open the Eclipse IDE loaded with Selenium Webdriver jar files, create a driver object for WebDriver and press the dot key. It will show you all of the possible methods provided by WebDriver.



The commands provided by Selenium WebDriver can be broadly classified in following categories:

1. Browser Commands
2. Navigation Commands
3. WebElement Commands

Browser Commands:

most commonly used Browser commands for Selenium WebDriver.
Get Command **get(String arg0) : void**
Get Title Command: **getTitle(): String**
Get Current URL Command:**getCurrentUrl(): String**
Get Page Source Command: **getPageSource(): String**
Close Command: **close():** void
Quit Command:**quit():** void

Navigation Commands:
WebDriver provides some basic Browser Navigation Commands that allows the browser to move backwards or forwards in the browser's history

```
driver.navigate().to("www.javatpoint.com");
driver.navigate().forward();
driver.navigate().back();
driver.navigate().refresh();
```

WebElement Commands

What is Web Element?

The term web element refers to a HTML element. The HTML documents are composed of HTML elements. It consists **a start tag**, **an end tag** and the **content** in between.

Given are some of the most commonly used WebElement commands for Selenium WebDriver.

clear() : **void**
sendKeys(CharSequence? KeysToSend) : **void**
click() : **void**
isDisplayed() : **boolean**
isEnabled() : **boolean**

IACSD

isSelected() : **boolean**

submit() : **void**

getText() : String

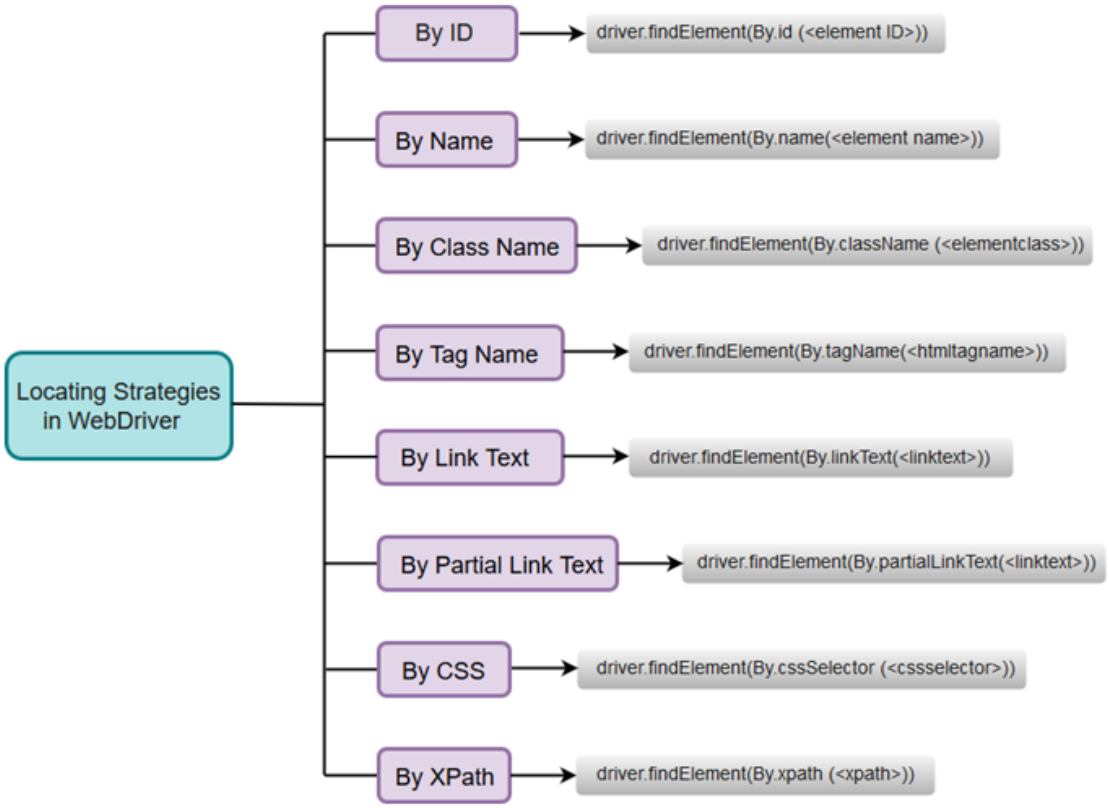
getTagName() : String

getCssvalue() : String

Locating Strategies

WebDriver uses locating strategies for specifying location of a particular web element.

Since, we are using WebDriver with java; each locating strategy has its own command in Java to locate the web elements.



By Id:

locate a particular web element using the value of its id attribute.

The Java Syntax for locating a web element using its id attribute is written as:

driver.findElement(By.id (<element ID>))

Ex: suppose we want to locate the text box whose id is fname as shown bellow

IACSD

```
<div class="col-md-12" style="font-size:15px;">
  <p>
    <b>TextBox :</b>
    <input id="fname" name="firstName" type="text">
  </p>
</div>
```

driver.findElement(By.id("fname"));

By Name:

locate a particular web element using the value of its "name" attribute.
The Java Syntax for locating a web element using its name attribute is written as:
driver.findElement(By.name(<element ID>)) ;
Ex: suppose we want to locate the text box whose name firstName is as shown bellow

```
<p>
  <b>TextBox :</b>
  <input id="fname" name="firstName" type="text">
</p>
```

driver.findElement(By.name("firstName"));

By TagName

locate a particular web element using its Tag Name.
The Java Syntax for locating a web element using its Tag Name is written as:
driver.findElement(By.tagName (<htmltagname>))
Ex:To locate a text box using its tagname
driver.findElement(By.tagName("input"));

By Class Name:

locate a particular web element using the value of its Class attribute.
The Java Syntax for locating a web element using its Class attribute is written as:
driver.findElement(By.className (<element class>))
Ex: To locate Automation testing checkbox by its class

```
<form action="#">
    <input type="checkbox" class="Automation" value="Automation"> ==
    " Automation Testing
                          "
    <input type="checkbox" class="Performance" value="Performance">
```

driver.findElement(By.className("Automation"));

IACSD

By XPath

Xpath is very popular locator strategy to perform operations on WebElements. Most of the time we need to create xpaths by our own.
Xpaths are of two types “**Relative/partial xpath**” & “**Absolute/complete xpath**” .

Relative/partial XPath method

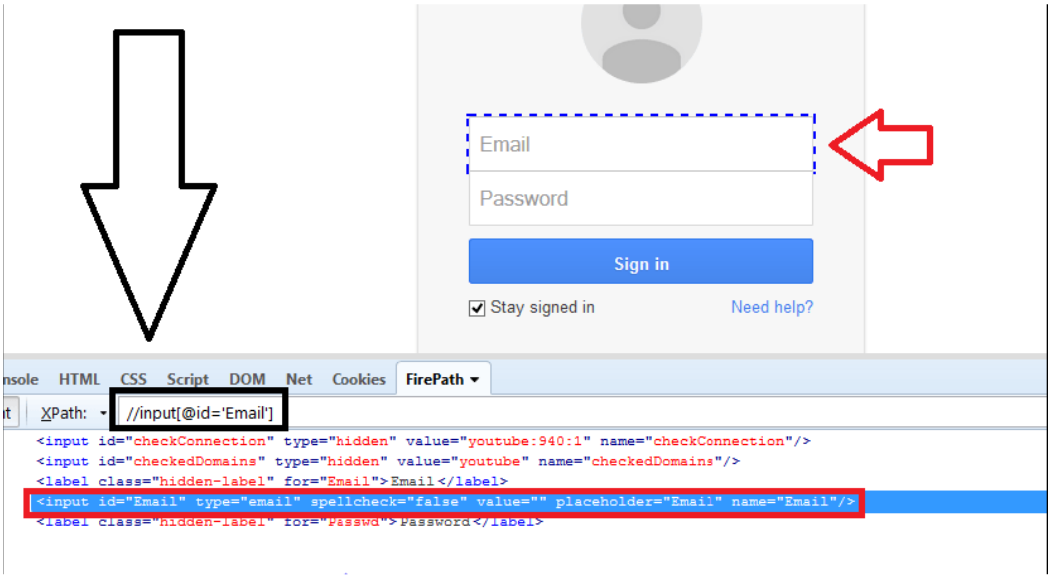
A relative xpath is one where the path starts from the node of your choise - it doesn't need to start from the root node. It starts with Double forward slash(//).

Using single attribute

Syntax :- // tagname[@attribute-name=’value1’+

Examples:-

// a[@href=’http://www.google.com’]
//input[@id=’name’]
//input[@name=’username’]
//img[@alt=’sometext’]

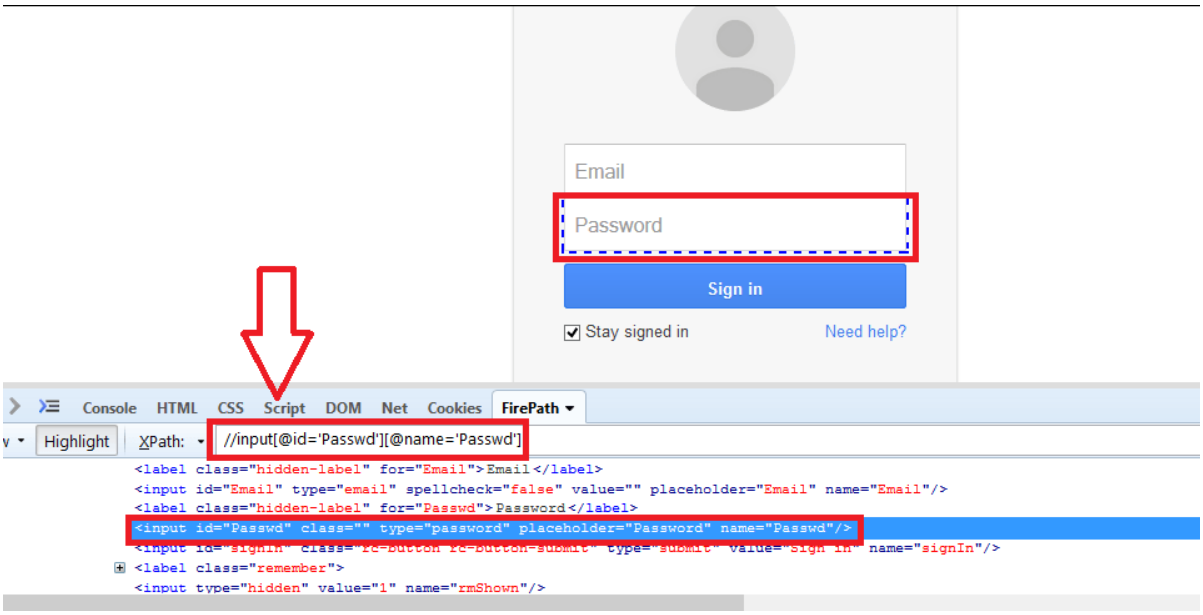


Using multiple attribute

Syntax :- //tagname[@attribute1='value1'][@attribute2='value2']

Examples :-

//a[@id='id1'][@name='namevalue1']
//img[@src=''+][@href='']



Using contains method

Syntax :- //tagname[contains(@attribute,'value1')]

Examples:-
//input[contains(@id,'')]
//input[contains(@name,'')]

//a[contains(@href,'')]
//img[contains(@src,'')]
//div[contains(@id,'')]

Using starts-with method

Syntax:- //tagname[starts-with(@attribute-name,'')]

Examples:-

//a[starts-with(@href, '')]
//img[starts-with(@src, '')]
//div[starts-with(@id, '')]
//input[starts-with(@id, '')]
//button[starts-with(@id, '')]

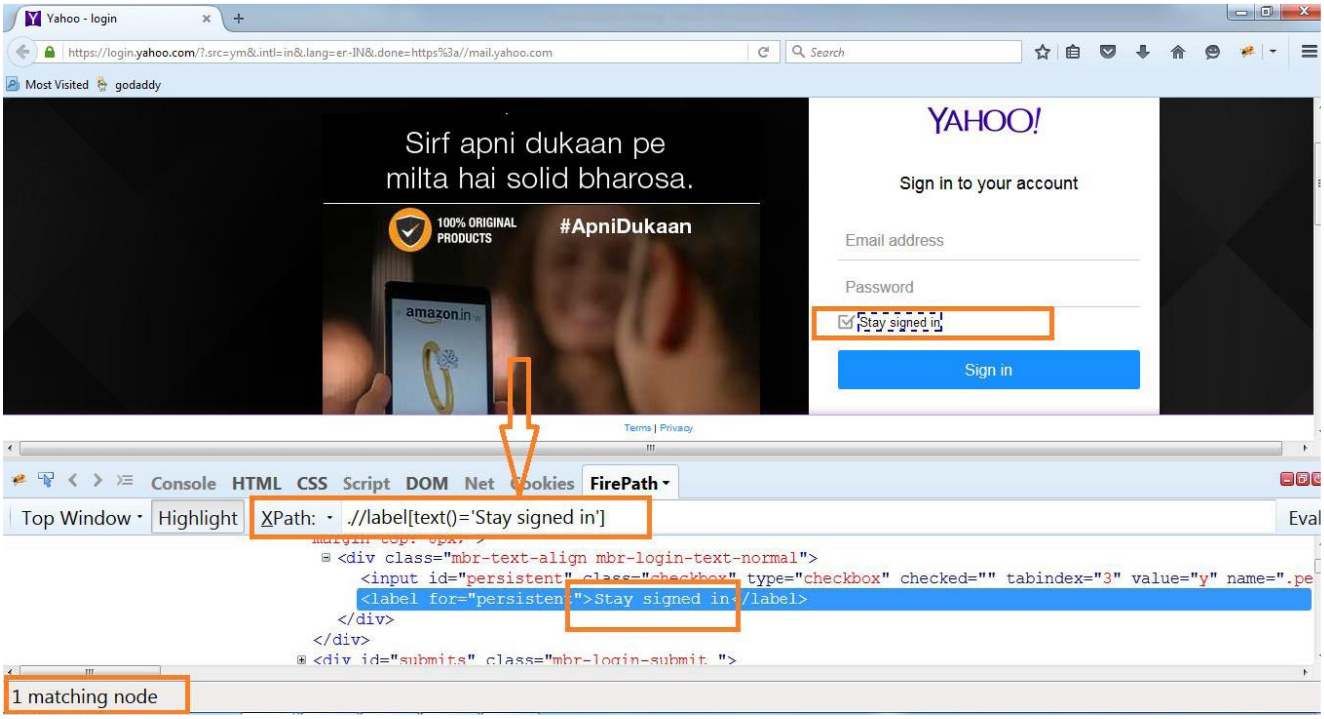
Using text method

Syntax:- //tagname[text()='text we are searching for']

Examples:-

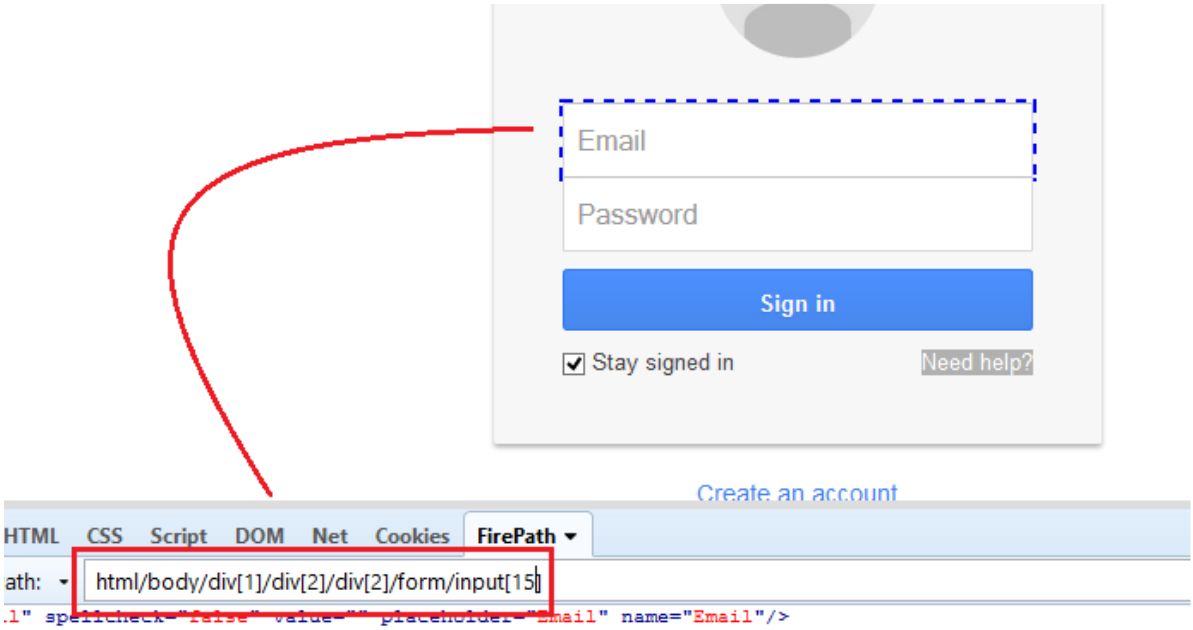
//div[text()='']
//label[text()='']
//a[text()='']
//p[text()='']

IACSD



Absolute/complete XPath method

Absolute XPath starts with the root node or a forward slash (/). The advantage of using absolute is, it identifies the element very fast. Disadvantage here is, if anything goes wrong or some other tag added or deleted in between, then this path will no longer works. So they are called Fragile xpaths.



Add interactions

Way to deal with ***CheckBox & Radio Button is*** exactly the same. The main difference between radio button and checkbox is checkbox you can select multiple but for radio button, only one selection is possible.

Radio button and Checkbox most of the time will have tag name “input” and one of the attribute as “name”. Before performing click action on them, sometimes we need to verify couple of other things such as .

- Verify whether radio button or checkbox is enabled.
- Verify whether radio button or checkbox is Displayed on UI or not.
- Verify whether checkbox and radio button is default selected/checked or not.

We can verify above things with the help of predefined methods such as

1> Verify whether radio button or checkbox is enabled :- **isDisplayed();**
2> Verify whether radio button or checkbox is Displayed on UI or not :- **isEnabled();**
3> Verify whether checkbox and radio button is default selected or not :- **isSelected();**

Note :- These methods returns Boolean (true or false)
In case of CheckBox it is always recommended to check if that is already in selected or deselected. Because, if it is already selected and when you click on the same element it will get deselected.
In case of Radio button it is not required as even though it is by default selected, clicking again on same will not deselect it.

isSelected() method can be used to identify whether the checkbox or radio button is selected or not
Example:-

```
System.out.println(radios.get(0).isSelected()); //is Selected or not
```

Example:- Radio button

```
import java.util.List;
import org.openqa.selenium.By;
```

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class Radios {
public static void main(String[] args) {
WebDriver driver=new ChromeDriver();
driver.get("https://www.facebook.com/");
List<WebElement> radios = driver.findElements(By.name("sex")); // all radio
System.out.println("Total radio buttons -> "+ radios.size()); // total no of radio
System.out.println(radios.get(0).isDisplayed()); //male radio button is displayed or not
System.out.println(radios.get(0).isEnabled()); // male radio button is Enabled or not
System.out.println(radios.get(0).isSelected()); // male radio button is Selected or not
radios.get(1).click(); // male radio button will be selected
}
}
```

```
public class CheckBox {
public static void main(String[] args) {
WebDriver driver = new FirefoxDriver();
driver.get("https://www.facebook.com/");
// ”keep me logged in ” check box is displayed or not
System.out.println(driver.findElement(By.xpath("//*[@id='persist_box']")).isDisplayed()
);
// ”keep me logged in ” check box is enabled or not
System.out.println(driver.findElement(By.xpath("//*[@id='persist_box']")).isEnabled());
// ”keep me logged in ” check box is selected or not
System.out.println(driver.findElement(By.xpath("//*[@id='persist_box']")).isSelected())
;
// click on ”keep me logged in ” check box
driver.findElement(By.xpath("//*[@id='persist_box']")).click();
}
}
```

DropDown list

Dropdown list is also called as list box.

Dropdown list of countries will look like below.

IACSD

Combo box is a combination of a drop-down list or list box and a single-line editable textbox, allowing user to either type a value directly or select a value from the list.
It looks like below.

Rules

1. Every drop down will have tagName as **"Select"**
2. Every element in droplist will have tag **"option"**

How to access the elements inside droplist

We will be using **"Select"** class to deal with drop down list
To use Select class we need to import **"org.openqa.selenium.support.ui.Select"**

Example

Let's use yahoo website for this exercise
Go to yahoo.com --> Mail --> sign up for new account --> "Month" dropdown mentioned in front of Birthday

Steps

1. Get the "xpath" of the "droplist"

```
WebElement month_dropdown=driver.findElement(By.xpath("//*[@id='month']"));
```

2. Send it to the "constructor" of select class

```
Select month=new Select(month_dropdown);
```

3. Use all the functions using "object" of Select class

Few commands:

month.getFirstSelectedOption().getText() ----> gives text of default selected option from drop list
month.selectByVisibleText("April"); ----------> select any element using text from droplist
month.selectByIndex(3); ----------> select any element using index. Index starts from "0".
month.selectByValue("4"); ----------> Value is attribute of option tag. Its text
month.getOptions().size() ----------> total number of elements in the droplist

Example:-
// Go to yahoo.com --> Mail --> sign up for new account --> "Month" dropdown mentioned in front of Birthday

```
WebElement month_dropdown=driver.findElement(By.xpath("//*[@id='month']"));
Select month=new Select(month_dropdown);
// Select value using Index
month.selectByIndex(3);
 //Select value using value attribute.
month.selectByValue("4");
//Select value from Visible text
month.selectByVisibleText("April");
 // Get Selected option from Dropdown. WebElement
first_value=month.getFirstSelectedOption(); String value=first_value.getText();
// Get All option from dropdown
List<WebElement> dropdown=month.getOptions();
for(int i=0;i<dropdown.size();i++)
{
String drop_down_values=dropdown.get(i).getText();
System.out.println("dropdown values are "+ drop_down_values);
}
```


IACSD

Selecting multiple elements from drop list.

month.isMultiple( ) ----------> tells whether the droplist supports multiple selecting options at the same time or not
(Note :- our yahoo example does not support multiselect, please use any other drop list)
There is no additional logic behind selecting multiple options of Select element.
All you need to do is to fire select commands on multiple elements one by one that's it.

```
month.selectByIndex(Index)
month.selectByIndex(Index)
// or can be used as
month.selectByVisibleText(text)
month.selectByVisibleText(text)
// or can be used as
month.selectByValue(value)
month.selectByValue(value)
```

Handling dropdown list without using "Select" class

We can work with drop down list without using Select class as well. Here is how we do it.

```
WebElement month_dropdown=driver.findElement(By.xpath("//*[@id='month']"));
month_dropdown.sendKeys("April"); // April will be selected from dropdown list
```

IACSD

Mouse simulation using Actions class

While automating the application we encounter the scenario where we need to hover on some menu and it display further more options, which is common case with e-commerce website.

Example: - if you go to flipkart and mouse hover on a category called “Electronics”, it will display lot many options.

To do all the mouse simulation we use “Actions” class.

With some of the browser it happens that once mouse hover action is performed, the menu list disappear within the fractions of seconds before Selenium identify the next submenu item and perform click action on it. In that case it is better to use ‘perform()’ action on the main menu to hold the menu list till the time Selenium identify the sub menu item and click on it.

```
public class Flipkart {

 public static void main(String[] args) {

  WebDriver driver = new FirefoxDriver();
  driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
  driver.get("http://www.flipkart.com/");

 //find the webelement where we need to do mouse hover
  WebElement MainMenu=driver.findElement(By.xpath("//li[@data-key='electronics']"));

//***** Mouse simulation using actions class ****
//create object of Actions class
  Actions act = new Actions(driver);
  act.moveToElement(MainMenu).build().perform();

// Right click (context click ) on the page
  act.contextClick(MainMenu).build().perform();

   }

}
```

Like hover we can do lot many things with actions class such as Right click, Double click, Click and Hold, Keyboard activities and so on.

Few of the Method names and their usage

Method Names	Usage
moveToElement(WebElement)	Mouse Hover
contextClick()	Right click on page
contextClick(WebElement)	Right click on specific Element
sendKeys(Keys.TAB)	For Keyboard events
clickAndHold(WebElement)	click on Element and Hold until next operation
release()	release the current control

Few more examples to list would be

```
// Right click (context click ) on the “Electronics” category
act.contextClick(MainMenu).build().perform();

//sending keyboard “Enter” key
act.sendKeys(MainMenu, Keys.ENTER).build().perform();
```

Drag and Drop (using source and destination)

```
public class DragDrop {
public static void main(String[] args) {
WebDriver driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("http://jqueryui.com/demos/droppable/");
// find the element which needs to be dragged
WebElement src = driver.findElement(By.xpath("//*[@id='draggable']"));

//find the element which needs to be dropped
WebElement dest = driver.findElement(By.xpath("//*[@id='droppable']"));
//using Actions clas
Actions act = new Actions(driver);
act.dragAndDrop(src, dest).build().perform();
```

```
}
```

Drag and Drop: - clickAndHold (using coordinates)

```
public class ClickAndHold {
public static void main(String[] args) {
System.setProperty("webdriver.chrome.driver",
"E:\\Tools\\Selenium\\chromedriver.exe");
WebDriver driver = new ChromeDriver();
driver.get("http://jqueryui.com/demos/draggable/");
WebElement obj = driver.findElement(By.xpath("//div[@id='draggable']"));
Actions act = new Actions(driver);
//act.dragAndDropBy(obj, 50, 50).build().perform();
act.clickAndHold(obj).dragAndDropBy(obj, 200, 200).build().perform();
}
}
```

IACSD

The JUnit framework

JUnit is a test framework which uses annotations to identify methods that specify a test.

JUnit is an open source project

Junit Environment Setup:

Prerequisite: Java should be installed

Step 1:

Download the latest version of JUnit jar file from http://www.junit.org. for your respective os and store in some folder.

Step 2:

Create Test Project (java project)

And add those junit jars into **build path** libraries. And then write your test classes.

Writing Junit Test Class

A JUnit *test* is a method contained in a class which is only used for testing. This is called a *Test class*. To define that a certain method is a test method, annotate it with the @Test annotation.

This method executes the code under test. You use an *assert* method, provided by JUnit or another assert framework, to check an expected result versus the actual result. These method calls are typically called *asserts* or *assert statements*.

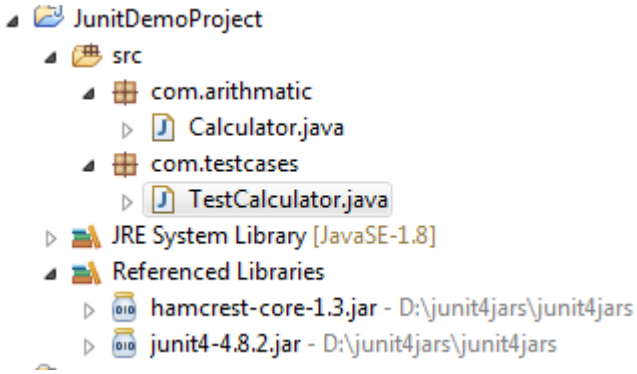
You should provide meaningful messages in assert statements. That makes it easier for the user to identify and fix the problem. This is especially true if someone looks at the problem, who did not write the code under test or the test code.

IACSD

Example JUnit test

See the project structure of sample Ex:

We have to simply create a java project and add junit jars .



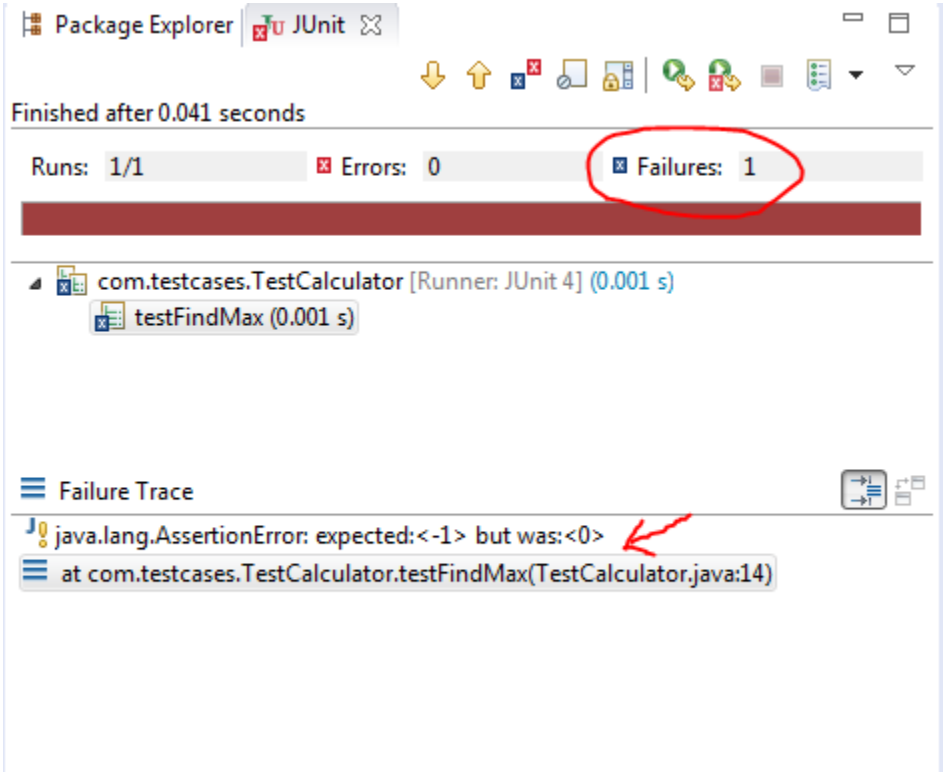
Then we can write the classes as bellow

```
package com.arithmatic;

public class Calculator {

       public static int findMax(int arr[]){
         int max=0;
         for(int i=1;i<arr.length;i++){
             if(max<arr[i])
                 max=arr[i];
         }
         return max;
    }

}
```

```
package com.testcases;
import static org.junit.Assert.*;
import org.junit.Test;
import com.arithmatic.Calculator;
public class TestCalculator {

         @Test
           public void testFindMax(){
               assertEquals(4,Calculator.findMax(new int[]{1,3,4,2}));
               assertEquals(-1,Calculator.findMax(new int[]{-12,-1,-3,-4,-2}));
```

```
        }
}
```

Then run the test case as
right click on TestCalculator class -> Run As -> Junit Test.



As you can see, when we pass the negative values, it throws AssertionError because second time findMax() method returns 0 instead of -1. It means our program logic is incorrect.

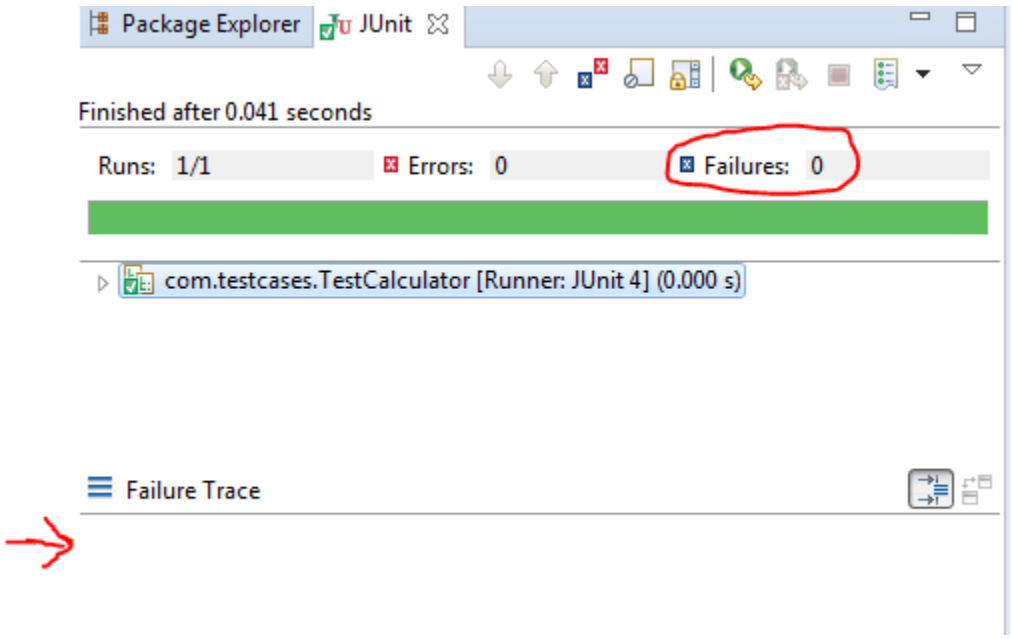
Correct program logic

As you can see, program logic to find the maximum number for the given array is not correct because it doesn't return -1 in case of negative values. The correct program logic is given below:

```
package com.arithmatic;

public class Calculator {
       public static int findMax(int arr[]){
         int max=arr[0];//arr[0] instead of 0;
         for(int i=1;i<arr.length;i++){
             if(max<arr[i])
                 max=arr[i];
         }
```

```
        return max;
    }

}
```

If you run the junit program again, you will see the following output.



Defining test methods

JUnit uses annotations to mark methods as test methods and to configure them. The following table gives an overview of the most important annotations in JUnit for the 4.x and 5.x versions. All these annotations can be used on methods.(next page)

Table of Annotations

JUnit 4	Description
`import org.junit.*`	Import statement for using the following annotations.
`@Test`	Identifies a method as a test method.
`@Before`	Executed before each test. It is used to prepare the test environment (e.g., read input data, initialize the class).
`@After`	Executed after each test. It is used to cleanup the test environment (e.g., delete temporary data, restore defaults). It can also save memory by cleaning up expensive memory structures.
`@BeforeClass`	Executed once, before the start of all tests. It is used to perform time intensive activities, for example, to connect to a database. Methods marked with this annotation need to be defined as `static` to work with JUnit.
`@AfterClass`	Executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database. Methods annotated with this annotation need to be defined as `static` to work with JUnit.
`@Ignore` or `@Ignore("Why disabled")`	Marks that the test should be disabled. This is useful when the underlying code has been changed and the test case has not yet been adapted. Or if the execution time of this test is too long to be included. It is best practice to provide the optional description, why the test is disabled.
`@Test (expected = Exception.class)`	Fails if the method does not throw the named exception.
`@Test(timeout=100)`	Fails if the method takes longer than 100 milliseconds.

Assert statements

JUnit provides static methods to test for certain conditions via the `Assert` class. These *assert statements* typically start with `assert`. They allow you to specify the error message, the expected and the actual result. An *assertion method* compares the actual value returned by a test to the expected value. It throws an `AssertionException` if the comparison fails.

The following table gives an overview of these methods. Parameters in [] brackets are optional and of type String.

Table of Methods to assert test results

Statement	Description
fail([message])	Let the method fail. Might be used to check that a certain part of the code is not reached or to have a failing test before the test code is implemented. The message parameter is optional.
assertTrue([message,] boolean condition)	Checks that the boolean condition is true.
assertFalse([message,] boolean condition)	Checks that the boolean condition is false.
assertEquals([message,] expected, actual)	Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays.
assertEquals([message,] expected, actual, tolerance)	Test that float or double values match. The tolerance is the number of decimals which must be the same.
assertNull([message,] object)	Checks that the object is null.
assertNotNull([message,] object)	Checks that the object is not null.
assertSame([message,] expected, actual)	Checks that both variables refer to the same object.
assertNotSame([message,] expected, actual)	Checks that both variables refer to different objects.


IACSD