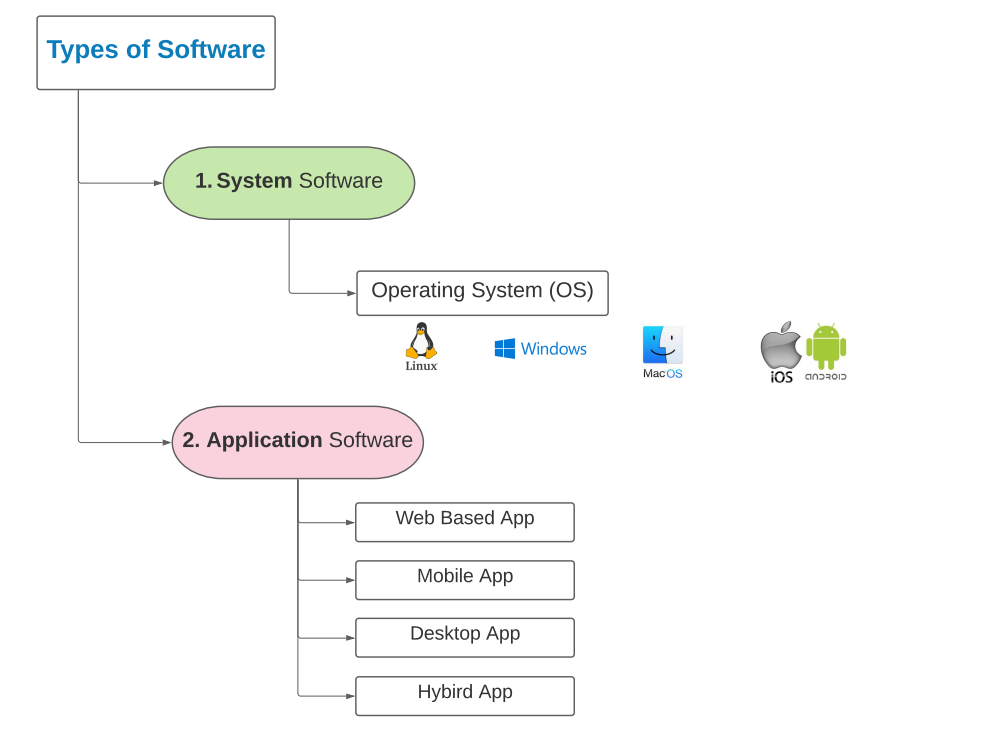
SOFTSKILLS CLASS NOTES

**Softskill Keywords:**

**Software:** Set of instructions that perform specific task for users. Softwares are designed to help people to perform activities.



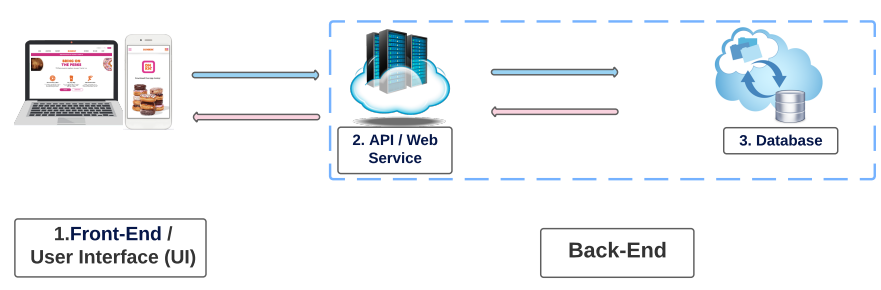
**1. WebBased Application:** Web apps are accessed over network connections using HTTP/HTTPS with any browser (Safari, Chrome, IE, Firefox, etc.)

**2. Mobile Application:** Mobile apps are designed to run on mobile devices such as phones, tablets, or watches.

**3. Desktop Application:** Desktop apps that you download and install on your device / on any Operating system (Mac OS, Windows, Linux )

**4. Cross-Platform/Hybrid Application:** Cross-platform software works on multiple Operating Systems(OS) or devices.

**Software Layers (3) FrontEnd, DataBase, API**

****

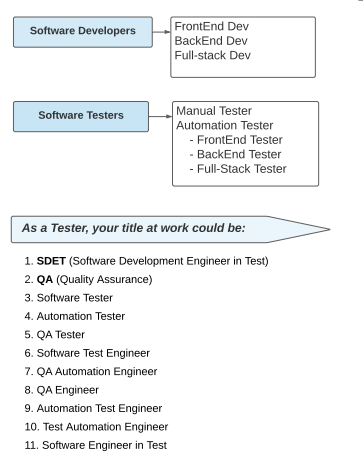
**Functional:** Functionality of a software describes what a software should do, or should not do.

Example: Users login. logout, upload files, edit username, click confirmation button, zoom in/out, etc

**Non-Functional:** Non-functional specify how the software should behave.

Example: Response time: user should download a file min-5seconds - max 35 seconds. Accessibility : Web, mobile, ipad, etc Installation: How to install? available browsers? Performance: How fast does it need to operate? Security: What are the security requirements? a cyber perspective

**Types of Software Testers:**



**Manual Tester:**

As a manual tester, you have a major role to play in software development. ... Your responsibilities are to test cases for testing processes, suggest documents to improve software quality, work with the development team to correct bugs and errors, design test plans and procedures, and communicate with the test manager.

**Automation Tester:**

The work for automated testers is instead spent coding the tests and making improvements to these tests repeatedly as adjustments are needed. Once the test is complete, however, automated testing allows for the recycled use of tests so that they do not have to go through this whole process again.

* **Front End Tester:**

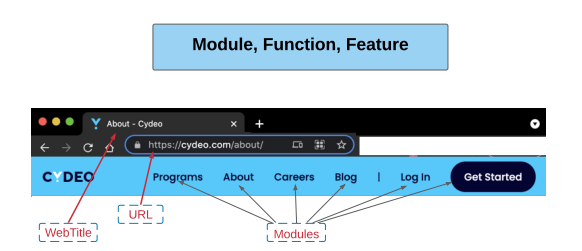
Front end testing is performed to check the functionality and usability of an application’s GUI. It usually involves validating menus, forms, buttons, and other elements of the application that are visible to end-users.

* **Back End Tester:**

Backend testing is commonly performed to check the database. The process takes place to check server-side parameters for a smooth transition. Backend database testing is one of the essential testing activities that happens on all programs.The storage of the data usually happens in the backend. A proper testing process ensures to remove any kind of threat in the database. The backend database allows engineers to showcase based on the requirements.

* **Full - Stack Tester:**

A Full-Stack tester does not know everything but rather someone who understands the mechanisms and can find the best test strategies in an end-to-end test approach to ensure a quality product is delivered.



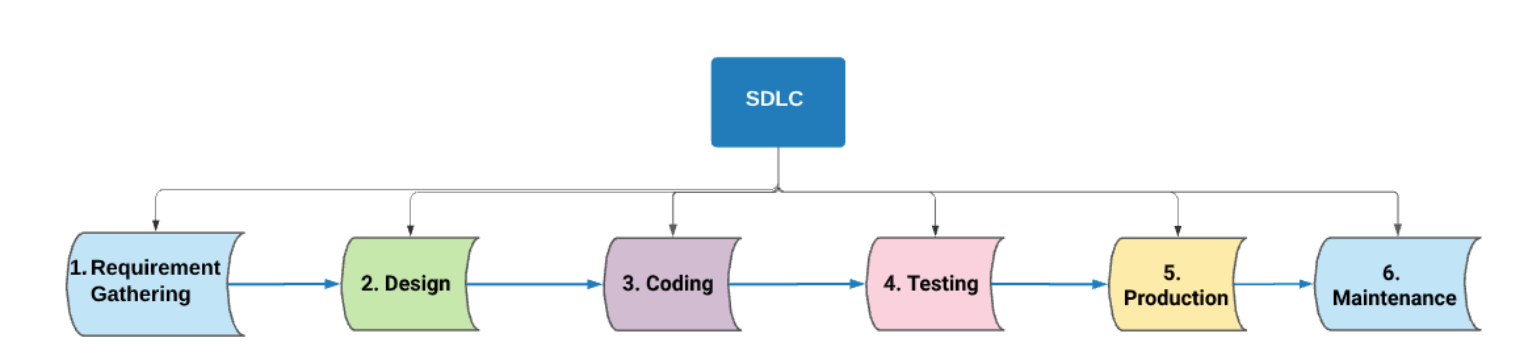
**URL:** URL stands for Uniform Resource Locator. URL is the address of a given unique resource on the Web.

**Web Title**: A website title identifies what the web page is about for both web users and search engines.

**Module**: A unit or component of a software, each module includes one or more features of the app.

**Feature:** Behavior of a software, what users can do with the app?

**Function**:how to achieve a feature, what actions users can perform on the app? (Click, zoom in & out, select, scroll page, enter/type a value, upload/download a file, video & voice call, etc)

****

**S**oftware **D**evelopment **L**ife **C**ycle

**Software Development Life Cycle** (SDLC) is a structured process that enables the production of high-quality, low-cost software, in the shortest possible production time.

1. **Requirement Gathering & Analysis**

\* Business TEAM collects the detail requirements

- Multiple sources with different requirements

\* Analyze the requirements and plan how to develop the app

- (Feasibility Study) team analyzes economic, legal, technology, and schedule

- TEAM asks - are the requirements possible and can we do this?

\* Document all the functional and non-functional requirements - documents are prepared

- (SRS) Software Requirement Specification - very important

- (MR) Market Requirement

- (FR) Functional Requirement

- (NFR) Non-Functional Requirement

\* Who is involved

* + - * Project manager is responsible for the entire project
      * Product owner is the voice of the customer in the development team
      * Business analyst manages the conversations between the product owner and the developer, helps express the conditions of the requirements and manage expectations
      * Client is the customer who pays and makes the requests
      * Stakeholder are investors in the development of the software
      * End users are people who will use the application
      * Subject matter expert (SME) are experts in the field in which application is being developed

\* Requirements should follow the SMART guidelines

- **S**pecific - clearly stated with details, not vague

- **M**easurable - specific, using units of measure, not vague

- **A**ttainable - possible to do for software developers

- **R**ealistic - possible to do for business

- **T**estable - pass/fail, results can be compared to requirements

1. **Designing**

\* Architecture of the application is designed. Create "Design Documents"

- What kind of development framework will be used.

- Micro service development.

- Design the application's UI, database, API etc.

- Design from a technical and "look and feel" perspective.

- Testing framework is also designed in this step.

\* Design Documents

**Document:** SDS (software design specification)

- Outline about the functionality of every module

- Relationship and dependencies between modules

- Database tables -> size and type

- Addresses all types of dependencies

- Listing of error messages

- Complete input and outputs for every module

\* Who is involved

* Business team -> project manager, product owner (communication with technical team), business analyst (translate requirements to be understandable for the technical team)
* Client/Stakeholders
* Architecture/Designers

1. **Coding / developing / implementing**

\* Developers build the software by writing code using the chosen programming language

- Codes are divided into small components (units)

- Developers test their own code

- Developers test their code alongside other developers code

- Developer lead will approve each units code

- Longest phase of the SDLC process

- Front end, back end, and services are developed

- All development happens in development environment

Who is involved

- Developers

- Front End

- Back End

- Full-Stack

1. **Testing**

\* Perform software testings based on functional and non-functional requirements -> planning of the testing process according to the requirements

- Analyze the requirement and plan how to test

- Test scenarios have been created according to scenarios and reviewed

- Exit criteria for testing has been prepared

- Write test case documents and prepare test data

- Identify bugs & write bug reports

- Test reports are documented

- Ensure the product released is a quality product

- All development happens in testing environment

\* Who is involved

- QA engineers (SDET)

- Performance testers

- Security engineers

- Client

- End-users -> BETA Testing

1. **Deployment / Production / Release**

\* Moving the developed software to the production environment so that the users can access the software

* Developers, testers, business team members come together for the release of the application
* Codes and new features are deployed to production
* Becomes available to end-users
* Usually happens on the weekend and/or at night
* Happens when testing has been completed successfully

\* Who is involved

- Everyone who is involved in the development of the app

- Business team -> Support deployment process

- Development team -> Support deployment process

- Dev-Ops team -> Responsible for deployment

- Client -> Sometimes

1. **Maintenance**

\* Keep application running correctly and accessible to end-users

- Fix bugs that were missed during testing and production step

- Project support from Dev-Ops, developers, and business team

- Update and improve the software by adding new features

- Adding new features

- New requirements gathered

- Design features of new function

- Test new features

- Deploy new code so features are accessible by end-users

\*Who is involved

All the members who were involved in building the app, or a company might create a team of 2-3 people to

maintain the app

**Software Environments**

Software environments refers to the collection of hardware and software tools a system developer uses to build software systems, testers test the software, and end users use it.

- Different environments are created independently of one another for different purposes

- Companies typically have three or four environments

- Each environment requires resources and it costs more money to deploy multiple

environments

- The environments are not identical to one another. Code is written in the dev

environment and then "moved" to other environments in stages

There are four different environments for a software:

- **Development environment** is a dedicated environment for developers to write

code for the application/software -> dev.sitename.com

- This includes front end and back end

- Is the most advanced and finished environment

- **QA/Test environment** is a dedicated environment for tester to test the

application/software -> qa.sitename.com

- Some companies have multiple test environments

- **Staging/Pre environment** is a dedicated environment for clients/stakeholders to

test the app and evaluate if they accept the current result -> stg.sitename.com

- It is the same environment as production but with mock data. (Mock data

can protect end users sensitive information)

- **Production environment** is a dedicated environment for end users

-> sitename.com

- Real data

**Software Testing**

What is Software testing

- Expected result is matching actual result -> Compare actual results against SRS

document

- Process to assure the quality of a software

- The goal is to prevent & identify any bugs in the app

- Verify if the actual software meets the client's expected requirements

What is tested in software testing

- Documents

- Code

- Software's functional and non-functional parts

Who performs testing

- Business team members

- Product owner, business analyst

- Designers

- Developers

- Testers

- The client/stakeholders

- End users

How is software testing done

Manual testing

- No programming

- Human error can occur

- Slower

- Not reproducible

- No tool or environment limitation

- Cheaper for short-term

Automatic testing

- Programming required

- Human error can occur but more reliable

- Faster

- Reproducible

- Automation tool limitation

- Cost-effective in the long run

Benefits of software testing

- Fast

- Reusable

- Cost reduction

- Improve security

- Customer satisfaction

- Reliable

- Enhance performance

- Repeatable

- Required in SDLC

- Better quality software

**Software Testing**

**Introduction**

**What is software testing?**

Software testing is a process to try to ensure software quality by finding bugs. Software testing is performed to check if the software meets the client's requirements. Testing measures a software's overall quality in its correctness, completeness, performance, and other functional and non-functional attributes. In short, software testing is done to verify if the software meets the customer's requirements, if it has high quality, if it is bug-free and fit for use.

**What to test in software testing?**

In software testing, not just the app itself being tested. All the outputs of each SDLC phase are tested.

-Documents are tested in the early steps of SDLC to prevent bugs.

-Codes that developers created to build the app.

-The software/system itself from a functional and non-functional perspective.

**Who performs testing?**

Each step of SDLC requires performing testing, and different people are responsible. Testing is/can be done by all technical people associated with the project. People who perform testing are:

1.Business team members & Designers: Test SRS, SDS and many other business docs called static testing.

2.Developers: Developers test their codes by performing Unit & Integration testings to ensure that the individual unit of the software work correctly.

3.Testers: Testers verify if developers developed the app as business required - it is called System testing. System testing is divided into tow parts: functional & non-functional testing. Functional testers test the application's functionality; performance testers check the application's non-functional aspects. Testers also prepare various testing documents like test plan doc, test case doc, bug report doc.

4. The client/stakeholders: To ensure that the final product meets the requests of the client, its call UAT(User Acceptance Testing) - Beta testing.

5. UAT team members: A group of automation engineers perform the application's UAT-Alpha testing to make sure the software can work in the real world.

**How is software testing done?**

- Testing is done manually and automatically.

**Manual Testing**

-Manual testing process to identify bugs by humans. Application must be tested manually before it is automated.

\*Pros:

1.Test all kinds of applications manually

2.Handle difficult functionalities

3.No environment limitations

4.Programming Knowledge is not required

5.Accurate UI feedback

\*Cons:

1.Time-consuming

2.Requires more human resources

3.Not all testing can be done manually - especially

4.Not reproducible

5.Testing process is slow

**Automation Testing**

Automated testing is done through automation tools and is generally more accurate than manual testing. 100% automation is not possible

\*Pros:

1.Cost-effective in the long run

2.Reduces the overall test execution time

3.Can be re-used

4.Automated test scripts remove the chance of human error

5.Helps in working with a large set of data

\*Cons:

1.Automation tools has limitation

2.Skilled automation testing experts to write test scripts

3.Additional effort to write scripts is required upfront

Why is testing required? What is the benefit of software testing? What are the objectives of testing?

1.Improve product quality

2.Save money

3.Improve security

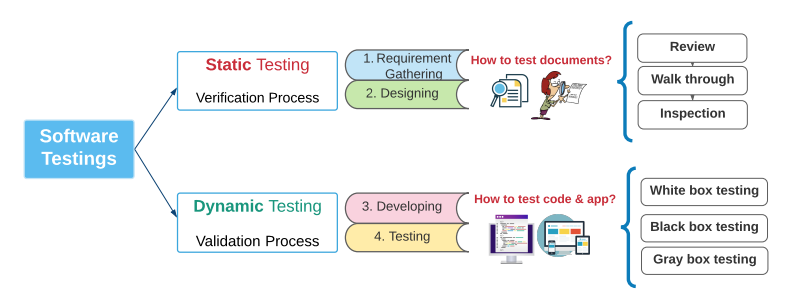
4.Ensure customer satisfaction

5.Effective performance

6.Required in the SDLC

**SDLC & Testing**

Testing in the first phase of SDLC

****

**Static Testing**

* Main goal is to prevent and avoid bugs in the development phase
* Are we building the software as requested and doing it correctly
* Does not require code to be executed and can only be performed using manual testing
* Static testing is also known as the verification process which is a process of creating requirements and design documents and then testing the document's correctness by validating whether they are developing the product correctly
* Cost of finding errors and fixing the documents is less in static testing
* In the information gathering phase testing occurs when the business team & the client test/check their documents manually

-> SRS documents

* In the design phase testing occurs when designers start designing the app in accordance of the SRS documents

-> SDS documents

-> Automation testing framework is designed

**How static testing is performed**

Review

Does not follow any specific process to find bugs in documents

Co-workers can review documents and provide informal comments

Walk-Through

The author will explain the document to their team and participants will ask questions, and take notes

Inspection

Designated moderator will conduct a strict review as a process to finalize the documents

**Dynamic Testing**

* Main goal is to find and fix bugs before software is released
* Are we building the right product
* Dynamic testing requires the code to be executed and involves both manual and automatic testing
* Testing is performed by developers, various testers, and the client by providing input and examining the output to find and fix bugs
* The cost of finding bugs and fixing them by making changes to the software is much higher in dynamic testing
* Dynamic testing is also known as the validation process which is a process of writing code and testing the code/software to verify that the product matches the requirements outlined in the documents and the clients requests
* In the developing phase testing occurs in the form of unit and integration testing by the developers in the dev environment

White box testing

* White box testing requires a good understanding of the applications internal code and behavior.
* A software's code is written by developers so the developers know the internal structure in detail.
* **J-Unit** is tool used

Performed by the developers.

Black box testing

* Black box testing is testing of an application without having any technical knowledge, code’s logic, how the code works, or knowledge of the internal structure, etc.
* Manual testers, the client and end users test the application without technical knowledge. They just give input and get the output/result, they do not understand the function process.

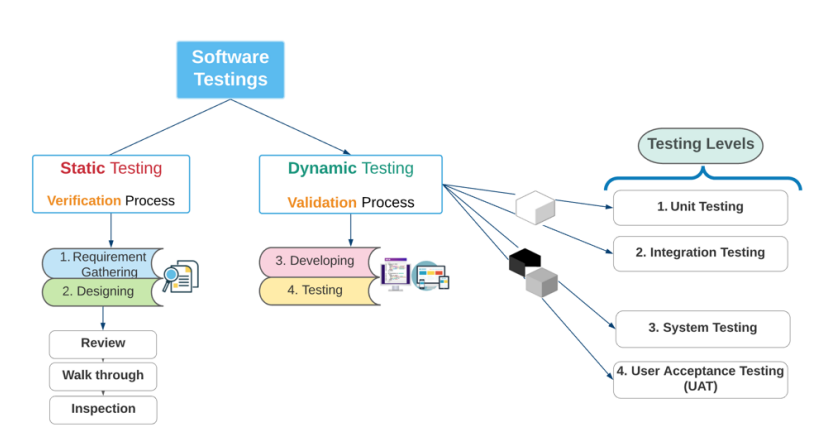
Performed by manual testers, the client and end users.

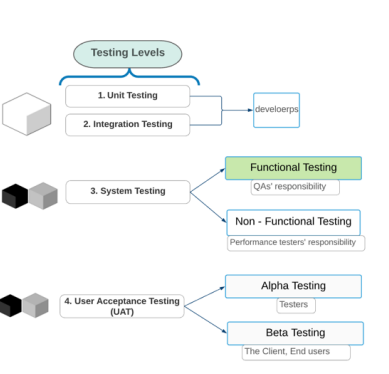
Grey box Testing

* Gray Box testing is to test the software application with partial knowledge of the internal code structure.
* It is a combination of black box and white box testing.
* Automation engineers, performance testers, and security engineers know partial code logic.

Performed by automation engineers/QA/SDET, performance testers, Security engineers

**Software Testing Levels**

****

****

**Unit Testing**

* The first level of testing -> lowest level of testing
* Also called module testing, or component testing
* Performed by developers
* Testing occurs in the development environment
* Unit testing is a part of white box testing because the developers know the internal code knowledge when they perform unit testing
* Test the software's individual unit or module from the code perspective
* Developers write unit tests for their code to make sure that the code works correctly. It allows developers to modify code without affecting the functionality of other units or the product as a whole.
* **J-Unit** is tool used

**Integration Testing**

* The second level of testing -> second lowest level of testing
* Testing the software and validating that the units perform as designed when integrated together -> testing the integration of two or more features together by testing small units of the code together
* Performed by developers after they have integrated the units
* Performed solely by developers in most companies
* Testing occurs in the testing environment
* Integration testing is a part of white box testing because the developers know the internal code knowledge when the perform integration testing
* Test a group of related modules to check data transfer and connectivity between several units/modules

**System Testing**

* The third level of testing
* Comparing the actual result with the expected result
* Performed by the QA testers and performance testers
* Testing occurs in QA or test environment
* System testing is a part of Black box testing when manual testers test the software without knowing the internal code
* System testing is a part of Gray box testing when automation testers partially know the internal code when using testing software
* The software/system is tested in its entirety from the application perspective
* Testers compare the actual software/result with the client's expected requirement
* System testing divides into functional and non-functional testings:

Functional Testing

Testing the operations of the software and validating the results against the document

-> QA/SDET/Automation engineers

QA / SDET validate every function of software as per the functional requirements

Non-Functional Testing

Testing the behavior of the system and validate the results against the documents

-> Performance testers

Performance Testers validate the performance, stress, volume and other parts of the software as per the non-functional requirements

**User Acceptance Testing(UAT)**

* The fourth level of testing
* Performed by client, end-users, testers
* Testing occurs in the staging or pre-production environment
* UAT is a part of Black/Gray box testing depending on the testers depth of knowledge of the internal code
* UAT aims to evaluate whether the software is acceptable for release. For UAT, the product owner(PO) provides special requirements based on real-world scenarios

UAT IS divided into Alpha and Beta testing:

* Alpha testing is when UAT is carried out by any organization's testers
* Beta testing is when UAT is carried out by the client or end-users

**Testing Layers**

Applications have three layers -> (UI/FrontEnd)/DataBase/API

* The layer we can see and interact with is UI
* The information that is displayed in the UI is stored in the database layer
* The communication that occurs between the UI and the database is the API layer
* Each layer needs to be tested

**Front End testing**

* Testing the application's web elements manually and automatically
* Use programming languages, and testing tools like Selenium, JUnit, Cucumber
* Front end testing deals with front end/UI
* Can be tested with manual or automated testing

**DataBase Testing**

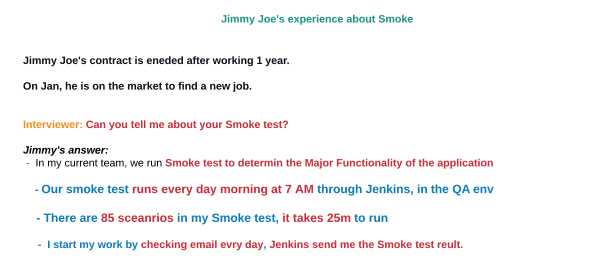
* Testing the software's data and information
* Testing if a functionalities data that are created
* Manual database testing is done by writing quarries with SQL and IDE
* Automation is done with programming language like java, and testing tools like JDBC
* Data is stored in the database and testing involves testing to ensure that the data is correctly stored in the database tables

**API Testing**

* Communication that happens between components happen through API
* In order to perform a function of a software API calls are made -> API requests
* API testing is one of the best type to verify a software's functionalities
* Testing if a functionalities' data that are created from UI is received and stored into the database or not
* Manual API testing is done by using an IDE or PostMan
* Automation is done with programming languages like java, and testing tools like RestAssured API libraries using java

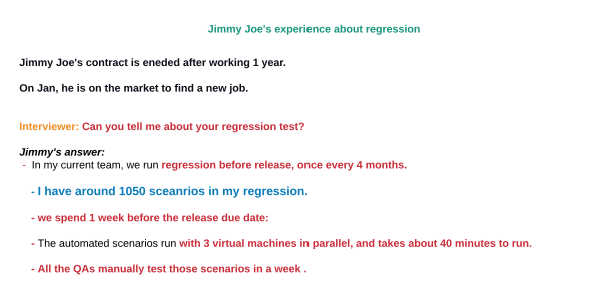
**Smoke Testing**

* Smoke tests occur during the development phase
* Smoke testing is performed everyday to determine whether the application is stable or not -> Checkpoint
* Typically scheduled to run in the early morning -> 6am
* Smoke test also checks to see if the QA environment is up and running
* Goal is to ensure that the major features of the software is working; new code is merged with existing code in testing environment after smoke tests
* It is like a gate, code is tested before being pushed to the testing environment. If the new code passes the smoke tests only then is the code pushed to the test environment for further testing
* Consists of a minimal set of tests to run -> 50-90 scenarios -> More scenarios take longer
* The set of tests are selected in a meeting between all the testers
* More scenarios are added to run as apart of the smoke test when there is a major function developed and added to the software
* Smoke test is meant to be quick to execute -> Faster = Better -> 15-30 min
* Smoke tests are done automatically using services like Jenkins, GitHub, and virtual machines
* If there is any bug/error found from the smoke test the QA team then needs to test the code manually to confirm the bug and identify which major function has the bug
* Smoke tests results are automatically generated and sent to the development teams and testers email accounts



**Regression Testing**

* Regression testing confirms that software previously developed features are still working well with the new added feature.
* The main goal is to make sure that a product functions as expected after any changes made to the application.
* Regression tests run using services like Jenkins
* Regression testing is performed both manually and automatically. All the scenarios that have test scripts (code) are automated. Some functionalities are not automated so they are manually tested.
* Ideally, regression testing is performed before software to the production environment/Release. However nowadays it depends. There is no fixed answer. Regression testing might be performed once a week, everyday, or once every 3 months. (We will say we do it once every 3 months)
* There are 500 to 3000 scenarios in regression test (We have around 700 scenarios)
* For 700 scenarios it would take about 3 hours to run the automated scenarios and 1 week (5 days) to test the manual scenarios.
* Automated scenarios could run in 2 or more virtual machines to make the process faster.



**Software Methodology**

* Software Testing Methods -> How to test software (Agile, Waterfall)
* Software Testing Levels -> When to test software(Unit, Integration, System, UAT)
* Software Testing Types -> What type of testing is used to test software (Performance, Smoke, Regression...)

An essential set of principles and rules for managing a project.There are several methods in the IT industry. All the project management methods have unique rules and principles that describe how to follow the SDLC.These rules are strict and define how to manage, plan, and control software development procedures

These methods include

* Waterfall -> sequential
* Agile -> non-sequential
* V-Model -> sequential

**Waterfall Method**

* In the construction industry a product has to be developed step by step. People should work carefully on each step because they cannot go back to the previous step to fix errors based on the waterfall method rule
* First software method used to develop software
* Each phase is completed before starting the next step
* Testing is not done concurrently with development
* Simple and easy to understand and use
* Each phase is divided into separate steps
* Each step depends on the outcome of the previous stages to proceed
* Waterfall guarantees success when the projects are small with clear requirements

Advantages

* Easy to use and implement, no need to train the team and familiarize them with the waterfall methodology
* The start and end points for each phases are set, easy to measure progress
* The stability of the model makes project management easier
* Since all phases occur without any overlap, reducing the project complexity
* Careful planning of project development structure reduces the number of issues
* Quality and detailed documentation

Disadvantages

* Poor software result for long duration and large projects
* If the customers requirements are not clear then it could create significant complications later in the project
* Little to no changes. Once the requirements are finalized any changes in requirements would not only be difficult but also costly

**Agile Method**

* Modern, well known, and widely used in software projects
* Started in 2001
* Under agile there are different implementations/frameworks
* Agile is a mindset/approach to software development and can be implemented differently with different frameworks
* Goal is to speed up development times in order to bring new software to market faster
* Shortening deployment time
* Getting feedback from users quickly -> client is involved in each step of SDLC
* Success of agile depends on (communication)
* Short and quick
* Self organized team
* Communication
* Quickly adjust the working process
* Quick and constant feedback
* To implement the agile in a company each member should know Agile-Manifesto -> rules about agile
* Is an advisory, public advice which contains the values and principles to follow in the agile methodology

**4 Agile Values**

1.Individuals and Interactions

\*Trust individuals and prefer interactions over processes and tools

2.Working Software

\*Focus on delivering working software rather than complete documentation

3.Customer Collaboration

\*More customer involvement not just negotiation

4.Responding to change

\*Willing to accommodate changes instead of being rigid to a plan

**12 Agile Principles**

1.Satisfy Customer

\*Short development cycles, quick delivery, and happy customer

2.Accept Changes

\*revise goals

3.Continuous Delivery

\*Deliver working software frequently

4.Collaboration

\*Working closely with everyone involved, especially the client, through development cycle

5.Trust and Support

\*Show confidence in the people involved

6.Face-to-Face conversation

\*Allow transparent interactions

7.Working Software

\*This defines the progress of project

8.Maintain Constant Pace

\*Agile processes to normalize the development speed

9.Good Ideas and Improvement

\*Continued attention to technical details

10.Simplicity

11.Self-Organizing

\*The key to good architectures, requirement, and designs -> organize

12.Revisit

\*How to become more efficient

**Advantages(Agile)**

* Better time-to-time market -> produce working software and making it available to end users more quickly
* Realistic approach to software development
* Strong collaboration
* Accept changes at any time
* Project goal is defined with the client
* Project quality improved

**Disadvantages**

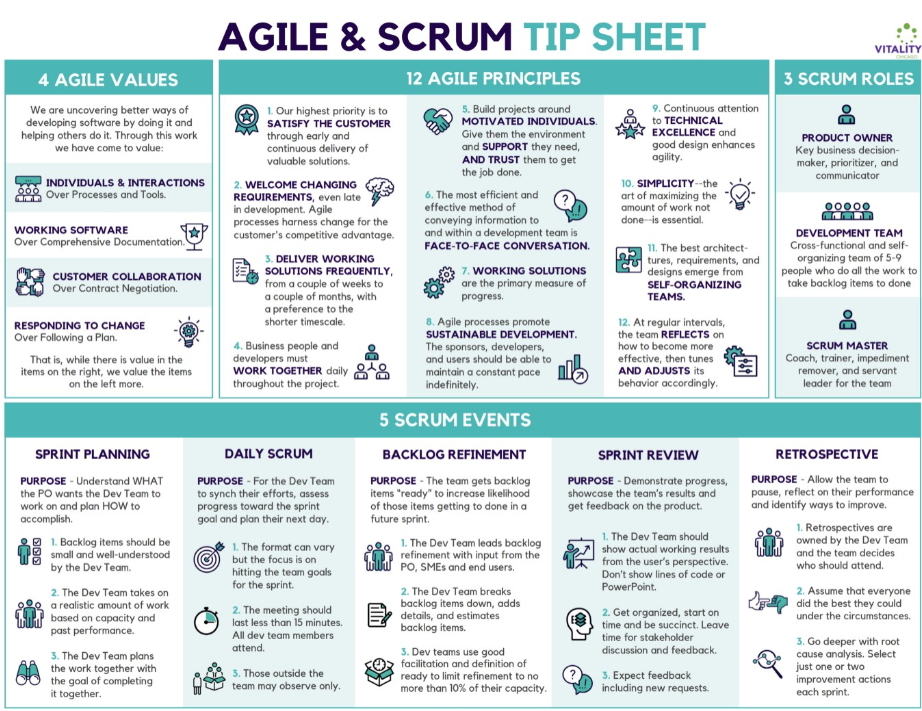
* Little to no planning
* Lack of documentation
* Training and skill required
* Organizational Transformation -> business and development members

Agile Principles in more detail can be found here

https://agilemanifesto.org/principles.html?msclkid=f11e8f4ab6b011ecb8c32bd1bd1c3885

Agile Values in more detail can be found here

https://agilemanifesto.org/?msclkid=f11e6449b6b011ecbf2f0be440d16d07



**SCRUM**

* Scrum is one of the Agile types/ frameworks
* Scrum is mainly used the most in software development projects compared to other Agile types
* Scrum encourages the team to work together, learn through experiences, self-organize while working on a problem and reflect on their wins and loses to improve continuously.
* No team leader, Problems and task sharing decisions are decided by the team as a whole.
* Scrum team creates and adapts their ways of working
* **Scrum 3 Roles:** Product owner, Development Team and Scrum Master
* **Scrum events (Meetings):** Grooming, Sprint Planning, Daily Stand Up, Demo and Retro
* **Scrum Artifacts:** Product Backlog, Sprint Backlog, Product Increment

**Sprint**

* Sprint is a fixed time box, also called iteration
* It is between 1 to 4 weeks (2 weeks sprint cycle is most common)
* Team members decide sprint length together
* An active sprint cannot be extended
* A sprint is used to accomplish **sprint goals:**
  + Conduct scrum meetings - grooming, sprint planning, stand up, demo and retro
  + Specific requirements must be completed (developed and tested)
  + Demonstrate the completed work to the client
* Each new sprint starts immediately after conclusion of the previous one
* Sprint cycle is decided by the scrum team (PO, SM, and Dev team)

**Product Owner (PO)**

* Only one PO per product, even with multiple teams
* Represents the voice of the customer
* Determines the release date and contents
* Defines the requirements and prioritizes them
* Takes an active role in Sprint Planning and Sprint Demo meetings
* Writes User stories and Acceptance criteria and manages them in Product Backlog **(Using Jira tool)**

**Scrum Master (SM)**

* SM is a facilitator (Someone who makes progress easier, who helps a team understand their common activities)
* Helps team members to follow Scrum practices in a productive way
* Removes obstacles / impediments / blocks / problems
* Shields the team from external interference / disturbances
* SM is the master of all the scrum meetings
* Ensures the team has a good relationship between each other

**Development Team (Dev-team)**

* Dev team ideally consists of 5 to 9 people
* Dev team completes tasks and assignments to develop and test the project
* They should be self-organizing and self- managing team
* Typically an agile team comprises of 3 to 4 developers, 1 tester, 1 technical lead, 1 product owner and 1 scrum master
* Cross responsibility - Delivering value

**User Story**

* User story is a short description of a feature told the perspective of the user or customer
* It is the smallest unit of work in Scrum
* A user story’s purpose is to show how a unit of work will deliver specific value back to the customer
* Typical user story template:

**As a < type os user >, I want < some goal> so that < some reason>**

As a restaurant manager, I want to add a new menu so that I can update the menu with new foods.

**Daily Stand Up Meetings**

* Also called **Daily Scrum** or **Stand Up**
* It takes roughly **15 minutes** (Done daily, typically in the morning)
* Designed to quickly inform team members about what’s going on across the team
  + **What did you do yesterday?**
  + **What will you do today?**
  + **Does anything block you? Do you have any impediments?**
* It is not a detailed status meeting. It is for status update, not for any discussion
* If discussion is required on a matter, another meeting should be scheduled
* Participants usually stand instead of sitting so that the meeting gets over quickly
* SM displays the **project management tool (Jira)**
  + SM displays the **Sprint Active Board** (What is being finished and what’s taking longer than expected)
  + SM displays the **Burndown Chart** (a graphical representation of work left to do versus time)
* **Attendees:** Scrum Master (Host the meeting), Development Team (Answers the 3 questions), PO and BA are optional

**Impediments in Scrum**

* Anything that is slowing down the team (Issues or risks that are likely to occur soon)

**Examples:**

* + Bugs (for a user story)
  + Team member is not skilled or knowledgeable enough
  + Technical issues - access issue, testing tool is not recognized, github issue etc
  + Lack of knowledge to the project
  + Less than perfect skill (in one area)
  + Operational issues
  + Proces issues - might have dependency on some other tasks

**Grooming Meeting**

**Backlog Grooming / Refinement**

This is not a scrum meeting according to the scrum organization. However, many companies conduct Grooming meeting in a regular schedule time

**Attendees:**

* Scrum Master - who facilitates the meeting
* Product Owner - who clarifies the details of the product backlog items / user stories and their respective acceptance criteria
* Development Team - who works on the user stories

**Grooming meeting process:**

* PO defines / explains the user stories that are listed in the product backlog to the dev team
* Discuss each item in detail, get enough clarity from the test/development view.
* User stories effort estimation/ giving points (optional)

**Time Box:** 1 - 2 hours for 2 week sprint or 2 - 4 hours for 3 to 4 week sprint

**When:** Before a new sprint or at the sprint starting day

**Story Point /Estimation**

* People are naturally better at relative estimation that absolute estimation
* In Scrum, teams use relative estimation method to size each user story
* Relative estimation takes less time and is easy to refine
* Easy to explain to others and justify expectations
* Estimate the user stories on Product Backlog
* Compare one story to another
* Allows you to select a predictable volume of work to be done in a sprint

Things considered for User Story estimation

* **Complexity :** Consider the complexity of the story
* **Risk:** Consider the team’s inexperienced with developing/testing this story
* **Implementation:** How much code will need to be written for this story? Have we written similar code before?
* **Interdependencies:** Consider other issues - Experience, Testing tool availability, skillset, application knowledge, environment availability, etc.

Steps to Estimate Story Points

* PO explains a user story in detail with its Acceptance Criteria
* Dev team members ask questions about the story
* Developers and QAs starts to give point
  + This can be done with Agile-Card
  + Use online Scrumpoker app https:[www.scrumpoker-online.org/en](http://www.scrumpoker-online.org/en)

**Sprint Planning Meeting**

* Establish goals for your sprint
* Scrum team identify the exact user stories they will finish in a sprint
* The entire team agrees to complete a set of user stories within a Sprint
* Each user story that is selected into the Sprint backlog has to meet the teams Definition of Ready (DOR) criteria
* SM created a Sprint Backlog
* Remind the team of the big picture or goal for the sprint
* Discuss any new information that may impact the plan
* Confirm team capacity
* Review the Definition of Ready (DOR) and make any appropriate updates based on technology, skill or team member changes since the last sprint
* Present proposed product backlog items to consider for the sprint backlog
* Scrum Master calls for a group consensus / group agreement on the plan
* Team and Product Owner agree upon the best plan they can make given what they know right now

Meeting Process:

* SM creates a new sprint backlog in project management app(e.g Jira)
* Takes/ pulls user stories from the Product backlog to the current Sprint backlog
* A team can finish / complete user based stories on their team capacity and team velocity

Attendees:

* **Scrum Master** - who facilitates the meeting
* **Product Owner** - who identifies the sprint goal with the dev team
* **Development Team** - who works on the user stories

Time Box: 1 hours for 2 weeks sprint or 2 hours for 4 weeks sprint

When: First day of each spring

**Team Capacity**

Capacity is based on the team's expected or projected future availability. Capacity is how much availability the team has for the sprint. This may vary based on team members being on vacation, ill, etc. The team should consider capacity in determining how many product backlog items to plan for a sprint. The team may want to consider taking on fewer product backlog items if capacity is expected to be less for the sprint. Likewise, if more team members are recently added, the team may want to take on more product backlog items.

**Team Velocity**

Velocity is based on actual points completed, which is typically an average of all previous sprints. Velocity is used to plan how many product backlog items the team should bring into the next sprint. Velocity should only be used for the team for planning purposes. The success of the team should always be based upon the delivery of value--i.e. a working increment of the product delivered to the customer.

**Sprint Backlog**

* Sprint Backlog is one of the Scrum artifacts
* Each sprint has one Sprint backlog
* Scrum team selects user stories from the Product backlog into the Sprint
* SM creates sprint backlog in sprint planning meeting
* Sprint backlog is used and maintained by Dev team and SM during the sprint
* Sprint backlog has: Sprint number, Sprint starting & ending date, List of tasks needs to be done in the current sprint’
* Total story points

**Demo Meeting**

* It’s also called Sprint Review meeting
* The meeting is facilitated by the Product Owner
* The entire Scrum team and stakeholder join the meeting to discuss what was done and what is not
* Team members demonstrate the product increment
* The Product Owner reviews and accepts the User Stories according to the Definition of Done (DOD)
* SM ends the sprint after the Demo meeting
* If any User Story is not completed, it will move back to the product backlog, and the team will work on it in the next sprint
* **1 to 2 hours** and the end of the spring (**Last day of sprint**)

**Attendees**:

* Scrum Master - who facilitates the meeting
* Development Team - who demonstrate their “done” tasks or Product increment to PO and the Client
* The client and Stakeholders - who owns the project
* The PO - accepts or rejects the User Stories based on the DOD
* Project Manager - who is responsible for the project delivery

**DoD = Definition of Done**

Even before starting sprint 1, the dev team all together decide DOD list together. The DoD may vary from one Scrum Team to another, but must be consistent within one team. The DoD is usually a short document in the form of a checklist, that defines when a product backlog item (i.e. user story) is considered “done”. It has various rationales and various ways to explain it:

* You need a **common definition of what “done”** (= “this user story is finished”) **means**. Otherwise it will mean something else for every person on the team.
* All your **non-functional requirements** reside in the DoD.
* A **general list of acceptance criteria** to be added to every story’s specific acceptance criteria.
* Many **improvements** you find in your retrospectives end up in the DoD.

Most teams start with no or a very simple DoD. They then add to the DoD after each sprint as needed. *Tip: Don’t paralyze yourself with an excessive DoD!* But keep in mind: “done” in an agile project means **“no more work needs to be done before shipping”**.So if someone says “the feature is done, but it only needs to be integrated, tested, deployed, …” it would NOT be considered “done” in an agile sense!

The best **check whether something is “done” is to simply ship it**! If you can ship it, it’s really done; if you cannot ship it, simply do the work missing before you can ship it to make it “done”. *Mind you: you don’t need to actually ship it, but you need to make believable that you could.*

A typical DoD might look like this example:

* Automated tests are written and all tests are green
* Code is refactored and reviewed
* Code is integrated with master branch
* Deployed to staging environment
* Translated into English and German

A concise definition of done will help you deliver quality, keep your slate clean and react flexibly to changing requirements.

DoR = Definition of Ready

The DoR is the little cousin of the DoD. It is a checklist of what needs to be done to a product backlog item **before the team can start implementing it** in the next sprint. You can view the definition of ready as the “DoD” the Product Owner has to fulfill so that the Development Team accepts the story in the Sprint Planning meeting.

Note that the *DoR is NOT part of the Scrum Guide* — and that is for good reason. The DoR should not be used as a phase gate for Sprint Planning or as a way to push away responsibility! It should rather be a guideline for the team of what needs to be done during backlog refinement.

Most teams start out with an empty DoR and add to it as needed. *Again: don’t paralyze yourself by coming up with lots of bullet points here!* It’s better to start simple and then add to the DoR as needed.

A typical DoR might look like this example:

* PO and Dev Team need to have talked about the story at least once
* Story must have clear business value
* Effort needs to be estimated
* Story must be broken down enough to fit a single sprint
* Story needs at least one acceptance criterium

In all honesty, you will likely not need to write this down. When you talk about new user stories in a backlog refinement session, you will intuitively drive stories towards being “ready for sprint”.

In case you want a good guideline for your DoR, consider the INVEST schema: A user story should be **I**ndependent, **N**egotiable, **V**aluable, **E**stimable, **S**mall and **T**estable.

* **Independent**. A user story should be independent from other stories. If you really write “user stories” as opposed to traditional work items or tasks, you will end up with drastically fewer dependencies automatically. Occasionally you might still have dependencies — in that case simply note the dependencies.
* **Negotiable**. A user story should describe *what* the customer needs, not *how* the developer should implement it. The development team should always be able to propose alternative solutions/implementations to deliver the business value for the customer.
* **Valuable**. The business value must be stated. This is often the “…so that…” part of the user story format: “As a *persona* I want *feature* so that I get *business value*”.
* **Estimable**. The development team has to be able to roughly estimate the effort of the user story. This often means that the development asked the product owner a few clarifying questions and came up with a rough idea of how it could be implemented.
* **Small**. It has to be small enough to be done within a sprint. If it is estimated to be bigger than a sprint, keep splitting the user story until you have small stories.
* **Testable**. You need to be able to test, whether the user story is done and fulfills its purpose. This usually means you need a set of clear acceptance criteria that you turn into test cases.

Again, don’t over-theorize the DoR. Stick with INVEST or agree on a simple format the development team needs before they can sensibly start work. Both the product owner AND the development team are responsible for getting a story ready in the sense of your DoR.

**Retro Meeting**

Retro meetings are used to create improvement opportunities. It provides team members the chance to reflect on the sprint.

**Meeting Process:**

* What went well?
* What went wrong?
* What should we do to improve our team’s efficiency?

**Attendees:**

* **Scrum Master** - Who Host the meeting
* **Development Team -** Who revisit the sprint and make summary of what went well and wrong

**Time Box:** 1 to 2 hours for 2 weeks sprint

**Whan:** At the end of the Sprint, after the demo meeting