# **Software Process**

Activities to be done in specific order to **develop a software**

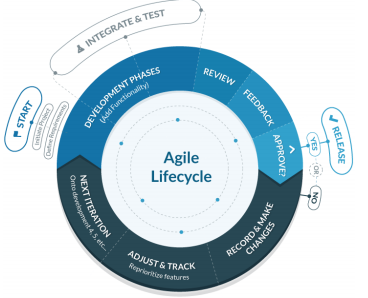
* **Specification requirements**
* **Design and implementation**
* **Validation**
* **Evolution**

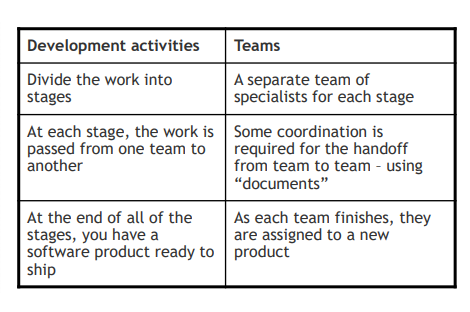
RIGID

* Condition in which incomplete and/or changing requirement is Expensive

# **Software Process Model**

|  |  |  |
| --- | --- | --- |
| **Waterfall Model** | **Spiral Model** | **Agile Model** |
| 1. Requirement analysis and definition 2. System and software design 3. Implementation and unit testing 4. Integration and system testing | * Incremental development based on risk and error | Iterative incremental process for **rapid software development** |
| Intensive planning and documentation nature  Milestones are planned ahead and progress is defined relative to the planned schedule |  | User satisfactory focused  Adaptive to changing requirement  Constant integration and deployment  Independent team work |
| Difficult to implement changes and errors are expensive |  | Difficult to facilitate throughout the phases. |



****

# **Artifacts**

* Items that can represent **work done**
  + Code
  + Requirements specifications

Preserve, communicate, maintain and protect **artifacts**

# Version Control system

Method for preserving versions of artifacts so that we can:

* Revert
* Compare changes
* Who made changes

|  |  |  |
| --- | --- | --- |
| Local VCS | Centralized VCS | Distributed VCS |
| Keep versions in **local computer** | **Single server** containing all the version files and number of client check out files from it | **Developers fully mirror the repository** including the full history. They themselves will be a repository source. |
| **No server settings** necessary | * Easy to collaborate * Easy to maintain admin works | * Easy collaboration * Admin controls |
| Difficult to distribute and Collaborate | Single point of Failure | Strict Server Management is required |

# **GIT**

* Web based **Central Version control**
* Stream of snapshots capturing **file system and changes**
  + Git **doesn’t store unchanged files** but points them to the previous files (pointer)
* Characteristics of GIT
  + Most of the operation is **local**
  + **Built in security** with Hash to store file contents
  + GIT only **adds data**

### **Commit -** Safely stored in **local database (repository)**

**Modified** - The file that has changed since the last commit

### Modified > Staged > Committed



## Working directory (TREE)

* Single checkout of one version of the project
* These files are pulled out off the compressed database and placed on disk for you to modifiy

## Staging Area (index)

* A file stores information about what will go into next commit

## Git directory (repository)

* Metadata and object database
* What is copied when you call **clone**

## Git repository

* Special directory containing project files
* .git stores the history of changes

## Metadata

* Each version should have
  + Unique name
  + Date
  + Author

## .git will keep a record of

* All of the information from past commits
* Repositories (local and remote)
* New files
* Adding and removing files to/from staging area
* Commit changes
* View commit history
* Undo changes

## Git Branching

* **Diverging** from the **main branch of development** and continue to do work **without messing with the main branch**

## Git add

* Will create a **blob** file containing the changes made to the git repo
* Git either creates new blob or points to old blob
  + Git will add only if the item has changed. If not it will point to the last committed blob item.
* It is in the staging area but not stored permanently, it will be deleted by the garbage collector if not committed.

## Git commit

* Create a **commit object** containing metadata and pointing to root project tree

There can be one of two types of files in git repositiory

* Tracked
  + Git knowing about the file
* Untracked
  + The file has yet been added as git object
* When you stage changes, you have **Clean directory**

## Remote repository

* Allows easy distribution, collaboration and communication between team members

## When git clone is called

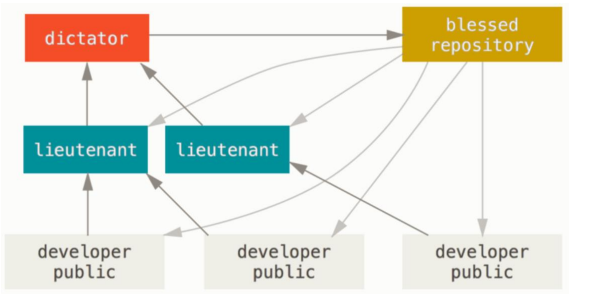
1. Names the repository origin
2. Pulls/downloads all it’s data
3. Creates a pointer to where it’s master branch is and calls it **origin/master** locally
4. Set you own **local master branch** be the same place where **origin master** branch is

## Integration manager workflow (PULL REQUEST)

1. **Project maintainer** pushes to their **public repo**
2. Contributor **clones** that repo and **make changes**
3. Contributor **sends** the **maintainer an email** asking them to **pull changes**
4. The maintainer **adds he contributor’s repository** as a **remote and merges locally**
5. The maintainer pushes **merged** **change** to the **main repository**

## Adv of Integration manager

* Development team can continually work on other features while maintainer assesses the work done
* Contributor does not need to wait for merge before starting another work



### Commit guidelines

* No White space error
* Commit logically separate changeset
* Use quality commit mssages

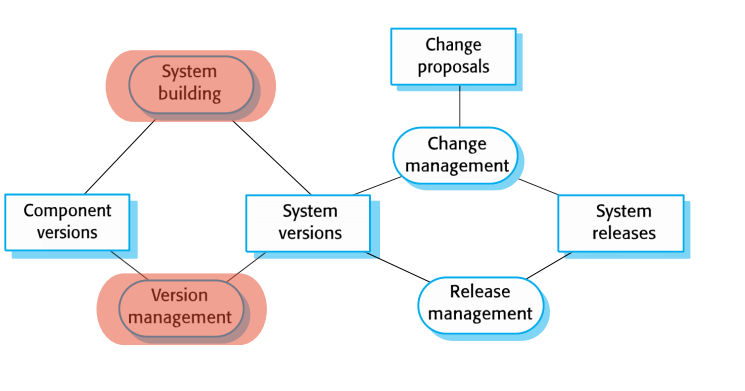
## Git organization

* Owner-member relationship where The owner has the admin authorization and can maintain accessibility to everything

# **Configuration management**

Management of

* Policies, processes and tools for managing changing software system
* Tracking what component versions incorporated into each system version
* Process of managing code-lines and base-lines



## System Building

* Assembling program component, data and libraries, then compiling these to create an executable system

## Version Management

* Keeping track of artifact, component and system versioning
* Ensure changes made to components by different developers do not interfere with eachother

## Change Management

* Keeping track of requests for changes to software from client
* Estimate the cost and impact of changed requirement and decide whether changes are plausible or worth it

## Release management

* Preparing software for external release and keeping track of system versions

## Base Line

* Ecosystem of the application and internal file versions

## Code Line

* Versions of **source code**

# Semantics of versioning

Major.Minor.Patch

### Major

* + Versions when you make **backward-incompatible** changes

### Minor

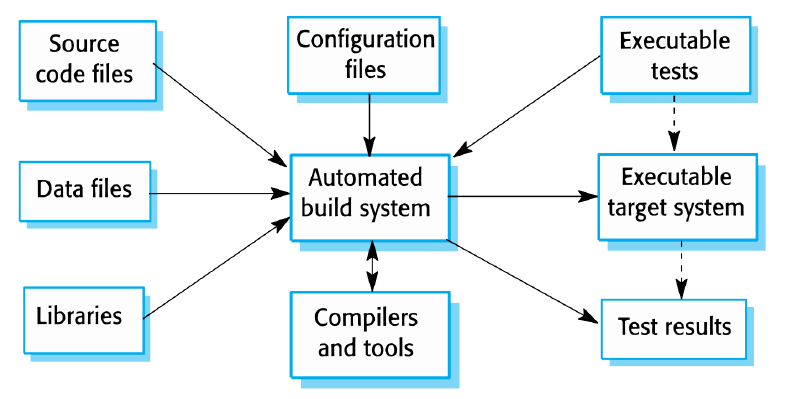
* + Versions when you **add functionalities** in a backward compatible manner

### Patch

* + Versions when you make a backwards compatible bug fix

## 

Firetruck uses Ladder 3.1.0

Ladder 3.1.9 is safe to use, Ladder 3.7.7 is safe to use. Ladder 4.1.1 is NOT because it is not backward compatible.

## **System building**

Process of creating **a complete executable system** by c**ompiling and linking**

# **GRADLE**

## TASKS

* Gradle uses objects called **tasks** to get instructions on how to make a **.jar** file
* Single atomic piece of work for a build
  + Compile a class

## Project

* Composition of several tasks
  + Creation of Jar file

## Build lifecycle

### Initialization

* + Projects are to participate in the build

### Configuration

* + **Task** **objects** are **assembled** into an internal object model called **Directed Acyclic Graph**

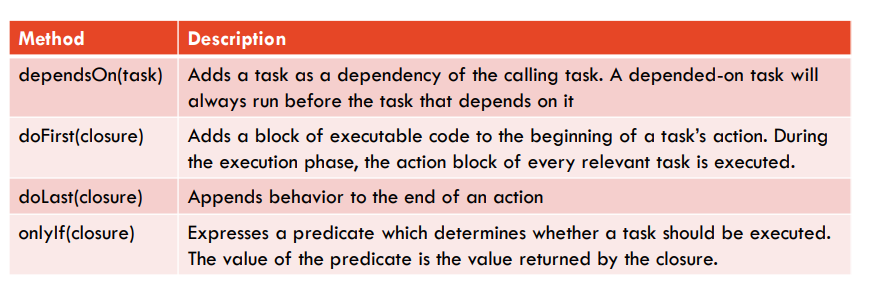
### Execution

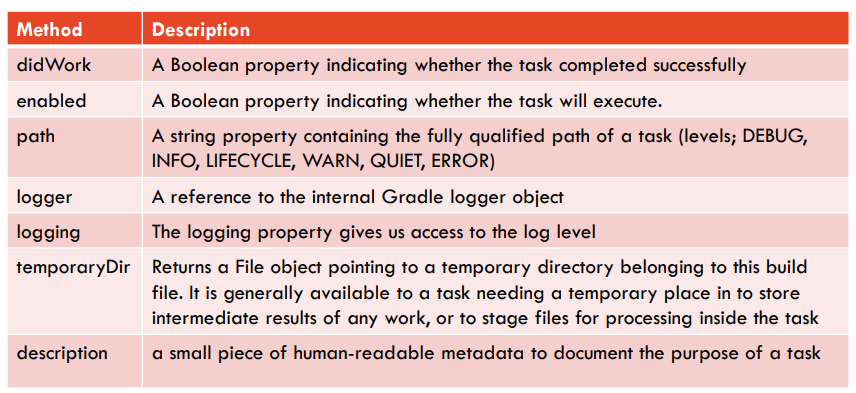
* + Build tasks are executed in the order required by their dependency relationship

## Configuration block

* To **setup variables** and **data structures** needed by the **task action** when it **runs in the build**
  + If the task requires information more than just build tasks, it can be called to configure the variables

## Closure

* Information about configuration and build actions
* Building code.

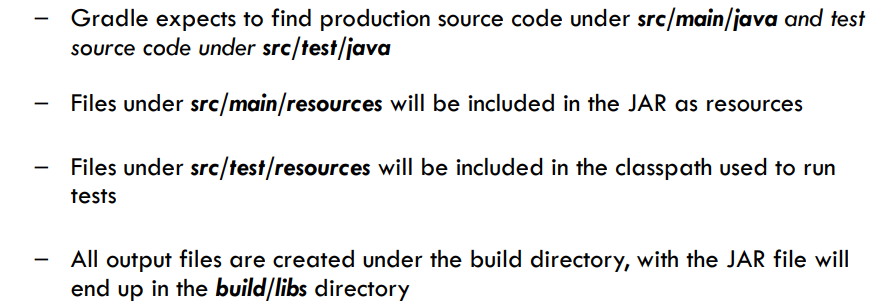


## Plug ins

* Allow adding new task definition
* Change the behavior of existing tasks
* Add new objects
* Create keywords to describe tasks that depart from the standard Gradle categories

## Java plug ins in Gradle

* Specific configurations that helps with Building projects.
* Default tasks area already defined for unit testing and creation of JAR file.



## Gradle clean

* Deletes the build directory, removing all built files

## Gradle assemble

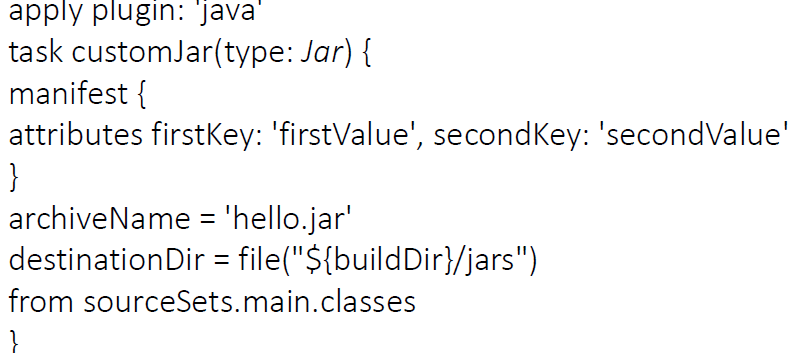
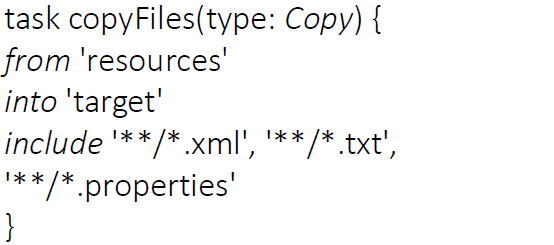
* Compiles and jar your code, but deos not run the tests

## Check

* Compiles and tests your code.

## When Gradle build is called

* Pre-built automatic tasks and projects to build java application. Specified by Java plug in/dependencies
* Class files generated and placed in a directory
* Test report files
* JAR Built using the Project Directory

**Copy Task Build Jar File**

# **Software testing**

## Software Quality

* + The satisfaction of client’s needs
  + Running in a correct behavior
  + Easy to use
  + No errors

## Software Quality Assurance

* Ensuring software under development meets a high quality and holds high integrity

## Software Testing

* Demonstrate that Software meets requirements
* Find incorrect or undesired behavior
  + System crash
  + Incorrect computation
  + Unnecessary interactions and data corruptions

## Who does tests?

* Development team
  + They should have their own testing procedures to ensure that the code functions properly
* Independent testers
  + To allow fresh perspectives

## Software Testing Process

* Select and prepare **suitable test cases**
* Selection of suitable **test technique**
* **Test execution**
* **Trade off analysis**

## Software Testing Levels

* Unit/ functional testing
  + Verify functionality of **Software Component** independently
* Integration Testing
  + Verify **interactions/communication** between software components
* System testing
  + Verify functionality and behavior of the **entire software system**
* Acceptance testing
  + Process of verifying **User satisfaction**
* Regression testing
  + Testing previous tests to make sure **new integrations have not introduced malicious interactions**

## Test driven development

* When you write tests before writing code

## Types of testing

* Defective testing
  + When you purposely try to break an application
* Validation
* When you are testing to meet the requirement ONLY

Choosing Test Cases

The number of **Tests** should be governed by the number of **edge cases** in each testing field

### Partition Testing / Equivalence partitioning

* Identify groups of inputs with common characteristics and partition one from each groups

### Guideline based testing

* Using previous experiences to guide what and where to test

## Code Coverage

* Extent to which a source code has been executed by a set of tests
  + Method
  + Statement
  + Branch
  + Condition
  + Loop

### Functional Testing

* Performs all expected functions properly

### Non-functional testing

* Security, performance and usability

## Junit

* Unit testing Framework with Java

## **Junit annotations**

### @Test

* Identifies a **method** as a **test** **method**

### @Before

* Executed **before** each tests to **prepare** **test** **environment** (variables, reading input data etc.)

### @After

* Executed **after** **each** **test** to **clean** up the **test** **environment**

### @BeforeClass

* Executed **once** **before** the start of all tests to perform time **intensive activities**

### @AfterClass

* Executed **once** **after** all the tests to perform **clean-up activities**

## **Assertion types**

### AssertTrue(Boolean, message), AssertFalse(Boolean, message)

* Checks whether Boolean is true or not

### AssertEquals(Expected, Actual, message)

* Tests whether two values are the same
* For arrays and objects, the reference is checked not the content of the arrays

### AssertNull(object, message), AssertNotNull(object, message)

* Tests whether the object is null or not

You can control the order of the tests but then why would you? It should be independent anyways

## **Exception handling in tests**

@Test(expected = IndexOutOfBoundsException.class)

This test will go through saying the test **will** throw the error

## **Testing Timeout**

@Test(timeout = 1000)

Time out is in **milliseconds,** so this will wait a second before saying **nope this failed**

## **Rules**

A way to add or redefine the behavior of each test method in a test class

### Temporary Folder

* Creates files and folders that are deleted when the test is finished

### Error Collector

* Lets execution of test to continue after first problem is found

### ExpectedException

* Allows in test specification of expected exception type and messages

### TimeOut

* Applies the same timeout to **all test methods** in a class

### RuleChain

* Allows ordering of test rules

# **Continuous Integration, Continuous Delivery And Deployment**

## Continuous integration

* It is a **practice** where the **developer** **integrates** **code** into **shared repository frequently**
* To **Ensure** that the **software** is always on a **working state**
  + The **application** is **built** and **tests** are done **automatically** in a **Regression** **testing** manner
* Minimise the duration and effort required by each integration
* Be able to deliver product version suitable for release in any moment

## Continuous integration is implemented via

* Version Controll System
* Automated build process
* Workflow Integrity by team members
* Configuration of the build system and testing process
* CI server to automate the process

## Maintain Effective Continuous Integration Workflow

## Regular Check-In

* + Small changes are less likely to break

## Create a comprehensive automated test

* + Unit testing
  + Component testing
  + Acceptance testing

## Keep the Build and Test process Short

* + So that it doesn’t bottleneck the process of integration

## Developer’s to manage their development workspace

## Use CI Server

* + Report generation
  + Automatic Testing
  + Application Building

## Team Practices

## Don’t Check in on a Broken Build

* + It is in the developer’s responsibility to fix it

## Always run all commit test locally or get your CI server to do it for you

## Wait for commit Tests to pass before moving on

## Never go home on a Broken Build

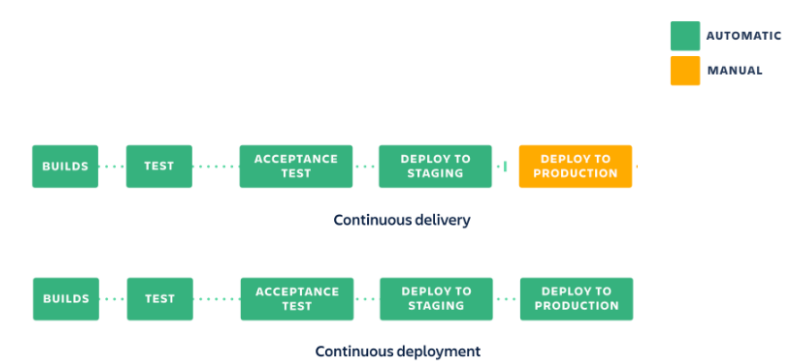
## Always be prepared to revert to the previous revision

## Time box fixing before reverting

## Do not comment out failing tests

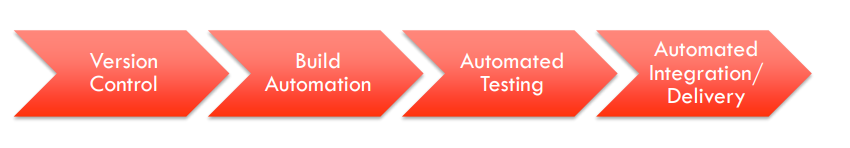
## Take responsibility for all breakage that results from your changes

# Continuous Delivery vs Continuous Deployment



Essential changes is the fact that deployment of product is done **automatically on Continuous Deployment** while **Manually on Continuous Delivery**

## Jenkins

****

* Automation server to automate tasks relating to
  + **Building**
  + **Testing**
  + **Delivering**
  + **Deployment**
* Running Multiple Steps
  + Retry
    - Retrying steps a number of times
  + Timeout
    - Exiting if a build takes too long
  + Clean up
    - run clean up step or perform post action based on the outcome of the pipeline

REPORT is generated using **Post** section.

If the test **fails** it will be marked as **unstable**

# Team dynamics

## Team dynamics in Waterfall method vs Agile

### Waterfall

* Teams are separated into clear authority lines
  + Specialist team
  + Project management
* Issue is that
  + There is a single point of failure
  + Inflexible
  + Lack of feedback from the client

### While in Agile

* Individuals and Teams are as important as anyone

## PEOPLE management

* Build project around **Motivated individuals,** give them the **environment and support** that they need and **trust** them to get their job done
* Face to face conversation
* Regularly reflects on method to **improve team effectiveness**
* **Businesspeople** and **Development teams** work together **throughout the project**
* **Self-organized team**

|  |  |
| --- | --- |
| Origin of Team dynamics  * Personality * Work style * Role * Culture | Solution Bad Team Dynamics  * Change the office layout * Team development workshop |

## Team Building Activities

### Goal Setting

* + Emphasize the importance of clear objective and individual/team goal

### Interpersonal relationship

### Stand up Meeting

### Roles and responsibility

* + Definition of clear roles and responsibilities
  + Identify required role
  + Clarify own responsibility
  + Clarify each team role’s responsibility

## Characteristics of High Performing Agile Team

### Cross functional skillset

### Mutual respect and mutual responsibility

### Sound Engineering practices

### Value and belief of afile practices and principles

### Apply agile practices effectively as individuals and as a team

### Receive continuous training and team monitoring

## AGILE PRINCIPLES

### Our highest priority is to **satisfy the customer** through **early** and **continuous** **delivery** of valuable software

### **Welcome** **changing** **requirements** even late in development.

### Deliver **Working** **software** **frequently**

### **Business people** and **developers** must **work together daily** throughout the project

### **Build** projects around **Motivated** **individuals**, give them the **environment** and **support** they need and **trust** them to get the **job** **done**

### **Working** **Software** is the **primary** **measure** **of** **progress**

### Agile processes promote **sustainable** **development**.

* + The sponsors developers and users should be able to maintain a constant pace indefinitely

### **Continuous** **attention** to technical excellence and good design enhances agility

### The best architectures, requirements and design emerge from **self**-**organizing** **teams**

### At **regular** **intervals**, the **team** **reflects** on how to become more **effective** **then** **tunes** and **adjust** its **behaviours** **accordingly**

### **Individuals and interactions** > processes and tools

### **Working software** > comprehensive documentation

### **Customer collaboration** > contract negotiation

### **Responding to change** > following plans

## Extreme Programming (XP)

* Development and delivery of **very small increments** of functionality
* **Relies on constant code improvement**
* Test Driven Development

# **SCRUM**

Built on three pillars

* Transparency
* Inspection
* Adaptation

## Values

### Commitment

* + Personally commit to get the work done

### Courage

* + Members can go through tough problems

### Focus

* + Focus on getting the work done

### Openness

* + Open about work and challenges with performing work

### Respect

* + Team members respect each other

## Who is scrum for?

* Small team
* Cross functional
* Self-organizing
* Deliver product iteratively

## **SCRUM ARTIFACTS**

|  |  |  |
| --- | --- | --- |
| **Scrum artifacts** | | |
| Items that can represent **work done** | | |
|  | Product Backlog | Sprint Backlog |
| Who is responsible | Product Owner | Development Team |
| What is it for | * User Story * Enhancement and fixes from previous sprint * Requirements | Set of items to be delivered by the end of sprint and a plan on how to do it. |
| What is included | It is a simple spreadsheet of some items that are   * customer featured   + UX design   + Interaction Scenario   + Use Case * internal tasks   + internal functionality | Comprised of   * To do * In progress * Done |

## **TEAM ROLES**

### **Product Owner**

* Understand requirement and it’s priority
* Manage the Product Backlog
  + Record product backlog items and order it
  + **Optimize the value** of the work the **development team performs**
  + Ensure **transparency** and **clarity** of the **product** **backlog**
  + Ensure the **development** **team** understand **product** **backlog**

### **Scrum Master**

* Keeps the team focused on using SCRUM properly
  + Service to **Product Owner** to ensure
    - Mutual **understanding** of **goals, scope and product domain**
    - Effective way for **managing** the **product backlog**
    - SCRUM team **understands** the **need for clear and concise product backlog items**
    - **Arranging** product backlog to **maximise value**
    - **Facilitate SCRUM events**
  + Service to **DEV team**
    - **Coaching** the **DEV team**
    - **Facilitate** SCRUM **event**
    - Remove **impediments** to DEV team’s progress
    - Create **high-value product**

### **Development Team**

* People who do the work of delivering a potentially releasable product
  + Self-organising
  + Cross functional
  + No sub-teams
  + Whole team is accountable

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Scrum Event** | | | | | |
| Sprint is a short, time-boxed period when a scrum team works to complete a set amount of work. | | | | | |
|  | Sprint Planning | Daily Standup | Development Work | Sprint Review | Sprint Retrospective |
| Length | Two 4 hour meetings | 15 minutes | 2-4 Weeks | 4 hours | 1-3 hours |
| Who is included | * Product Owner * Dev Team | * Product Owner * Scrum Master * Dev Team | * Dev Team | * Product Owner * Scrum Master * Dev Team * Stake Holder | * Scrum master * Dev team * Product Owner(optional) |
| What to do | * **Product Owner** and **Dev team** select **items to be delivered** * **Estimate** **time** and work required | **Briefings** on   * What they did * What they will be doing * Any challenges   **No problem solving during the meeting**   * Could be done in the **follow up meeting** |  | **Dev team**   * Demonstrates working software   **Stake hodler**   * Provides feedback   **Product Owner**   * Updates the Product backlog with change in requirements   **Scrum Master**   * Facilitates the event | **Reflection** on themselves   * What to **improve** * What went **well**   **Scrum** **master**   * Adds **improvements** in the **Backlog** |
| Breaking down **defined user** story into **tasks**   * Dev team |
| Outcome | Sprint Backlog |  | Working Software | Revised Product Backlog  Probable item for next iteration | Potential improvements for next sprint |

# **Requirements**

## Plan and document driven software development

Aim to make **predictable budget and schedule**

* + Requirements
  + Cost
  + Scheduling
  + Monitoring
  + Change management
  + Risk analysis
  + Testing

## Method to Collect Requirements

### Interview Stakeholders

* + Information discussions and formal questions

### Cooperatively create scenario

* + Initial state, happy/sad paths, concurrent and final state

### Create use Cases

|  |  |
| --- | --- |
| **Functional requirement** | **Non-functional requirement** |
| * What information goes in and out * Interactions between features * UX * functionality | * Performance * Security * usability |

* + User and system interactions to realise functions

**Stakeholders** should **manage requirement** checking’s as well when **creating requirements**

* Validity
* Consistency
* Completeness
* Feasibility

# **Behaviour Driven Development**

* **Conceptual** approach for **specifying application’s behaviour**
* Business value (features) >> Acceptance criteria >> code to deliver
* **Given – When- Then canvas**

## User Story

* Short and concise description of how application is expected to be used
* Simple lay language
* **As a- I want to be able to – So that**
* Removes any unnecessary functionalities that users wouldn’t use

## SMART

##### Specific

##### Measurable

##### Achievable

##### Relevant

##### Time-Boxed

* 5 whys to uncover business values/needs

## Acceptance Criteria

* Condition of satisfaction to be written at the back of

## User Interface Sketch

### Low-Fidelity UI

* Rough sketch with low tech approach to show how UI looks like

### High-Fidelity UI

* More defined and descriptive design approach

### Scenario

* Written description of the system interaction from **user’s perspective**

### Story Board

* Similar to scenario but visual

### User Story

* Description of a single feature

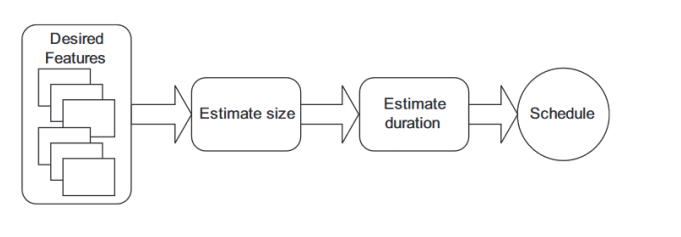
### Scenario Format

* Scenario title
* Given
* When
* Then
* (And Then)

|  |  |
| --- | --- |
| Implicit Scenario | Explicit Scenario |
| * Features that are not specified in the user story * Integration Testing | * Requirements that are explicitly specified in the user story * Acceptance testing |

|  |  |
| --- | --- |
| Imperative Scenario | Declarative Scenario |
| * Tend to have complicated **WHEN** statements and a lot of **AND** steps * Ensure that UI details match customer expectations | * Focuses on the **features being described** by using the **step definition** to make a **domain language for the application** * **Domain Language**   + **Informal** and **specific** to your **application** rather than a generic term |

# **Estimation and Progress Tracking**



## Estimating Size of feature – Story point

* Metric value for the overall size of a user story, feature or piece of work
* # of story points ~ Overall Size of feature
* Scale could be either Fibonacci or sequential doubles
* This is aimed to
  + Measure the user story only
  + No emotional measure
  + Team velocity is considered

|  |  |
| --- | --- |
| Ideal Day | Elapsed Day |
| The amount of time user story will take to develop | The amount of time development will wake **with consideration of interruptions** |

## Estimation Technique

### Expert option

### Analogy: Comparing previous Experiences

### Disaggregation – Breaking stories into smaller tasks

### Planning Poker

# **PLANNING POKER**

* Process of selecting a Story point easier and interactively

1. Select a base feature to make it a 1 and make everyone agree to it.
2. Select each feature and allow the team to choose the item’s Story point
   1. No one is able to know each other’s points as of yet
3. Reveal and see the values

a. If there is any outliers or disagreements, let them talk about why they would choose that number

1. Discuss until there is a mutual agreement amongst the team
2. Repeat until the backlog is completely done.

* This is aimed to **avoid individual influences/bias**

|  |  |
| --- | --- |
| Planning poker > Idea day | Ideal Day > Planning Poker |
| Story points are more Constructive and metric | Easier to explain |
| Estimation process is faster | Easier to roughly estimate |
|  | Velocity predictions easier |

## **Velocity**

Measure of **teams rate of progress**

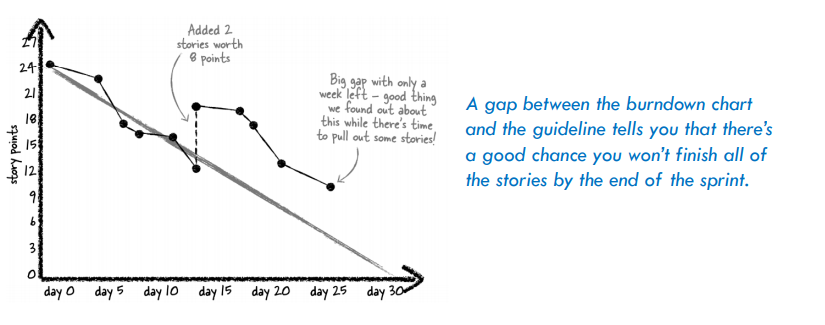
**Velocity = Number of SP done / Time it took to complete**

### **Estimating Velocity**

* Historical Value
  + Unreliable because each iteration is never the same
* Run an Iteration/sprint
  + Dip your feet in the water to see the rate
* Make a forecast
  + Estimate the available hours per person
  + Randomly select stories and expand them into basic tasks, repeat until number of hours are filled.
  + Convert the velocity determined in the prior step into a range

## Burn Down chart

* X-axis: first and end date of the sprint
* Y-axis: Story points that are Currently left
* Comparison between the number of story points left with ideal progress



# Ethics and Professional Frameworks

|  |  |
| --- | --- |
| Ethics vs Morals | |
| Ethics | Morals |
| Rules of conduct accepted within a social Context | Principles of right and wrong that guide personal behaviour |
| Externally defined | Internally defined |

## Professional Framework

* Primacy of Public interest
* Enhancement of Quality of Life
* Honesty
* Competence
* Professional Development
* Professionalism

|  |  |  |
| --- | --- | --- |
| General Ethical Principles | Professional responsibility | Professional Leadership principles |
| * Society and human well being * Avoid conflicts * Be honest and trustworthy * Be fair * Respect work, privacy * Honour confidentiality | * Quality of work and processes * Quality of review and judgement * Competence | * Quality of work * Training and competence * Public interest decisions to use or retire system |

# Privacy

* Access computing and communication resources only when authorized or when compelled by public good
  + Individuals and organisations have the right to **restrict access tot heir system and data** so long as the restrictions are consistent with other principles

## Intellectual property

* Category of property that includes intangible creations of human intellect
  + **Intangible**
    - Difficult to protext

## IP Types

|  |  |  |
| --- | --- | --- |
| Copy Right | Trade Mark | Trade Secret |
| * Grant a creator of an original work clusive right * Granted automatically in Australia * Does not protect ideas or method of implementation | * Sign, design or expression which distinguishes products or services * Formal registration and require meeting certain criteria | Process, formula, design, pattern which is unknown and provide a competitive advantage |

## Patent

Form of IP protection which grant inventor the right to exclude others from making, using, selling or importing an invention for a limited time

* Inventor
  + Exclusive rights to the invention
* Public
  + Enrich body of knowledge and innovation
* Criteria
  + New
  + Non-obvious
  + Useful
* Local Protection vs international protection

## Licensing

## Open Source

* Software with source code that anyone can inspect, modify and enhance
* Something people can modify and share because it is **designed to be publicly accessible**
* **There are conditions** onto how they use, study, modify and distribute software
* **Consequent distribution** after modification should also be **open source**
* Focuses on distribution and collaboration
* Does not need to be free of charge

## Free Software

* Ability to run,copy, change, distribute and improve source code .

Open source focuses on availability of the source code and **ability to modify and share**

* Development methodology

Free software focuses on User’s **Freedom** to use program, modify it and share it.

* Social movement

# Copyleft

* Practice of providing freedom to use the software and distribute the application in condition that the licence is preserved
* Any changes made onto the code should also be open source and distribution should not be paid

# Open Source Licenses

## MIT License

* Short and simple permissive license
* Condition only includes preservation of copyright and license notice

|  |  |  |
| --- | --- | --- |
| Permission | Condition | Limitation |
| * Commercial use   + commission * Distribution * Modification * Private use | License and copyright notice | Liability  Warranty   * Any damage done from using the source code is in the responsibility of the implementor and not the creator |

## Apache License 2.0

* Requires preservation of copyright and license notices
* Contributor provide an express grant of patent rights
* Cannot use the software as a trade mark
* You must document changes made onto software

|  |  |  |
| --- | --- | --- |
| Permissions | Conditions | Limitations |
| Commercial use  Distribution  Modification  Patent use  Private use | License and copyright notice  State changes | Liability  Trademark use  Warrenty |

## Mozilla Public License 2.0

* Copyright and licenses notice preserve

|  |  |  |
| --- | --- | --- |
| Permissions | Conditions | Limitations |
| Commercial use  Distribution  Modification  Patent use  Private use | License and copyright notice  Disclose Source   * Any changes you need made and source code   Same License file   * Modification of source code should be released on the same license | Liability  Trademark use  Warrenty |

## Affero General Public License 3.0

|  |  |  |
| --- | --- | --- |
| Permissions | Conditions | Limitations |
| Commercial use  Distribution  Modification  **Patent use**  Private use | Disclose source  License and copyright notice  Network use is distribution  Same License  State changes | Liability  Warrenty |

## General Public License 3.0

Public availability onto complete source code of licenced work and modification

|  |  |  |
| --- | --- | --- |
| Permissions | Conditions | Limitations |
| Commercial use  Distribution  Modification  Patent use  Private use | Disclose source  License and copyright notice  Same license  State change | Liability  Warrenty |

## Lesser General Public Licence

|  |  |  |
| --- | --- | --- |
| Permissions | Conditions | Limitations |
| Commercial use  Distribution  Modification  Patent use  Private use | Disclose source  License and copyright notice  Same license (library)   * If you use the software **as a library** the condition does not apply   State change | Liability  Warrenty |

## The Unlicense

No conditions in distribution

|  |  |  |
| --- | --- | --- |
| Permissions | Conditions | Limitations |
| Commercial use  Distribution  Modification  Patent use  Private use |  | Liability  Warrenty |