Introduction to Software Engineering and Software Process Models

> *Introduction to Software Engineering and Software Process Models Slide*

Learning Objectives #todo

- [x] ~~Intro to Software Engineering~~
- [x] ~~Software Process~~
  - [x] ~~Software Phases~~
- [x] ~~Software Process Models~~
  - [x] ~~Waterfall Model~~
  - [x] ~~Evolutionary Development~~
  - [x] ~~Process Iteration~~
    - [x] ~~Incremental Development~~
    - [x] ~~Spiral Development~~
  - [x] ~~Rational Unified Process RUP Model~~
  - [x] ~~Modern Software Models~~
    - [x] ~~Agile~~
      - [x] ~~Scrum~~
      - [x] ~~DevOps~~

Intro to Software Engineering

1. **Software Engineering** is focused on developments of *large and complex software intensive systems*
   - ***Real World Goals***, ***Services*** it provides and ***Constaints*** on systems
     - What should the System be able to do in the end?
   - ***Precise specification*** and ***implementation*** of those specifications
     - *Behaviour and Structure is considered*
   - ***Activities*** considered to ensure those specs and goals are met
     - What sort of things does the system needs to do and how should it react?
   - ***Evolution*** of those system over time (+ system families (Think IPhones!))
     - How does it improve overtime?
   - ***Processes***, ***methods*** and ***tools*** for the development
     - Is it economically friendly?
     - How long will it take?

Software Process

1. **Software Process** tells us *how a software is designed, implemented and then released.*
   - This can be categorised into several activities (or 'Phases') which better simplifies the concepts.
     - These activities are structured to help develop a software systme

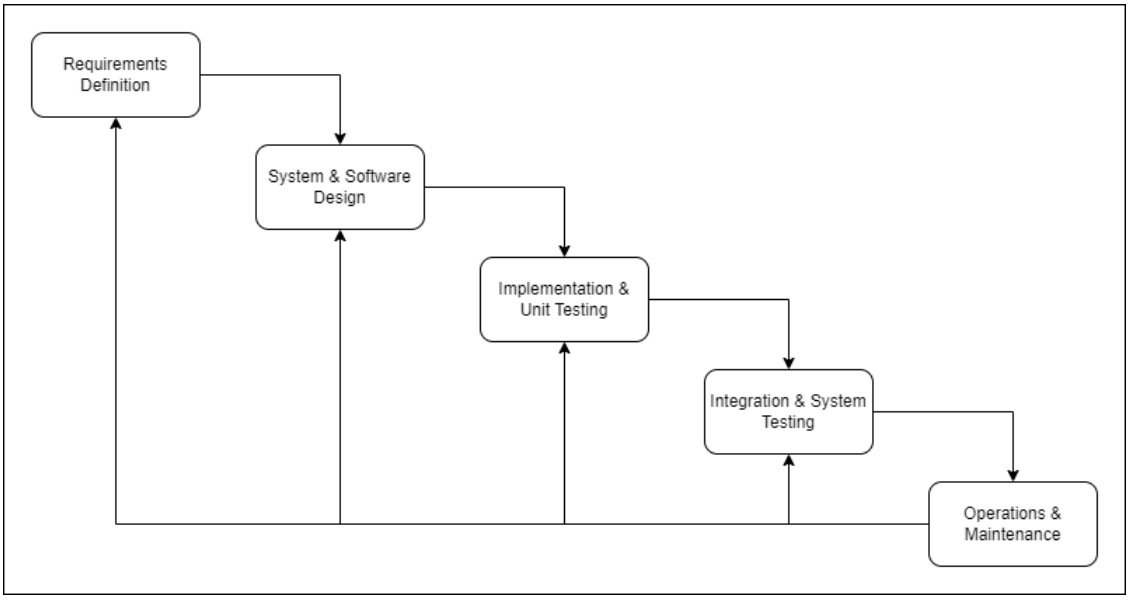
> **Software Phases**
>
> 1. **Specification** - Find what kind system we want to develop
> 2. **Design** - Plan our software system
> 3. **Implementation** - Code our software system
> 4. **Verification and Validation** - Check if code works as intended

5. **Maintenance** - Respond to problems found after release

Software Process Models

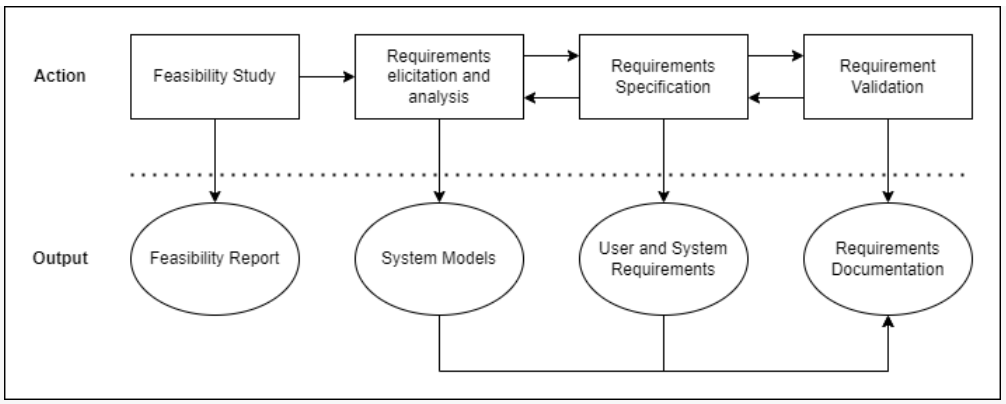
1. **Software Process Models** (SPM) is an *abstract representation of a Software Process*.
   - This helps to *describe the process in different perspective*.
2. Focused Models for this Module:
   1. Waterfall Model
   2. Spiral Model
   3. Evolutionary Development
   4. Modern Software Models: Agile, Scrum, DevOps
3. Example of Generic Process Models:
   - Waterfall Model - Separate and distinct phases of Specification and Developments
   - Evolutionary Development - Specification, Development and Validation are interleaved and evolved during the project
     - Agile Practices of Iterative and Continuous Revision
   - Component-based software engineering - System is assembled from existing components

Waterfall Model



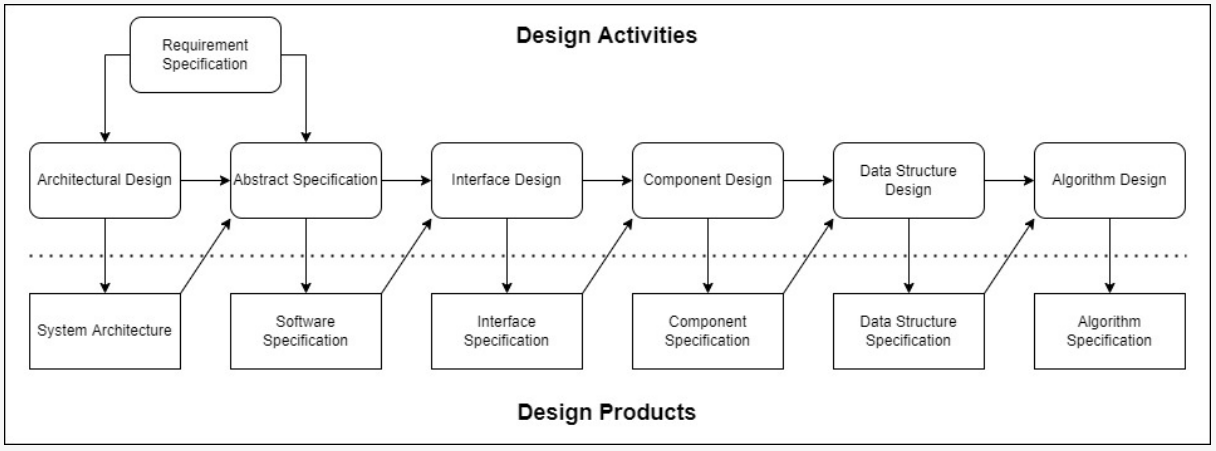
1. The Linear Sequential Life Cycle (or 'Watefall') Model *describes that each phase in the model must be completed fully before continuing to the next*.
   - As previously mentioned, the five phases describes the entire model

> **Watefall Model Phases**
>
> 1. **Requirement Analysis and Definition** - Establish what services and constraints are required:



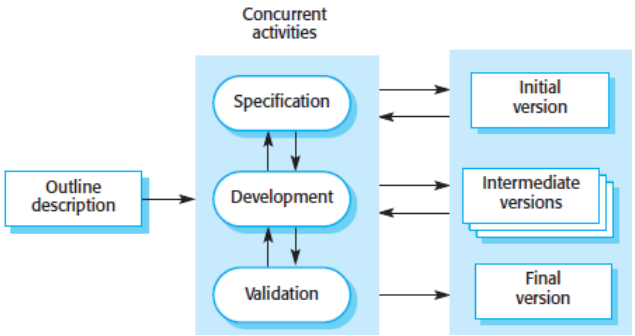
2. **System and Software Design** - Design a software structure to meet those requirements and constraints

> *Main design should be: **Abstract Specification, Data Structures and Algorithm Design**:*



3. **Implementation and Unit Testing** - Using the design to implement a code and testing each component of the code independently
4. **Integration and System Testing** - Testing the entire system as a whole and test important properties (Performance, security, etc).
   - We can do an *Acceptance Testing* (Test with Customer Data)
5. **Operation and Maintenance** - Deploying the software and continue to test and solve problems brought up by the community.
   - We can deploy updates (Evolution) to implement new features

Evolutionary Development



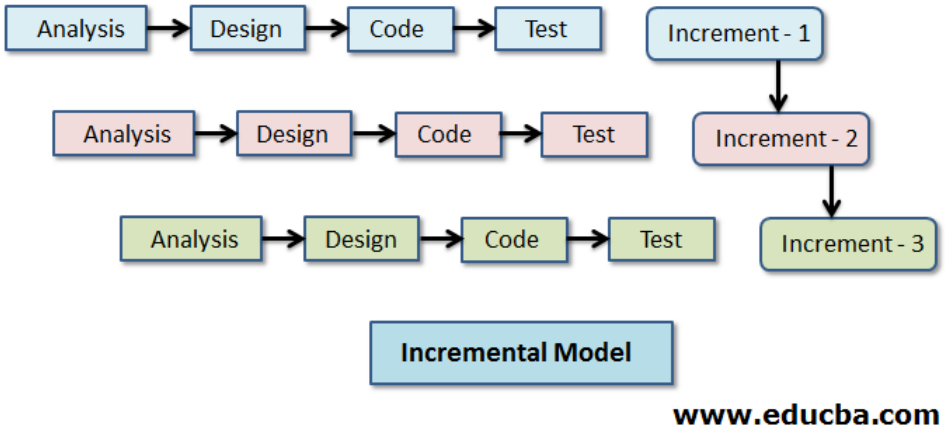
1. In Evolutionary Development, is *essentially a collection of smaller incremental* [*Waterfall Model*]. And it can be describe by the following:
   - **Exploratory Development** - Starting with a well-understood requirements and add features as proposed by the customer (like using Incremental) until it is acceptable
   - **Throw-Away Prototyping** - Understand the system requirements and have poor understanding of the requirements to clarify what is really needed.
2. In sense, ED is the idea to **gradually improve the software** through *discussing with the client about their thoughts and ideas on the current version*.

Process Iteration

1. **Evolution of System Requirements** will *always be a constant issue in which we need to rework the system may be required*.
   - **Iteration** can be applied to any generic proecss models.

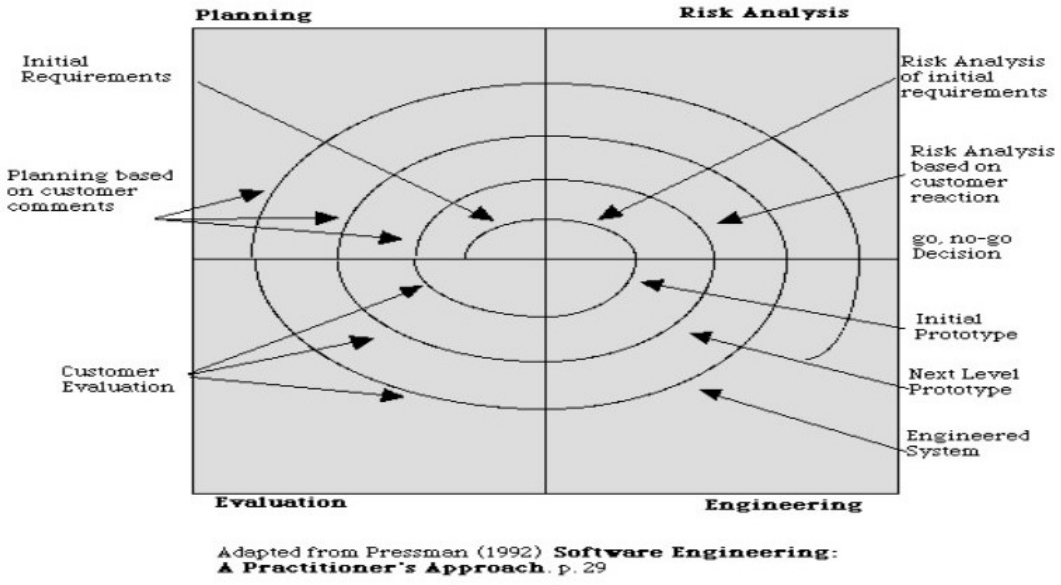
2. Two approaches are:
    1. Incremental Development
    2. Spiral Development

Incremental Development



1. **Development and Delivery** is *broken into smaller increments* where each increment we aim to *develop a required functionality one at a time with keeping in mind the feedback between each increment.*
    - Note that User Requirements are prioritised and done in early increments
    - When we start an increment development, we do not change the requirments but requirements for later increments can continue to evolve
2. Advantages:
    - Early increments acts as a prototype (which gives us requirements to work on later increments)
    - Failure of the overall project is reduced
    - Highest Priority System Services tend to recieve the most testing
    - Customer values is delivered with each increment (system functionality is avaialble earlier)

Spiral Development

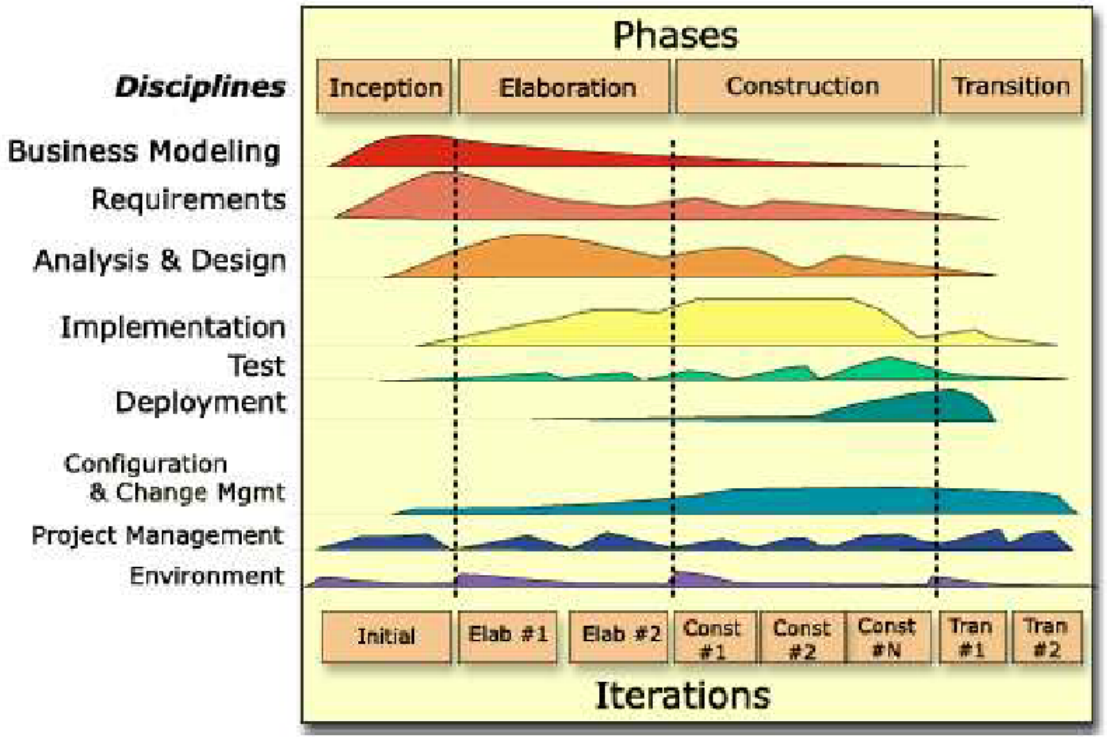


1. The **Spiral Model** can be thought as a Watefall Model with no backtracking but returning to the first phase and go through each phase again as many times. (Iterative)
    1. Each loop *represents a phase* and there are *no fixed phases*
        - Loops in the spiral are chosen *depending on what is needed*
    2. Risks are *(explicitly) assessed and resolved* throughout the process
    3. Many prototypes are built as part of the process

2. **Spiral Model Sectors**

1. **Objective Settings** - Specific objectives for the phase
2. **Risk Assessments and Reduction** - Risk identified and activities to reduce those risk
3. **Development and Validation** - A development model is chosen, develop the prototype and test it.
4. **Planning** - Evaluate the project and plan the next phase of the spiral

Rational Unified Process (RUP) Model



-

1.

> **RUP Model Phases**
>
> 1. **Inception** - Propose the project (Define the problem and constraints)
> 2. **Elaboration** - Develop an understanding of the problem domain and system architecture
> 3. **Consturction** - System Design, Programming, and Testing
> 4. **Transistion** - Deploy the system in its Operating Environment

2. Revisits amongsts the phases may occur but the magnitude (As shown by the shapes on each section) will vary
3. This focuses on the product itself and its quality, emphasises on addressing high risks at early stages
4. It is a Generic Process Model that separates activities from phases

---

Modern Software Models

Agile

1. The old approaches were organised and carefully planned, there was still some customer unsatisfaction due to:
   - Out of Budget, Time, Scope and Quality
     - *This might be due to things they asked for are not actually needed*
2. Agile focuses on **four values and twelve principles**.

> **Agile Values**

1. Individuals and Interactions over process and tools
2. Working Software over Comprehensive Documentation
3. Customer Collaboration over Contract Negotiation
4. Responding to changes over following a plan

Agile Principles

1. Customer Satisfaction
2. Embracing Changes
3. Speed Delivery
4. Collaboration
5. Empowerment
6. Effective Communication
7. Good Metrics
8. Steadiness
9. Operational Excellence
10. Simplicity
11. Self-Organisation
12. Continuous Improvements

- Because of this we can improve performance and reach satisfaction

3. Agile is a *group of Software Development Methods based on highly imperative and incremental development*. For example: Test Driven Development (TDD)
4.

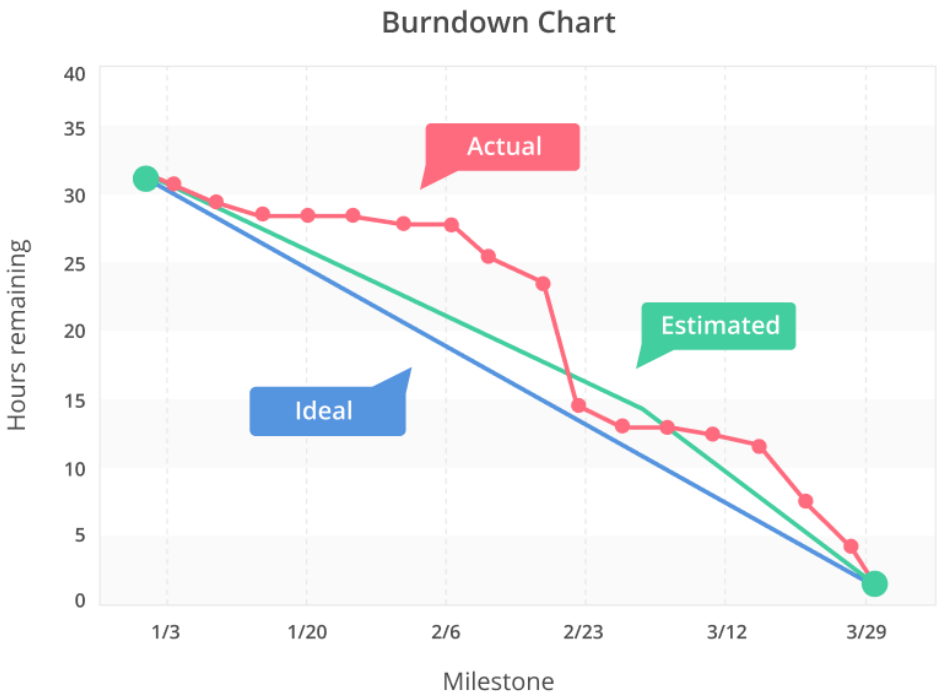
Test Driven Development

There are three main phases throughout this model:

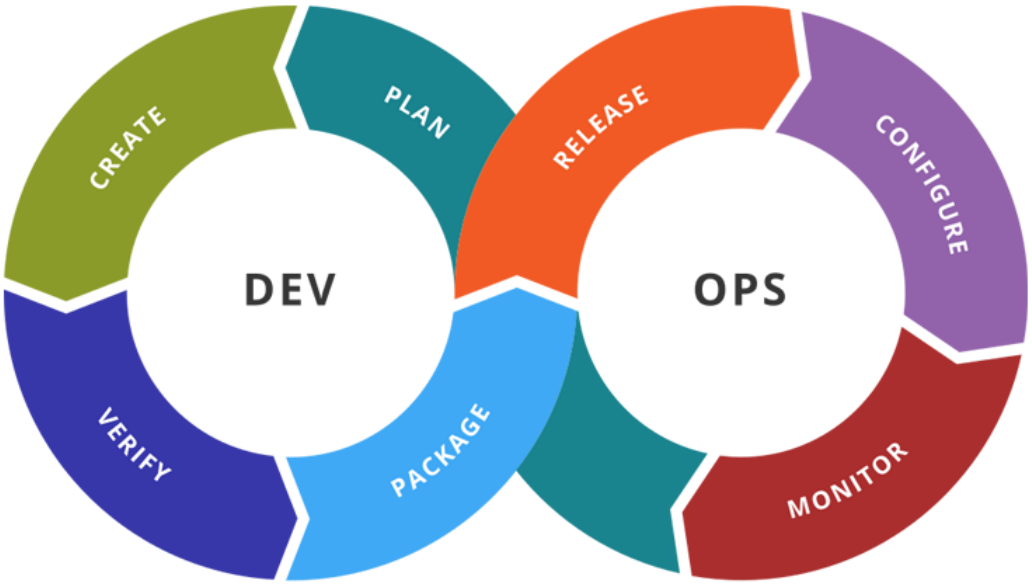
1. **Red (Fail)** - Write test cases (for requirements) that fails as there wouldn't be any code for it
2. **Green (Pass)** - Make enough code so that the test case pass but we will notice that the rest of the code may deteriorate
3. **Refactor** - Modify to improve design and adequality
4. Repeat

Scrum

1. *Features are written in the perspective of end users* (**User Stories**) and a collection of these stories is called the '**Product Backlog**' which contains all the things that makes the product great.
   - This means we need to pick out which stories that are included in the product
2. To begin to use the Scrum Methodology, we need to *employ specific roles*:
   1. **Product Owner** - Ensures the right features makes to the Product Backlog, represents the user and customers and sets the direction
   2. **Scrum Master** - Ensures the project is running smoothly and essentially the mastermind of the group
   3. **Development Team** - These people are our Developers and Testers
3. How ***Release Planning*** is made:
   1. ***Identify User-Stories*** to put in the release and becomes the 'Release Backlog'
   2. ***Priortise*** each stories and ***estimate*** the time needed to work required for each of them
   3. **Plan 'Sprints'** (Short Duration Milestones) to *tackle a managable chunk of the project* (Basically breaking the large problem into smaller managable parts) where at the end of each sprint is a product with 100% completion of each stories.
      - In the end, a final product is made!
      - A **burndown chart** is used to *monitor the amount of work remaining to finish a sprint or release* (Hence, why Scrum is popular).
        - The **slope of the graph (Burndown Velocity)** is the *average rate of productivity per day to help predict an estimated completion date*
        -



DevOps



1. **DevOps** allows *Developers and Operators to work better*.
   - It aims to shorten the systems development life cycle and provide continuous delivery with high software quality
   - This *Improves collaboration and productivity* by automating infrastructure, workflows and continuously measure application performance
2. DevOps teams will want to **automate EVERYTHING** (Code Testing, Workflows, Infrastructure, etc)
   - Aim to break down software into smaller parts and work on them individually (They'll be integrated, tested, monitored and deployed) in short periods.
     - Verus coding the entire thing which requires more time to test
     - The Development and Production environment would be the same!
   - Because of this method, the frequency of deployment increases and therefore new code can be deployed quickly.
3. Enables to adopt an iterative process to monitor, measure and improve the code and operation everyday to allow teams to respond to market needs or any things that impacts the software
   - Source Control Systems (Like Git) allows easy manage, track and document to track their code!