Software Development Methodologies

Contents

SDLC, methodologies, good software

Software Engineering

Waterfall, iterative, Agile (Scrum, LEAN, Kanban)

SDLC

types, methods, levels

Application Testing

AWS

types, characteristics, providers,
client-server, clusters, virtualization,

Cloud Computing

automation

devops, operations, tools, lifecycle ∞

DevOps Introduction

CVCS, LVCS, DVCS

VCS, types, git, local, GitHub, Remote

SCM

Bamboo

CI, CD → tools → Jenkins, TravisCI,

CI/CD Pipeline

physical, VM, containers

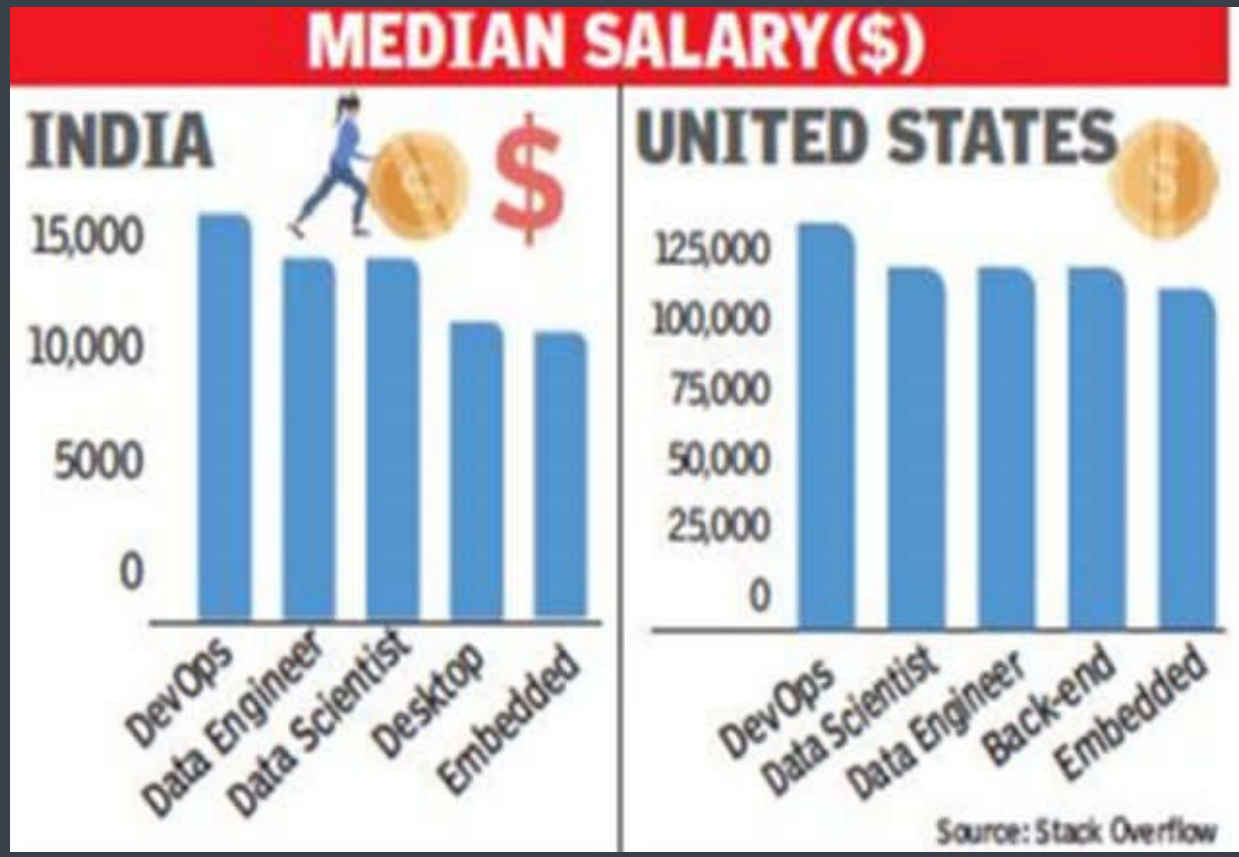
deployment, Docker, Highly Available, Horizontal scaling

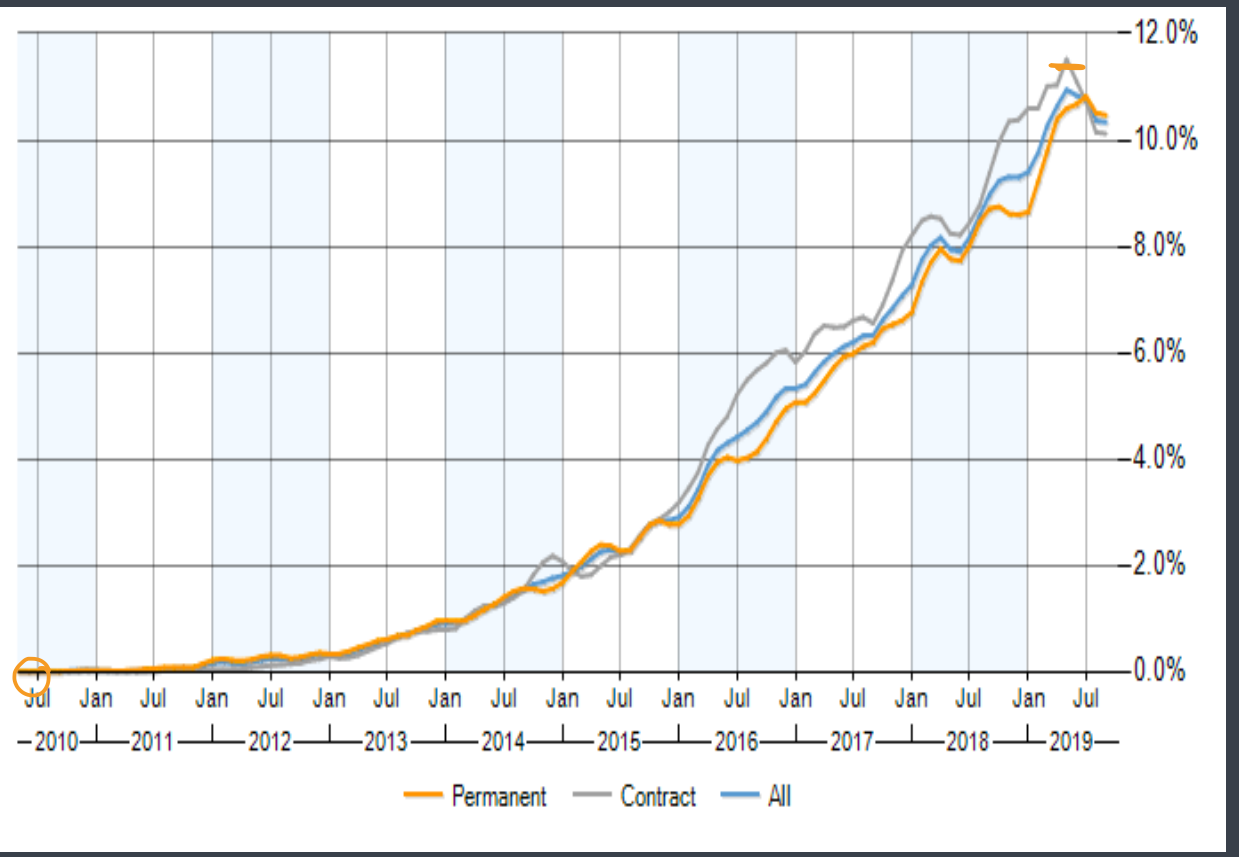
Containerization

fundamentals, docker swarm, kubernetes

Container Orchestration

Why should you bother about DevOps?





Pre-requisites

- Basic Linux knowledge
- Any Development Platforms
- Any language (preferred JS/Java)
- Willingness of learning something new

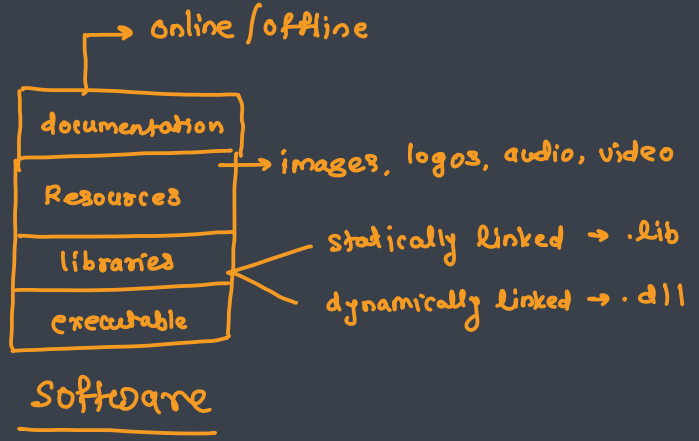
About Instructor

- 13+ years of experience
- Associate Technical Director at Sunbeam
- Freelance Developer working in various domains using different technologies
- Developed 180+ mobile applications on iOS and Android platforms
- Developed various websites using PHP, MEAN and MERN stacks
- Languages I love and use in every programming: C, C++, Python, JavaScript, TypeScript, PHP , Java , Kotlin , Swift

Software Engineering

Introduction

- Software
  - Software is more than just a program code
  - A program is an executable code, which serves some computational purpose
  - Software is considered to be collection of executable programming code, associated libraries and documentations
  - Software, when made for a specific requirement is called software product
- Engineering
  - All about developing products using well defined principles, methods and procedures



What is Software Engineering?

- Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures
- The application of a systematic, disciplined , quantifiable approach to the development, operation and maintenance of software
- Establishment and use of sound engineering principles in order to obtain software that is reliable and work efficiently on real machines
- The process of developing a software product using software engineering principles and methods

Software Types

- System Software → Operating System, device drivers
- Application Software → blender, 3D max, Houdise
- Engineering/Scientific Software → Catia, Idea, AutoCAD
- Embedded Software → RTOS, IoT
- AI Software → chatGPT, Bard
- Legacy Software → Calculator
- Web/Mobile Software → Websites → amazon.com, etc
  mobile → contacts, Camera

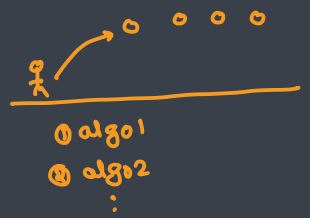
Why SE is important

budget

- Helps to build complex systems in timely manner
- Ensures high quality of software
- Imposes discipline to work
- Minimizes software cost
- Decreases time

Software Evolution

- **S-type (static-type)** → logic / formula
  - Works strictly according to the pre-defined specifications and solutions
  - Solution and method to achieve it can be understood immediately before coding starts
  - Least subjected to the changes
  - E.g. Calculator program for mathematical computation
- **P-type (practical-type)** → Games
  - Software with a collection of different procedures
  - Is defined by exactly what procedures can do
  - The specification can be described and solutions are not obvious instantly
  - E.g. Gaming software



- **E-type (embedded-type)** ***
  - Works closely as the requirement of real-world environment
  - Has a high degree of evolution as there are various changes in laws, taxes etc. in the real world
  - E.g. online trading software

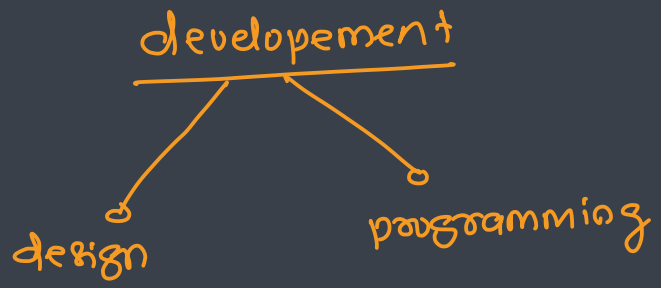
E-Type software evolution laws

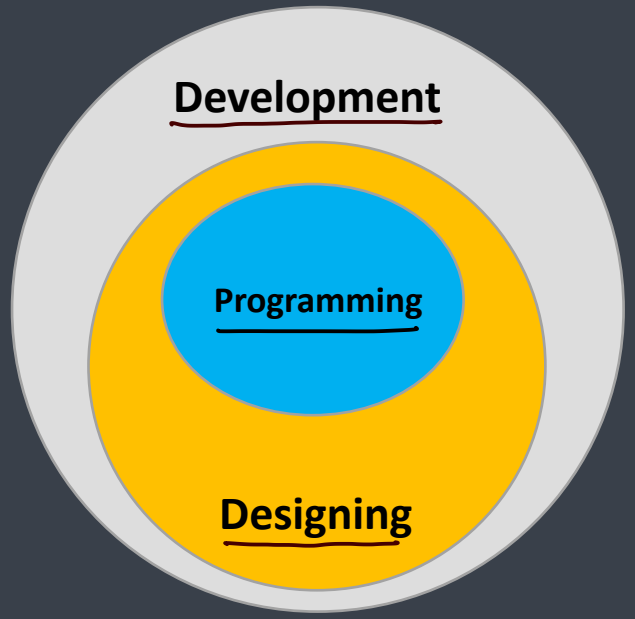
- **Continuing change**
  - An E-type software system must continue to adapt to the real world changes, else it becomes progressively less useful
- **Increasing complexity**
  - As software evolves, its complexity tends to increase unless work is done to maintain or reduce it
- **Conservation of familiarity**
  - The familiarity with the software or the knowledge about how it was developed, why was it developed in that particular manner etc. must be retained at any cost, to implement the changes in the system
- **Continuing growth**
  - In order for an E-type system intended to resolve some business problem, its size of implementing the changes grows according to the lifestyle changes of the business
- **Reducing quality**
  - An E-type software system declines in quality unless rigorously maintained and adapted to a changing operational environment
- **Feedback systems**
  - The E-type software systems constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.
- **Self-regulation**
  - E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal.
- **Organizational stability**
  - The average effective global activity rate in an evolving E-type system is invariant over the lifetime of the product.

Software Paradigms

- Refer to the methods and steps, which are taken while designing the software
- There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand
- These can be combined into various categories, though each of them is contained in one another
- Consists of
  - Requirement gathering
  - Software design
  - Programming

Good software





Characteristics of good software

- A software product can be judged by what it offers and how well it can be used
- Well-engineered and crafted software is expected to have the following characteristics
  - Operational
  - Transactional
  - Maintenance

Operational → functional

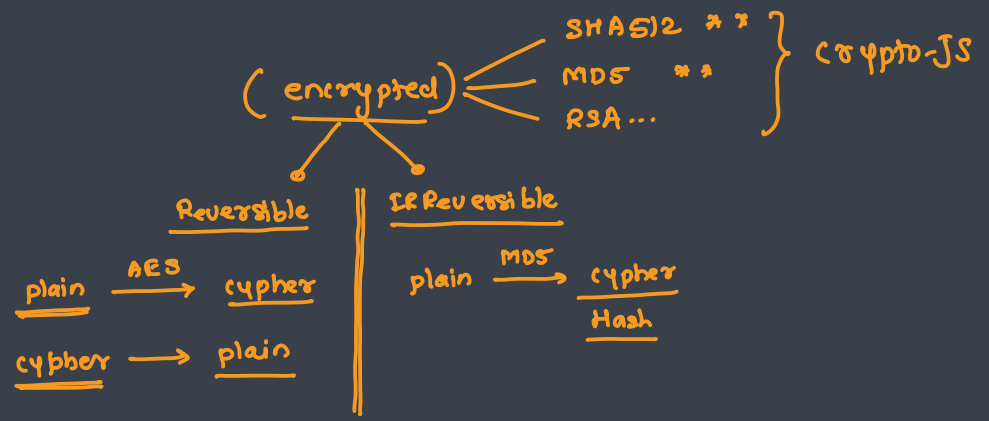
- This tells us how well software works in operations
- Can be measured on:
  - **Budget**: Software should be developed within the budget
  - **Usability**: Software should be usable by the end user
  - **Efficiently**: Software should efficiently use the storage and space (→ performance)
  - **Correctness**: Software should provide all the functionalities correctly without having any bug/issue
  - **Functionality**: Software should meet all the requirements of the user
  - **Dependability**: Software should contain all the dependencies in its package
  - **Security**: Software should keep the data safe from any external threat (****)
  - **Safety**: Software should not be hazardous or harmful to the envrionment

→ Authentication → application must be accessible to only valid users
↳ email / password

→ Authorization → application must be visible to only authorized (valid) users.

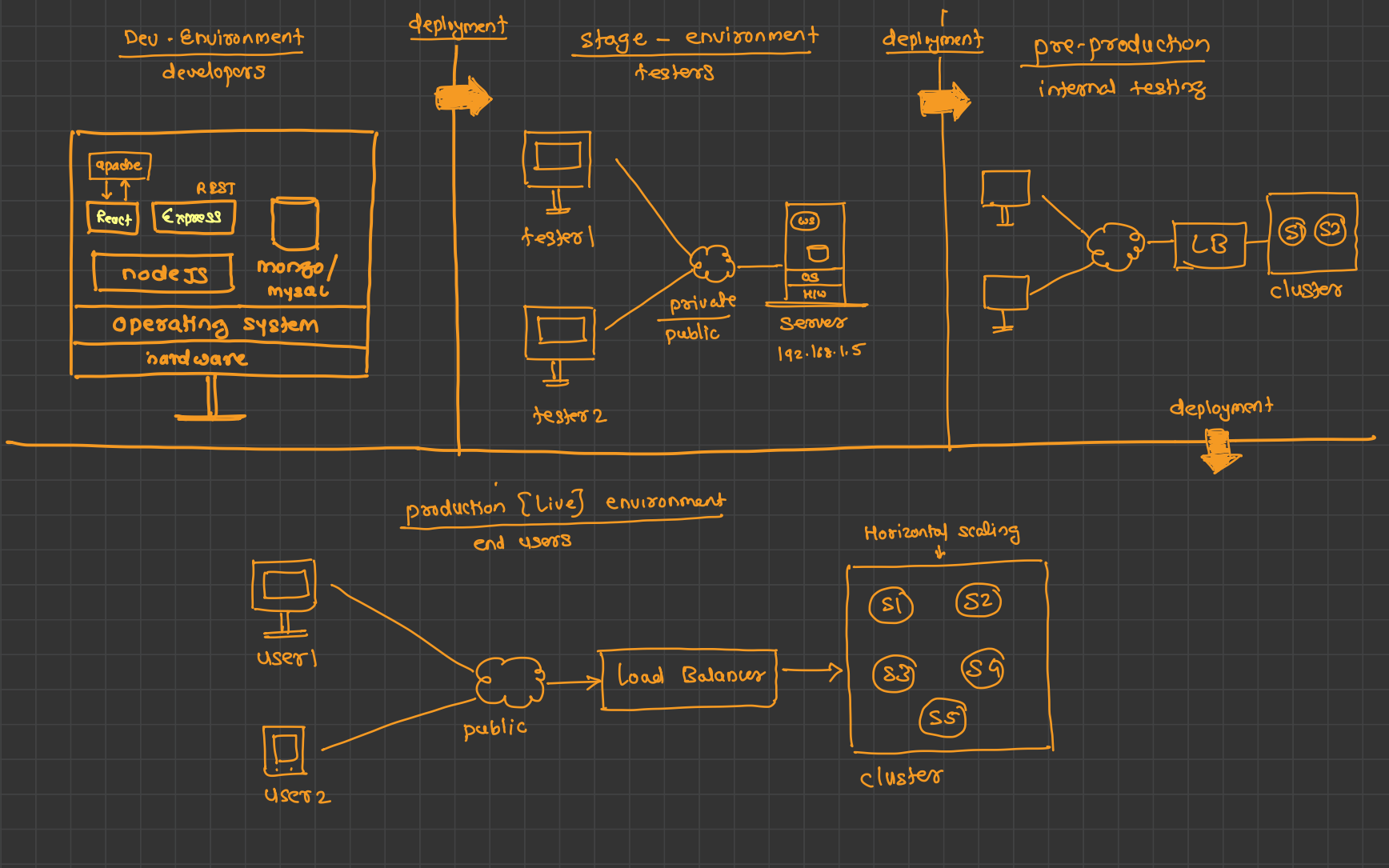
Security

- confidentiality → data must be protected
- Integrity → data must not change automatically
- Availability → the application must be available to users.
  ↳ scalability
  - Vertical
  - Horizontal



Transitional

- This aspect is important when the software is moved from one platform to another
- Can be measured on
  - Portability: If the software can perform the same operations on different environments and platforms, that shows its Portability
  - Interoperability: It is the ability of the software to use the information transparently
  - Reusability: If on doing slight modifications to the code of the software, we can use it for a different purpose, then it is reusable
  - Adaptability: It is an ability to adapt the new changes in the environment


Dev - Environment
developers
deployment
Stage - environment
testers
deployment
pre-production
internal testing
apache
REST
React
Express
nodeJS
mongo/
mysql
operating system
hardware
tester1
tester2
private
public
os
os
H/w
Server
192.168.1.5
LB
S1
S2
cluster
deployment
production [Live] environment
end users
Horizontal scaling
user1
user2
public
Load Balancer
S1
S2
S3
S4
S5
cluster

Stack (MERN)

JavaScript

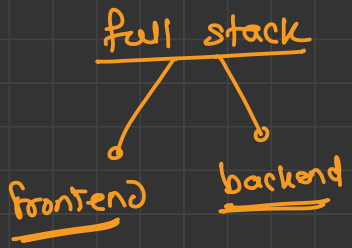
- M – mongo → database
- E – Express → web server
- R/A → React → framework / language for developing UI
- N → node → platform

Stack (LAMP)

- ← MySQL
- ← Apache
- ← PHP / Perl / Python
- ← Linux (OS)

Stack (WISA)

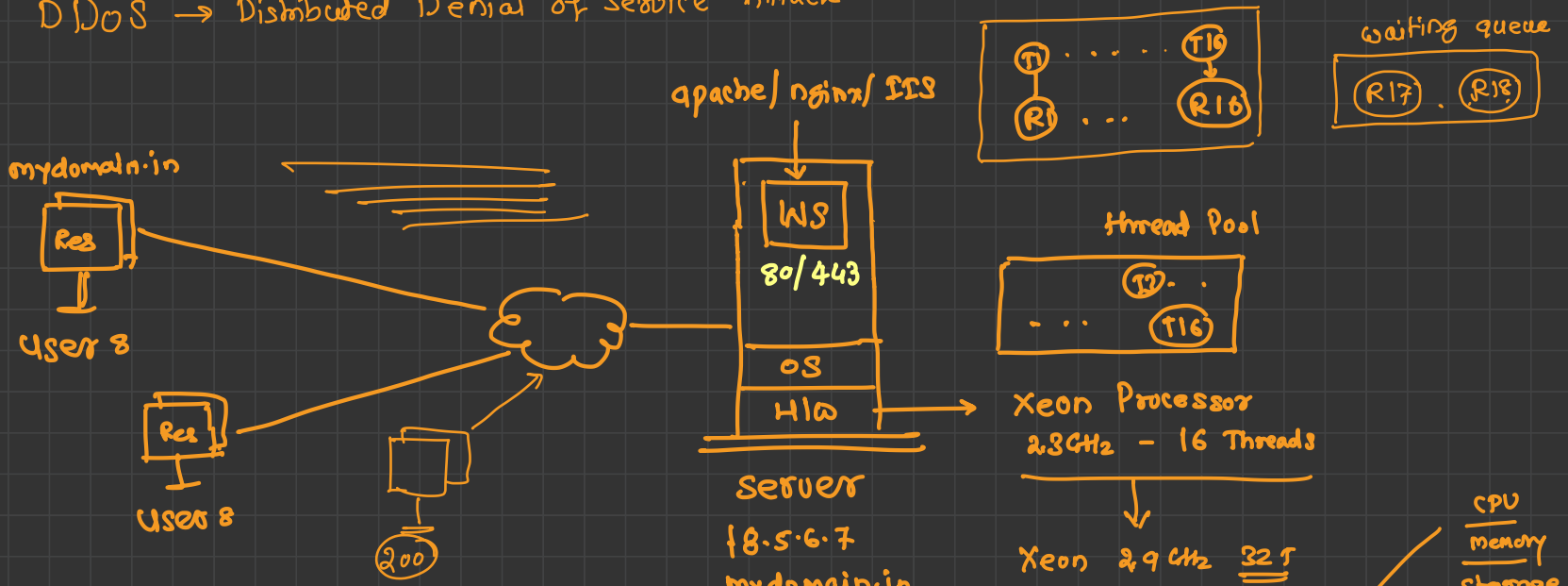
- ← SQL server
- ← IIS
- ← Asp.net
- ← Windows



Maintenance

- Briefs about how well a software has the capabilities to maintain itself in the ever-changing environment
- Can be measured on
  - Maintainability: The software should be easy to maintain by any user
  - Flexibility: The software should be flexible to any changes made to it
  - Extensibility: There should not be any problem with the software on increasing the number of functions performed by it
  - Testability: It should be easy to test the software
  - Modularity: A software is of high modularity if it can be divided into separate independent parts and can be modified and tested separately
  - Scalability: The software should be easy to upgrade

DDoS → Distributed Denial of Service Attack



downside

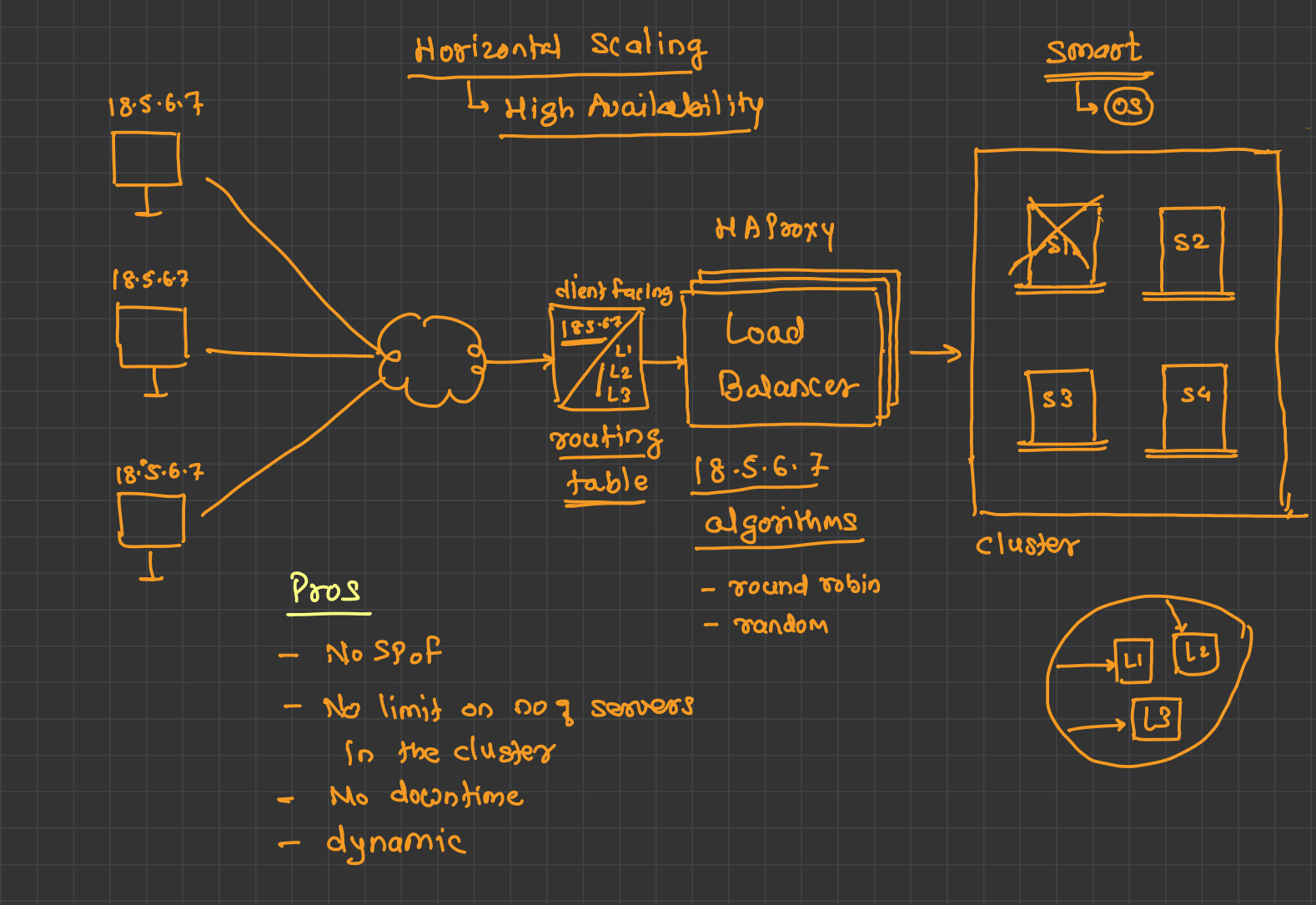
- Single dependency
- Single Point of failure (SPOF)
- limitation on configuration upgrade
- downtime required

upgrading configuration ↑ server

Vertical scaling

Horizontal Scaling

↳ High Availability



Smart

↳ OS

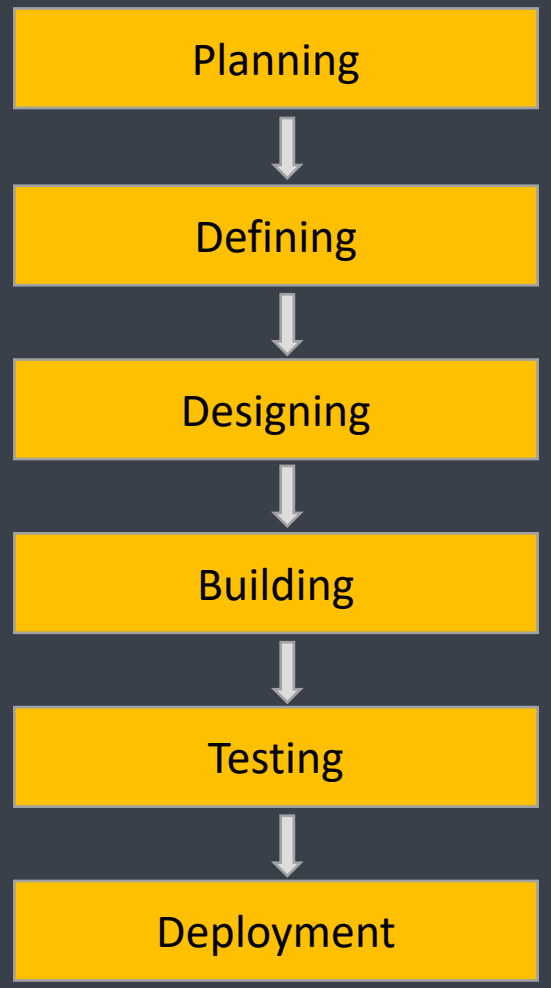
Pros

- No SPOF
- No limit on no of servers in the cluster
- No downtime
- dynamic

Software Development Life Cycle

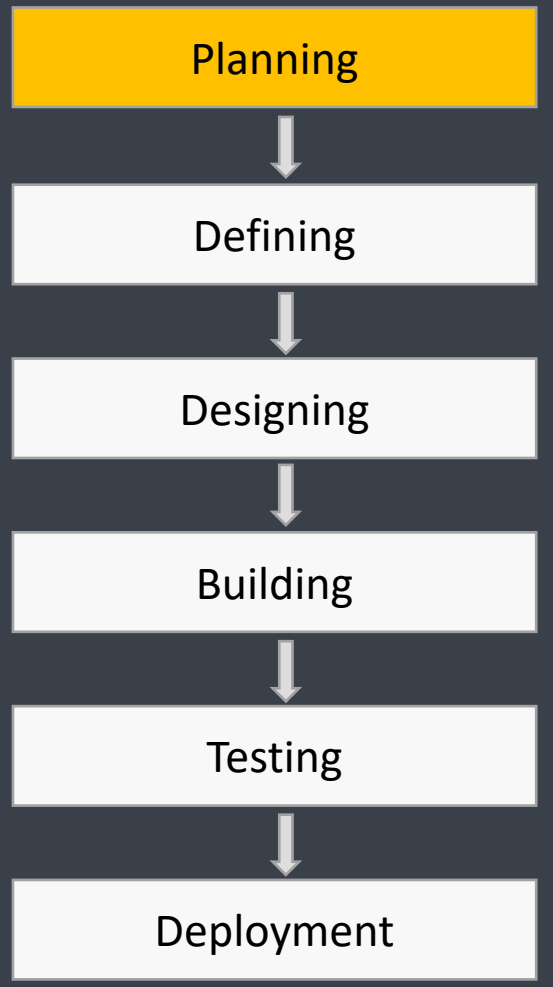
Overview

- Also called as Software Development Process
- Is a well-defined, structured sequence of stages in software engineering to develop the intended software product
- Is a framework defining tasks performed at each step in the software development process
- Aims to produce a high-quality software that
  - meets or exceeds customer expectations
  - reaches completion within times and cost estimates
- Consists of detailed plan (stages) of describing how to develop, test, deploy and maintain the software product



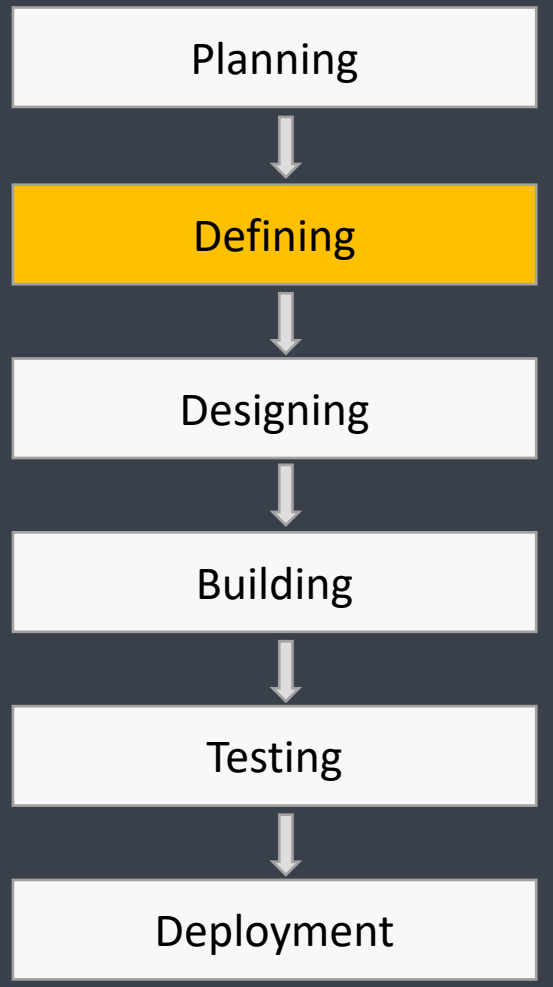
Planning and Requirement Analysis

- The most important and fundamental stage in SDLC
- It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry
- This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas
- Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage
- The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks



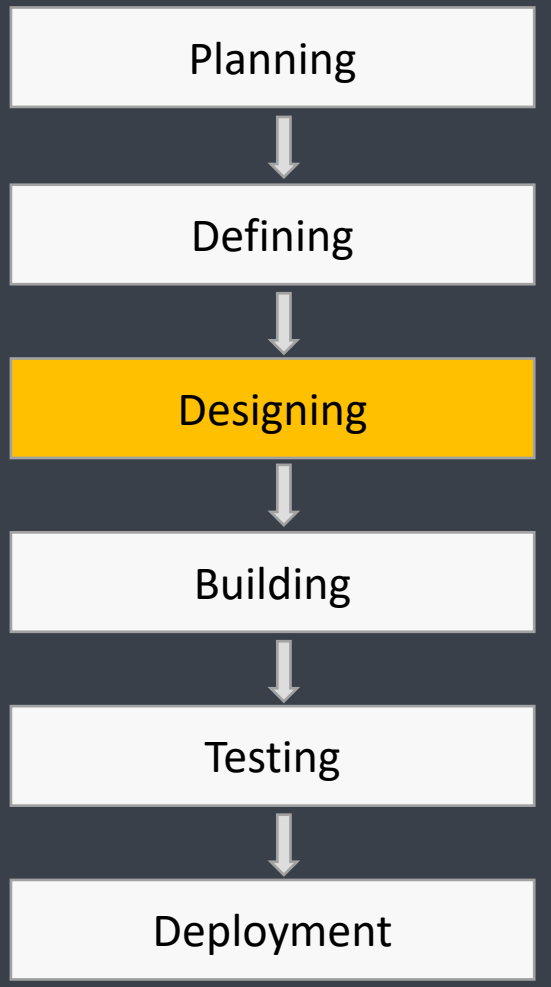
Defining Requirements

- Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts
- This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle



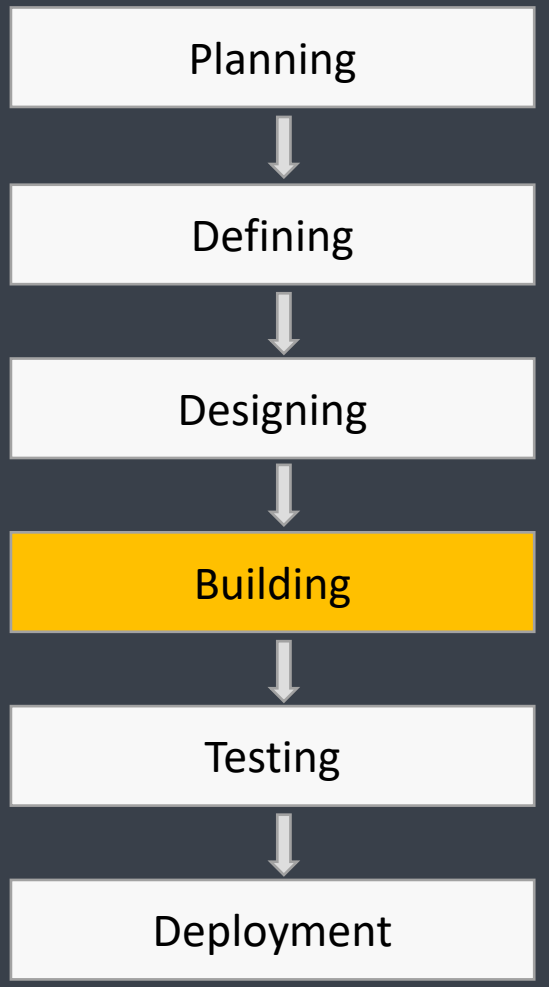
Designing the Product Architecture

- SRS is the reference for product architects to come out with the best architecture for the product to be developed
- Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification
- This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product
- A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any)
- The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS



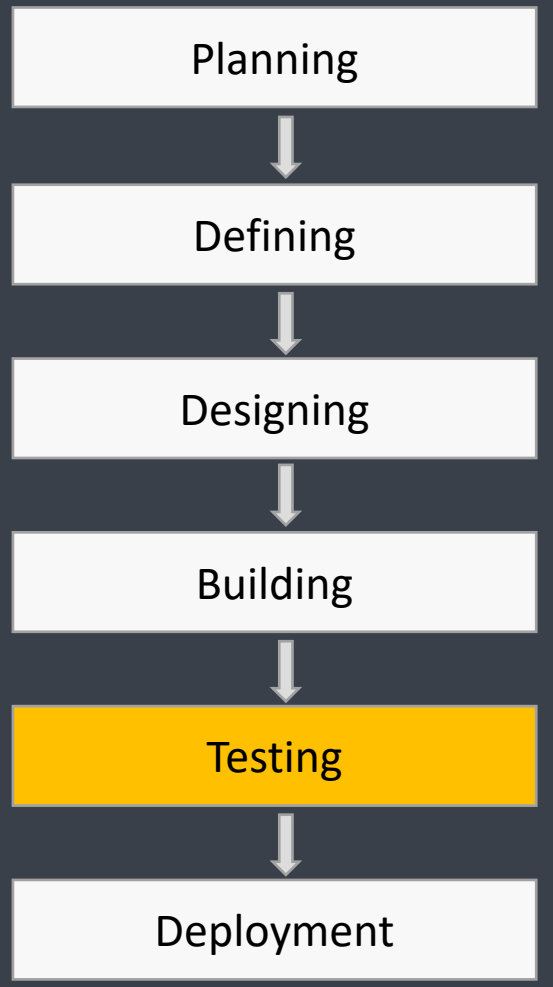
Building or Developing the Product

- In this stage of SDLC the actual development starts and the product is built
- The programming code is generated as per DDS during this stage
- If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle
- Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code
- Different high level programming languages such as C, C++, Java, PHP etc. are used for coding
- The programming language is chosen with respect to the type of software being developed



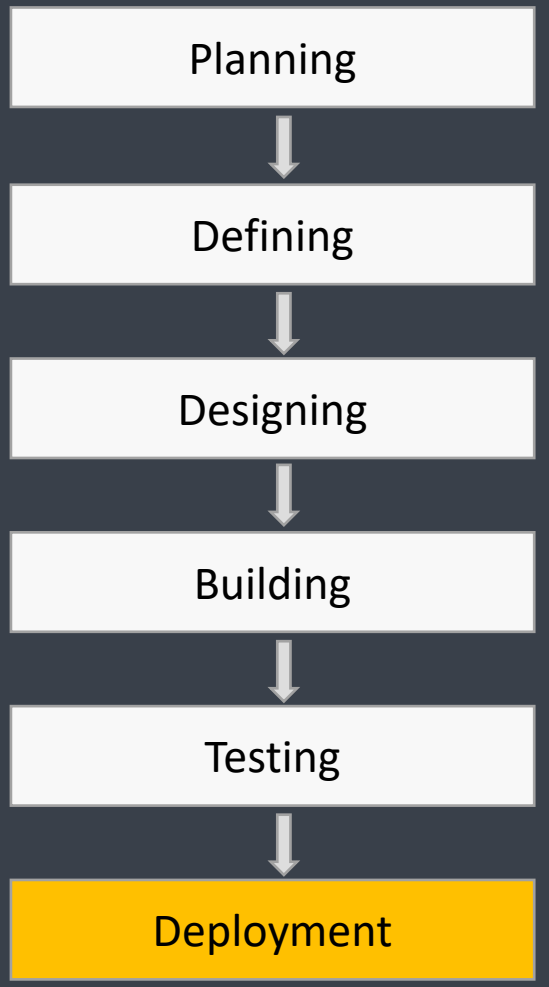
Testing the Product

- This stage is usually a subset of all the stages as in the modern SDLC models
- The testing activities are mostly involved in all the stages of SDLC
- However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS



Deployment in the Market and Maintenance

- Once the product is tested and ready to be deployed it is released formally in the appropriate market
- Sometimes product deployment happens in stages as per the business strategy of that organization
- The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing)
- Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment
- After the product is released in the market, its maintenance is done for the existing customer base



SDLC Models

- There are various software development life cycle models defined and designed which are followed during the software development process
- Also referred as Software Development Process Models
- Models
  - Waterfall Model
  - Iterative Model
  - Spiral Model
  - V-Model
  - Big Bang Model
  - Agile Model

Waterfall Model

- Requirement Specification
- System Design
- Design Implementation
- Verification & Testing
- System Deployment
- Software Maintenance

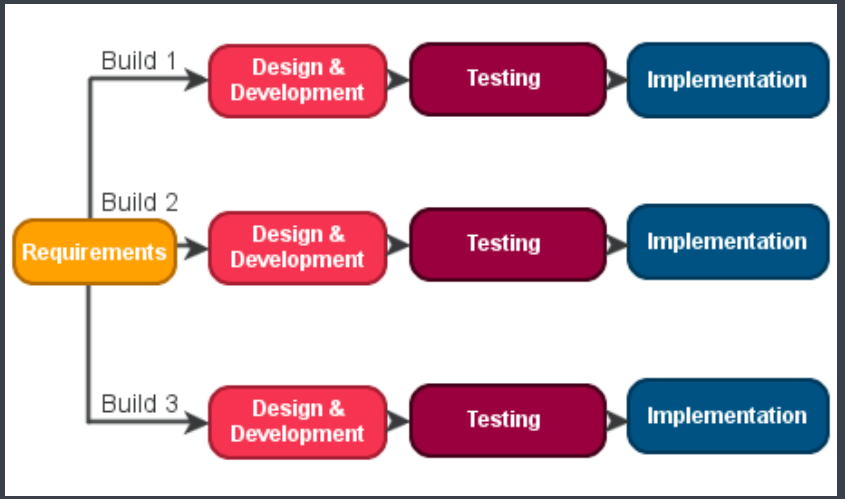
Waterfall Model – Procs and Cons

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model
- Each phase has specific deliverables and a review process
- Phases are processed and completed one at a time
- Works well for smaller projects where requirements are very well understood
- Clearly defined stages
- Well understood milestones
- Easy to arrange tasks
- Process and results are well documented

- No working software is produced until late during the life cycle
- High amounts of risk and uncertainty
- Not a good model for complex and object-oriented projects
- Poor model for long and ongoing projects
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model
- It is difficult to measure progress within stages
- Cannot accommodate changing requirements
- Adjusting scope during the life cycle can end a project
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early

Iterative Model

- In the Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed
- An iterative life cycle model does not attempt to start with a full specification of requirements
- Instead, development begins by specifying and implementing just part of the software, which is then reviewed to identify further requirements
- This process is then repeated, producing a new version of the software at the end of each iteration of the model



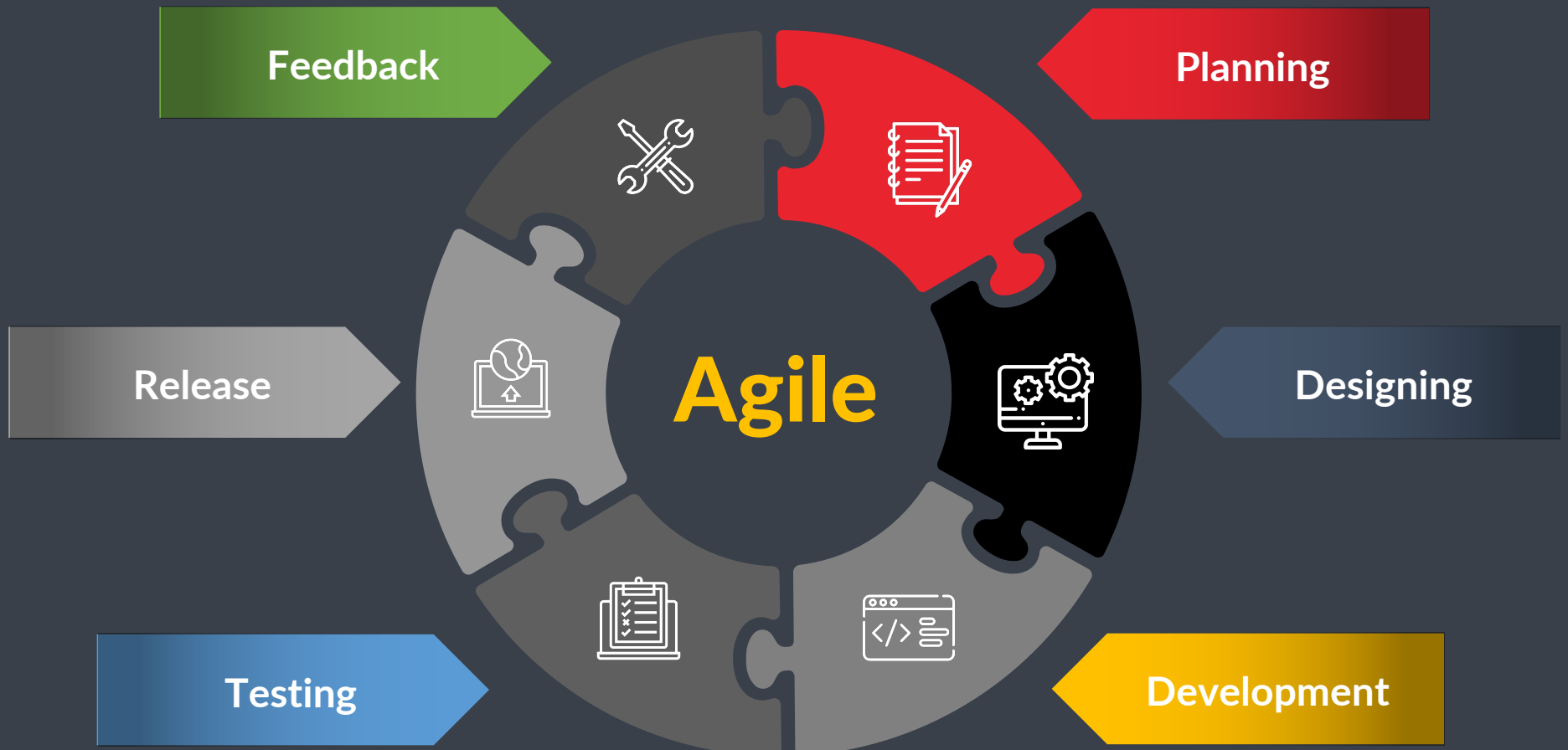
Iterative Model Pros and Cons

- Some working functionality can be developed quickly and early in the life cycle
- Results are obtained early and periodically
- Parallel development can be planned
- Progress can be measured
- Less costly to change the scope/requirements
- Testing and debugging during smaller iteration is easy
- Risks are identified and resolved during iteration; and each iteration is an easily managed milestone
- Easier to manage risk - High risk part is done first
- With every increment, operational product is delivered
- Issues, challenges and risks identified from each increment can be utilized/applied to the next increment
- Risk analysis is better
- It supports changing requirements
- Initial Operating time is less

- More resources may be required
- Although cost of change is lesser, but it is not very suitable for changing requirements
- More management attention is required
- System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle
- Defining increments may require definition of the complete system
- Not suitable for smaller projects
- Management complexity is more
- End of project may not be known which is a risk
- Highly skilled resources are required for risk analysis
- Projects progress is highly dependent upon the risk analysis phase

Agile Methodologies

Agile Development



Agile Manifesto

- In February 2001, at the Snowbird resort in Utah, 17 software developers met to discuss lightweight development methods
- The outcome of their meeting was the following Agile Manifesto for software development
  - We are uncovering better ways of developing software by doing it and helping others do it
  - Through this work, we have come to value
    - Individuals and interactions over Processes and tools
    - Working software over Comprehensive documentation
    - Customer collaboration over Contract negotiation
    - Responding to change over Following a plan
  - That is, while there is value in the items on the right, we value the items on the left more

Principles of Agile Manifesto

- Customer Satisfaction
  - Highest priority is given to satisfy the requirements of customers through early and continuous delivery of valuable software
- Welcome Change
  - Changes are inevitable during software development
  - Ever-changing requirements should be welcome, even late in the development phase.
  - Agile processes should work to increase customers' competitive advantage
- Deliver a Working Software
  - Deliver a working software frequently, ranging from a few weeks to a few months, considering shorter time-scale
- Collaboration
  - Business people and developers must work together during the entire life of a project
- Motivation
  - Projects should be built around motivated individuals
  - Provide an environment to support individual team members and trust them so as to make them feel responsible to get the job done
- Face-to-face Conversation
  - Face-to-face conversation is the most efficient and effective method of conveying information to and within a development team

Principles of Agile Manifesto

- Measure the Progress as per the Working Software
  - Working software is the key and it should be the primary measure of progress
- Maintain Constant Pace
  - Agile processes aim towards sustainable development
  - The business, the developers, and the users should be able to maintain a constant pace with the project
- Monitoring
  - Pay regular attention to technical excellence and good design to enhance agility
- Simplicity
  - Keep things simple and use simple terms to measure the work that is not completed
- Self-organized Teams
  - An agile team should be self-organized and should not depend heavily on other teams because the best architectures, requirements, and designs emerge from self-organized teams
- Review the Work Regularly
  - Review the work done at regular intervals so that the team can reflect on how to become more effective and adjust its behavior accordingly

Agile Methodologies

- The most popular Agile methods include
  - Rational Unified Process
  - Scrum
  - Crystal Clear
  - Extreme Programming
  - Adaptive Software Development
  - Feature Driven Development
  - Dynamic Systems Development Method (DSDM)

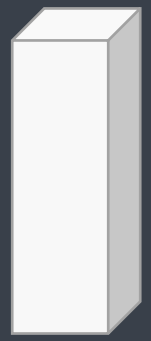
What is Scrum ?

- Scrum isn't a process, it's a framework that facilitates processes amongst other things
- Is an agile way to manage a project
- Management framework with far reaching abilities to control and manage the iterations and increments in all project types
- One of the implementations of agile methodology
- Incremental builds are delivered to the customer in every two to three weeks time
- Ideally used in the project where the requirement is rapidly changing
- The framework is made up of a Scrum team, individual roles, events, artefacts, and rules

Scrum Principles

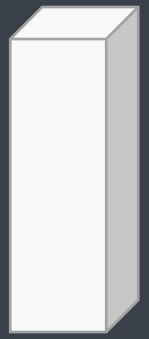
- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- Welcome changing requirements, even late in development
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- Business people and developers must work together daily throughout the project
- Build projects around motivated individuals
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation
- Working software is the primary measure of progress
- Agile processes promote sustainable development
- Continuous attention to technical excellence and good design enhances agility
- Simplicity, the art of maximizing the amount of work not done, is essential
- The best architectures, requirements, and designs emerge from self-organizing teams
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

Scrum Pillars



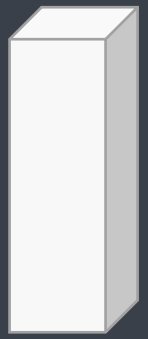
Transparency

All parts of the process should be transparent, open, and honest for everyone to see



Inspection

Everything that is worked on during a sprint can be inspected by the team to make sure that it is achieving what it needs to.



Adaptation

If a member of the Scrum team or a stakeholder notices that things aren't going according to plan, the team will need to change up what they're doing to fix this as quickly as possible

Scrum Values

Courage	Courage to do the right thing and work on tough problems
Focus	Focus on the sprint and its goal
Commitment	Commitment to the team and sprint goal
Respect	Respect, for each other by helping people to learn the things that you're good at, and not judging the things that others aren't good at
Openness	Be open and honest and let people know what you're struggling with challenges and problems that are stopping you from achieving success

How scrum pillars help us?

Self-organization

- This results in healthier shared ownership among the team members
- It is also an innovative and creative environment which is conducive to growth

Collaboration

- Essential principle which focuses collaborative work

Time-boxing

- Defines how time is a limiting constraint in Scrum method
- Daily Sprint planning and Review Meetings

Iterative Development

- Emphasizes how to manage changes better and build products which satisfy customer needs
- Defines the organization's responsibilities regarding iterative development

Scrum Roles

Product Owner

- Job to understand and engage with the stakeholders to understand what needs to be done and create that backlog
- Also need to prioritize that backlog

Scrum Master

- Helps the entire team achieve the scrum goals and work within scrum
- Support the product owner with their responsibilities in terms of managing the backlog as well as, supporting the development team

Dev Team

- The people who are creating the product or service and delivering done increments at the end of each sprint
- Includes developers, tester, writers, graphics artists and others

Scrum Artefacts

Product Backlog

Sprint Backlog

Increment

Scrum Events

Sprint Planning	• Happens at the start of every sprint • it should probably be about between four and eight hours
Daily Scrum	• This is a very short time-boxed event • Usually only lasting no more than 15 minutes
Sprint Review	• Collaborative events to demo what has been achieved and to help keep everyone who's involved working together
Sprint Retrospective	• About continuous process and improvement and we need to take what we've learned into the next sprint planning session
Sprint	• It is really the beating heart of scrum and all the scrum events take place in it

Agile vs traditional models

Agile Methodologies	Traditional Methodologies
Incremental value and risk management	Phased approach with an attempt to know everything at the start
Embracing change	Change prevention
Deliver early, fail early	Deliver at the end, fail at the end
Transparency	Detailed planning, stagnant control
Inspect and adapt	Meta solutions, tightly controlled procedures and final answers
Self managed	Command and control
Continual learning	Learning is secondary to the pressure of delivery

Agile - Advantages

- Very realistic approach to software development
- Promotes teamwork and cross training
- Functionality can be developed rapidly and demonstrated
- Resource requirements are minimum
- Suitable for fixed or changing requirements
- Delivers early partial working solutions
- Good model for environments that change steadily
- Minimal rules, documentation easily employed
- Enables concurrent development and delivery within an overall planned context
- Little or no planning required
- Easy to manage
- Gives flexibility to developers

Agile - Disadvantages

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is a very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.

Scrum tools

- Jira – https://www.atlassian.com/software/jira/
- Clarizen – https://www.clarizen.com/
- GitScrum – https://site.gitscrum.com/
- Vivify Scrum – https://www.vivifyscrum.com/
- Yodiz – https://www.yodiz.com/
- ScrumDo – https://www.scrumdo.com/
- Quickscrum – https://www.quickscrum.com/
- Manuscript – https://www.manuscript.com/
- Scrumwise – https://www.scrumwise.com/
- Axosoft – https://www.axosoft.com/

Agile Methodologies – Scrum Terminologies

- **Scrum**
  - A framework to support teams in complex product development
- **Scrum Board**
  - A physical board to visualize information for and by the Scrum Team, used to manage Sprint Backlog
- **Scrum Master**
  - The role within a Scrum Team accountable for guiding, coaching, teaching and assisting a Scrum Team and its environments in a proper understanding and use of Scrum
- **Scrum Team**
  - A self-organizing team consisting of a Product Owner, Development Team and Scrum Master
- **Self-organization**
  - The management principle that teams autonomously organize their work
- **Sprint**
  - Time-boxed event of 30 days, or less, that serves as a container for the other Scrum events and activities.
- **Sprint Backlog**
  - An overview of the development work to realize a Sprint's goal, typically a forecast of functionality and the work needed to deliver that functionality

Agile Methodologies – Scrum Terminologies

- **Sprint Goal**
  - A short expression of the purpose of a Sprint, often a business problem that is addressed
- **Sprint Retrospective**
  - Time-boxed event of 3 hours, or less, to end a Sprint to inspect the past Sprint and plan for improvements
- **Sprint Review**
  - Time-boxed event of 4 hours, or less, to conclude the development work of a Sprint
- **Stakeholder**
  - A person external to the Scrum Team with a specific interest in and knowledge of a product that is required for incremental discovery
- **Development Team**
  - The role within a Scrum Team accountable for managing, organizing and doing all development work
- **Daily Scrum**
  - Daily time-boxed event of 15 minutes for the Development Team to re-plan the next day of development work during a Sprint

Scrum Workflow