Software Engineering

Aniket Mishra

T.E. Semester –VI
Choice Based Credit Grading Scheme with Holistic Student Development (CBCGS- H 2019)
TCET Autonomy Scheme (w.e.f. A.Y. 2020-21)

BE (Computer Engineering)							SEM : V		
Course Name : Software Engineering							Course Code : PCC-CS603		
Teaching Scheme (Program Specific)					Examination scheme				
Modes of Teaching / Learning / Weightage					Modes of Continuous Assessment / Evaluation				
Hours Per Week-					Theory (100)		Practical/ Oral (25)	Term-work	Total
Theory	Tu.	Practical	Contact Hours	Credits	IA	ESE	PR/OR	TW	
3	-	2	5	4	25	75	25	25	150

IA: In-Semester Assessment - Paper Duration – 1.5 Hours

ESE: End Semester Examination - Paper Duration - 3 Hours

The weightage of marks for continuous evaluation of Term work/Report: Formative (40%), Timely completion of practical (40%) and Attendance / Learning Attitude (20%)

Prerequisite: Object Oriented Programming, Frontend Backend connectivity

Course Objective: The objective of the course is to introduce to the students about the development of software product, the processes that provides a framework for the engineering methodologies and practices. Also to give the information regarding the phases including the analysis, design, testing methodologies and quality assurance.

Course Outcomes: Students will be able to:

SN	Course Outcomes	RBT level
1	Understand the use of basic and advanced models in software engineering	L1, L2
2	Analyze the scenarios to design the UML diagrams	L1, L2, L3, L4
3	Understand and apply the different techniques of project estimation an understand the tracking methods	L1, L2, L3, L4
4	Understand the design concepts and apply them to the project	L1, L2, L3, L4
5	Identify risks, manage the change to assure quality in software project.	L1, L2, L3, L4
6	Apply the principles of testing and develop test plan for the project	L1, L2, L3, L4

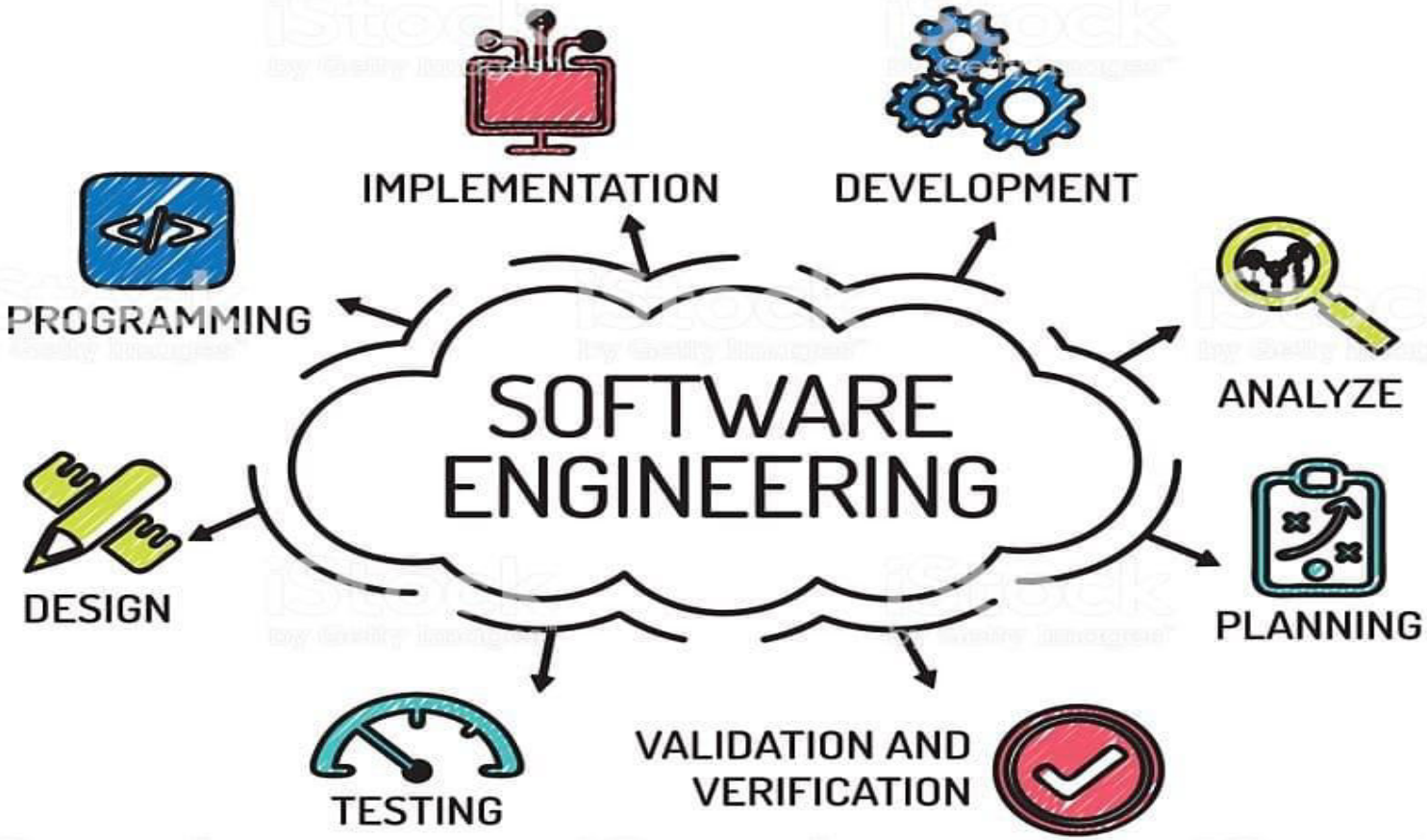
TCET
DEPARTMENT OF COMPUTER ENGINEERING (COMP)
(Accredited by NBA for 3 years, 3rd Cycle Accreditation w.e.f. 1st July 2019)
Choice Based Credit Grading Scheme (CBCGS)
Under TCET Autonomy

Detailed Syllabus:

Module No.	Topics	Hrs.	RBT Levels
1.0	**Introduction** Introduction to software engineering, Importance of Software engineering Software Process, Various models for Software Development (Waterfall, Spiral, Agile (Scrum), V-Model, RAD, DevOps), Capability Maturity Model (CMM).	6	L1, L2, L3
2.0	**Requirements Analysis and Modelling** Requirement Elicitation, Software requirement specification (SRS),Data Flow Diagram(DFD), Feasibility Analysis, Cost- Benefit Analysis, Developing Use Cases (UML), Requirement Model – Scenario-based model, Class-based model, Behavioral model.	8	L1, L2, L3, L4
3.0	**Project Scheduling and Tracking** Software Project Estimation: LOC, FP, Empirical Estimation Models - COCOMO II Model, Estimation for agile: planning poker, user story planning Project scheduling: Timeline charts, CPM Fishbone diagram	4	L1, L2, L3, L4
4.0	**Software Design** Design Concepts, Characteristics of Good Design, Effective Modular Design – Cohesion and Coupling. Architectural Styles, UI Design	8	L1, L2, L3
5.0	**Software Risk, Configuration Management & Quality Assurance** Risk Identification, Risk Assessment, Risk Projection, RMMM, Software Configuration management , Software Quality Assurance: Software Reliability, Formal Technical Review (FTR), Walkthrough, Quality Assurance Standards	8	L1, L2, L3, L4
6.0	**Software Testing and Maintenance** Software Testing, Unit testing, Integration testing Verification, Validation Testing, System Testing, Test plan, White-Box Testing , Basis Path Testing, Control Structure Testing, Black-Box Testing, Software maintenance and its types, Software Re-engineering, Reverse Engineering	11	L1, L2, L3, L4
	Total Hours	45	

Software Engineering ...?

- Software engineering is defined as a process of analyzing user requirements and then designing, building, and testing software application which will satisfy those requirements.
- Software Engineering is a collection of techniques, methodologies and tools that help with the production of
- a high quality software system
- with a given budget
- before a given deadline
- while change occurs.


IMPLEMENTATION
DEVELOPMENT
PROGRAMMING
ANALYZE
SOFTWARE ENGINEERING
DESIGN
PLANNING
TESTING
VALIDATION AND VERIFICATION

Requirement of software engineering as a subject:

- Students are proficient in programming language , however they have lack of experience in analysis and design of a system.
- To learn technical aspects of analysis and design of complex systems.

Objective :

- Develop complex system in context of frequent change.
- Produce high quality software with in time limits.
- Develop technical knowledge
- Develop managerial knowledge.
- Many software became over budget.
- Larger software was difficult and quite expensive to maintain.
- Demand for new software increased faster compared with the ability to generate new software.

Importance :

- Reduces complexity
- Reduces the software cost
- Reduces time
- Handling big projects
- Reliable software
- Effectiveness

- **Reduces complexity :** Major projects are difficult and complex to develop. This project can be subdivided into smaller parts and each of those parts can be worked upon independently.
- **Reduces the software cost:** Utilizing the method of software engineering, the development process can be reduced because of a planned approach by the programmer which helps the programmer to remove the processes that are not necessary. therefore the cost of the software production and the development of the product is highly reduced.

- **Reducing time :** Software engineering is the process of making a perfect production plan for the development of software.
- Creating software according to the software engineering approach then it will decrease the amount of time taken in the completion of the software.
- **Handling projects :** The process has to maintain the stages of the development so after every single month it can check its development and track its progress as the process provides many resources to the project and therefore it requires the project to be completed in time. Therefore, in this case, the software engineering approach is very beneficial to handle these break projects without any problem.

- **Reliable software :** There should be no bugs and errors and also it should be reliable therefore the software that the developer has developed should work at least for a given span of time.

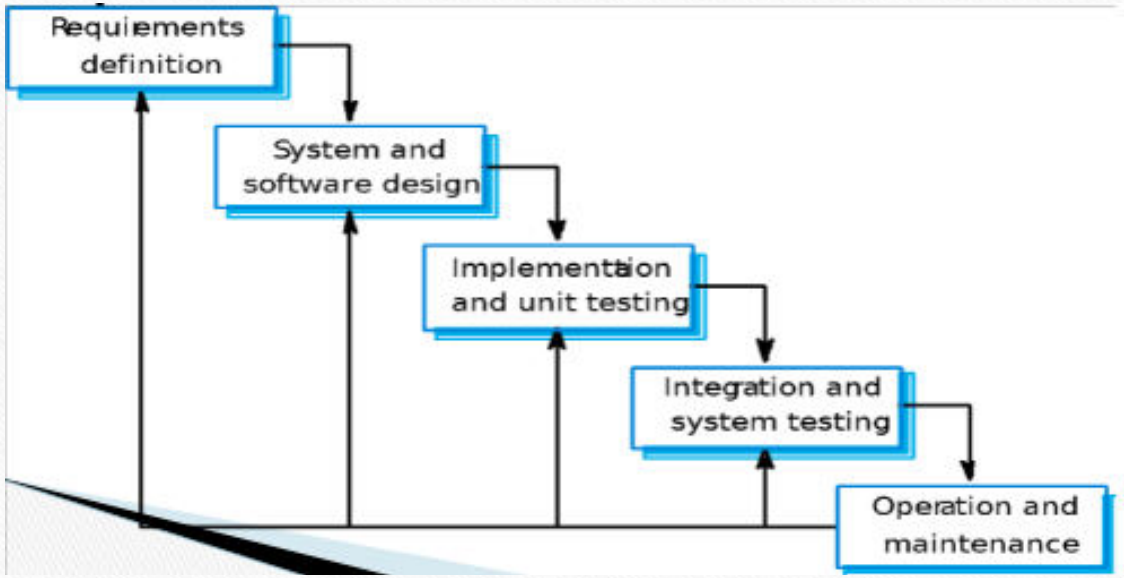
- **Effectiveness :** The software that is developed by the company is only effective if it has followed the standards that the company provides. These standards are the main focus of the company and due to the software standards, the software has become more effective.

Software life cycle models :

- The following are the software life cycles models,
- Waterfall
- RAD
- Spiral
- Open source
- Agile process

Waterfall model :

- This model is also known as the linear sequential model or the software life cycle.



Why Waterfall model ...?

- **Requirements are complex**
- The client does not know the functional requirements in advance
- **Requirements may be changing**
- Technology enablers introduce new possibilities to deal with nonfunctional requirements
- **Frequent changes are difficult to manage**
- Identifying milestones and cost estimation is difficult
- **There is more than one software system**
- New system must be backward compatible with existing system (“legacy system”)
- Phased development: Need to distinguish between the system under development and already released

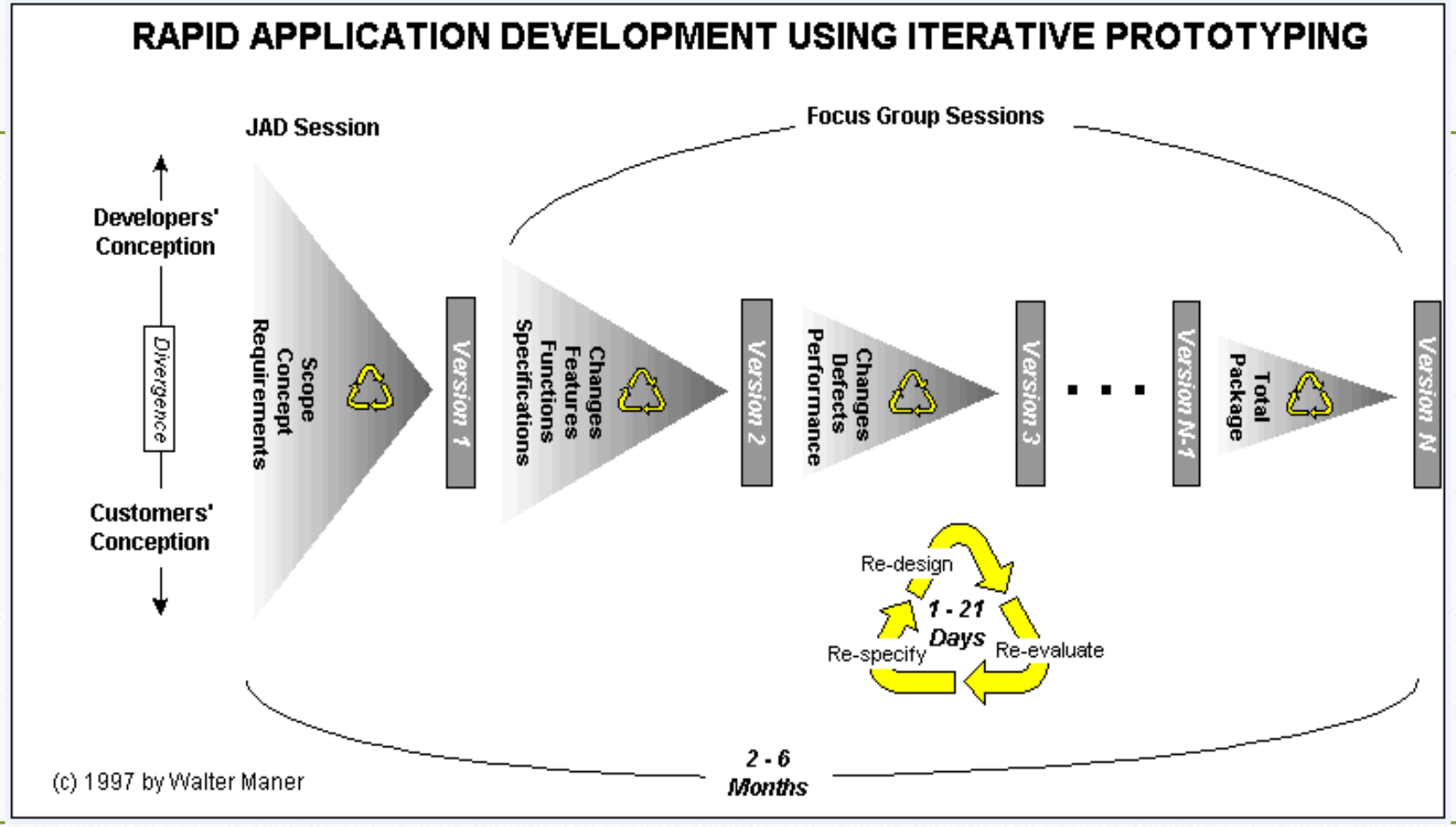
Advantages :

- Testing is inherent to every phase of the waterfall model
- It is an enforced disciplined approach
- It is documentation driven, that is, documentation is produced at every stage.

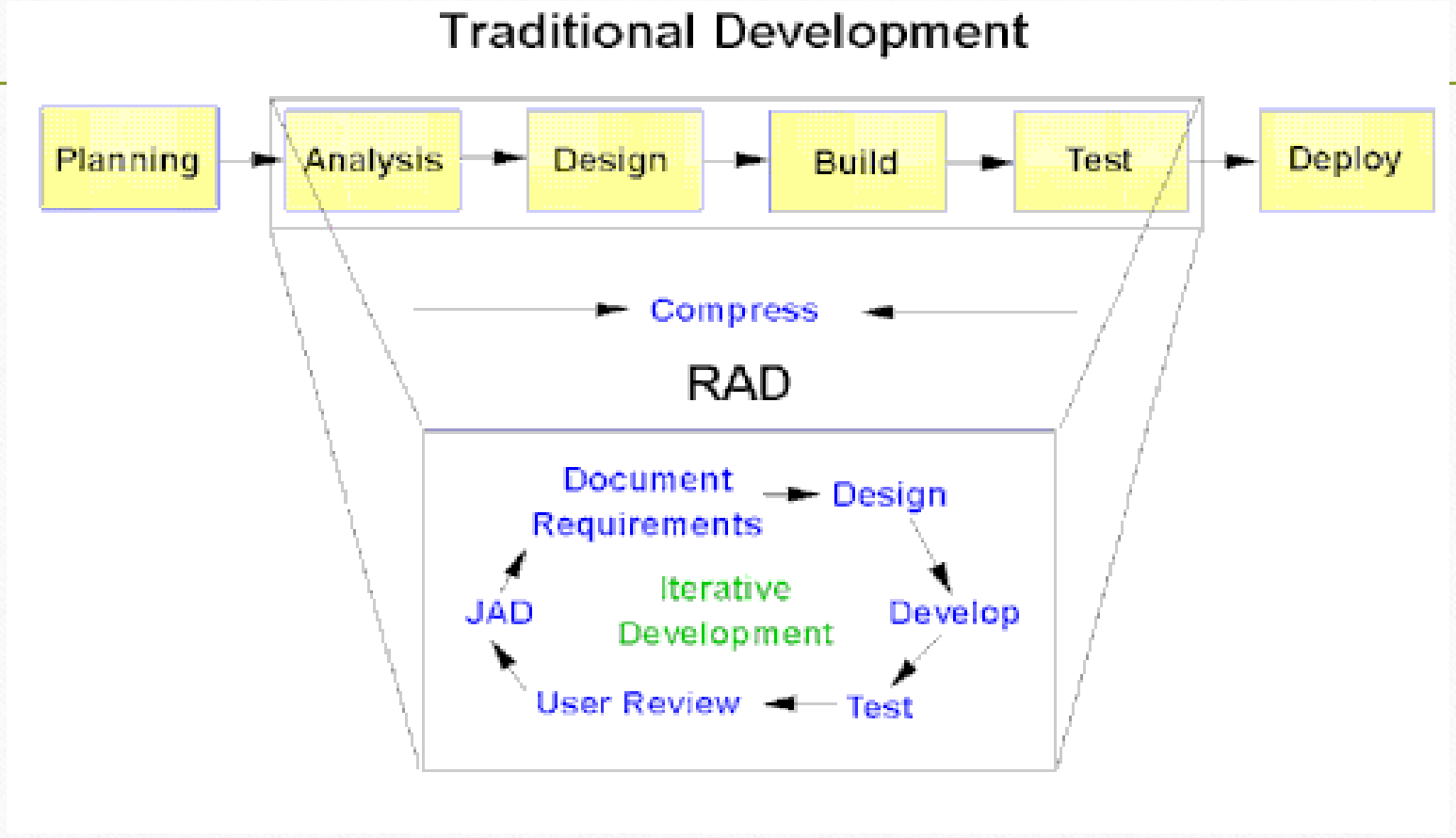
Disadvantages :

- Projects rarely flow its sequential flow as we cannot go back to introduce change.
- Small changes or errors that arise in the completed software may cause a lot of problem.
- it happens that the client is not very clear of what he exactly wants from the software. Any changes that he mentions in between may cause a lot of confusion.
- The greatest disadvantage of the waterfall model is that until the final stage of the development cycle is complete, a working model of the software does not lie in the hands of the client

RAD Model


RAPID APPLICATION DEVELOPMENT USING ITERATIVE PROTOTYPING
JAD Session
Focus Group Sessions
Developers' Conception
Divergence
Customers' Conception
Scope Concept Requirements
Version 1
Changes Features Functions Specifications
Version 2
Changes Defects Performance
Version 3
Version N-1
Total Package
Version N
Re-design
1 - 21 Days
Re-specify
Re-evaluate
2 - 6 Months
(c) 1997 by Walter Maner

Traditional VS RAD



RAD Definition

- Rapid Application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle (anywhere from 60-90 days). The RAD model is a high-speed adaptation of the linear sequential model / Waterfall model. The RAD approach encompasses the following phases:

RAD approach encompasses the following phases:

1. Business Modelling

What information drives the business processes?

What information is generated?

Who generates it?

Where does the information flow?

Who processes it?

RAD approach encompasses the following phases: contd..

2.Data Modelling

The information flow defined as part of the business modeling phase is refined into **a set of data objects** that are needed to support the business.

3. Process modeling

Processing descriptions are created for **adding, modifying, deleting or retrieving** a data object.

RAD approach encompasses the following phases: contd..

4.Application generation

RAD assumes the use of fourth generation techniques. Rather than creating software using conventional third generation programming languages,

RAD process **works to use the automated tools** to facilitate the construction of the software.

4. Testing and turnover

This saves time, money and the overall time to test an application also reduces considerably.

Advantages for using RAD

- To converge early toward a design acceptable to the customer and feasible for the developers
- To limit a project's exposure to the forces of change
- To save development time, possibly at the expense of economy or product quality

Disadvantages for using RAD

- To prevent cost overruns
  (RAD needs a **team already disciplined in cost management**)
- To prevent runaway schedules
  (RAD needs a team already disciplined **in time management**

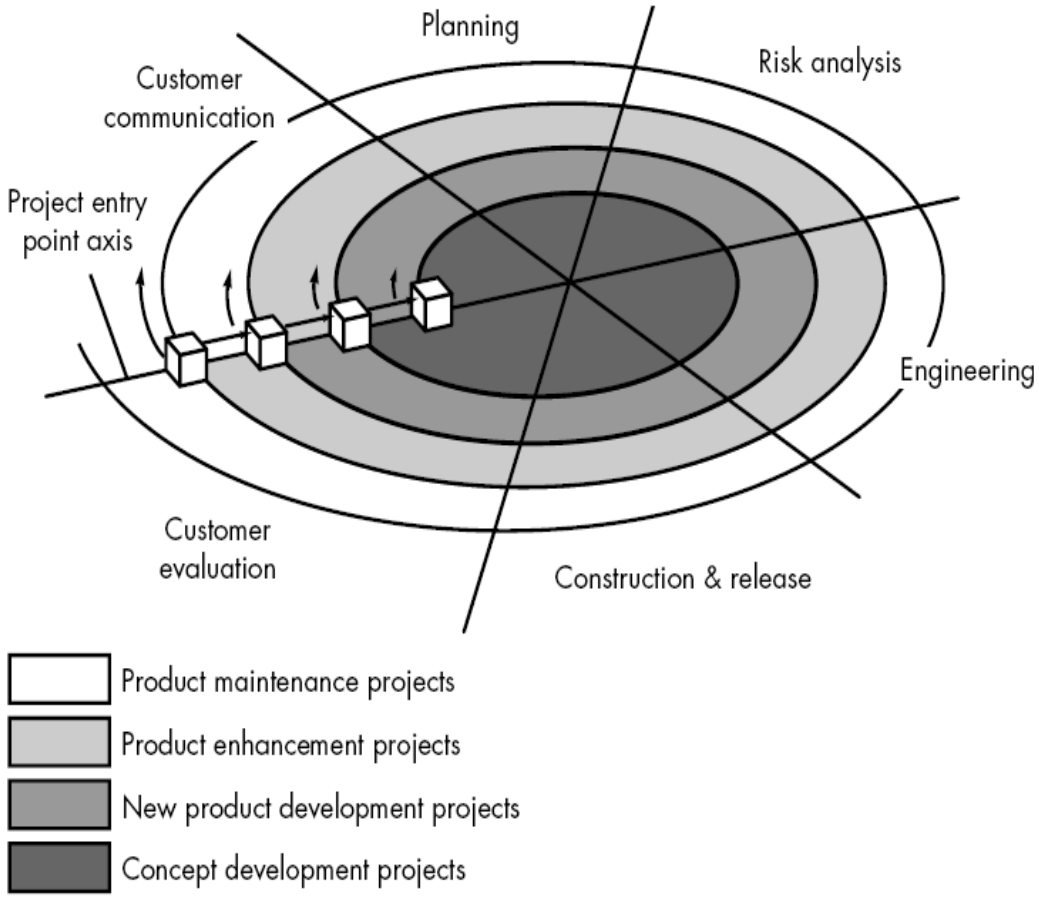
Evolutionary models

- Useful in following situations
- When Business and product requirements often change as development proceeds,
- tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure;
- Software engineers need a model which will be able to accommodate change.

Spiral Model

- The *spiral model, originally proposed by Boehm , is an evolutionary software* process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model.
- It provides the potential for rapid development of incremental versions of the software.
- Using the spiral model, software is developed in a series of incremental releases.
- During early iterations, the incremental release might be a paper model or prototype.
- Each loop in the spiral represents a phase in the process.
- Risks are explicitly assessed and resolved throughout the process.

FIGURE 2.8
A typical spiral model



Spiral model sectors

- A spiral model is divided into a number of framework activities, also called *task Regions*.
- **Customer communication—tasks required to establish effective communication** between developer and customer.
- **Planning—tasks required to define resources, timelines, and other project related** information.
- **Risk analysis—tasks required to assess both technical and management** risks.
- **Engineering—tasks required to build one or more representations of the** application.
- **Construction and release—tasks required to construct, test, install, and** provide user support (e.g., documentation and training).

Advantages of Spiral model

- It promotes reuse of existing software in early stages of development.
- Allows quality objectives to be formulated during development.
- Provides preparation for eventual evolution of the software product.
- Eliminates errors and unattractive alternatives early.

Disadvantages of Spiral Model

- **Complex:** The Spiral Model is much more complex than other SDLC models.
- **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
- **Too much dependability on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.
- **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.

OPEN-SOURCE Software

Generically, *open source* refers to a program in which the source code is available to the general public for use and/or modification from its original design free of charge, i.e., open. Open source code is typically created as a collaborative effort in which programmers improve upon the code and share the changes within the community. Open source sprouted in the technological community as a response to proprietary software owned by corporations

Open Source Initiative (OSI)

OSI dictates that in order to be considered "OSI Certified" a product must meet the following criteria:

- The author or holder of the license of the source code cannot collect royalties on the distribution of the program
- The distributed program must make the source code accessible to the user
- The author must allow modifications and derivations of the work under the program's original name
- No person, group or field of endeavor can be denied access to the program
- The rights attached to the program must not depend on the program's being part of a particular software distribution
- The licensed software cannot place restrictions on other software that is distributed with

Example:

- Linux/Open-Source offers more than cost savings
  - Freedom, Flexibility; Platform to build a business on
  - Set of commodity software components that can be put together for different purposes
- Open-Source Software is more than Linux
  - Sendmail, Apache, MySQL, PostgreSQL, LTSP
  - JBoss, SQL-Ledger, OpenCRM, SpamAssassin
  - Evolution, OpenOffice, Mozilla, KDE/Gnome, GAIM

Advantages are:

1.Mostly software free or less cost.
2.right to use the software in any way.
3.no single entity on which the future of the software depends.
4.No per-copy fees can be asked for modified versions.
5.availability of the source code and the right to modify it.

Disadvantages are:

1.Mostly used commercial applications.
2.security bugs
3.Support issues

Agile Process

- Agility
  - The ability to both create and respond to change in order to profit in a turbulent business environment
    - Companies need to determine the amount of agility so they need to be competitive
- Why do we go for Agile process

1) the customer did not give all of the expected requirements and

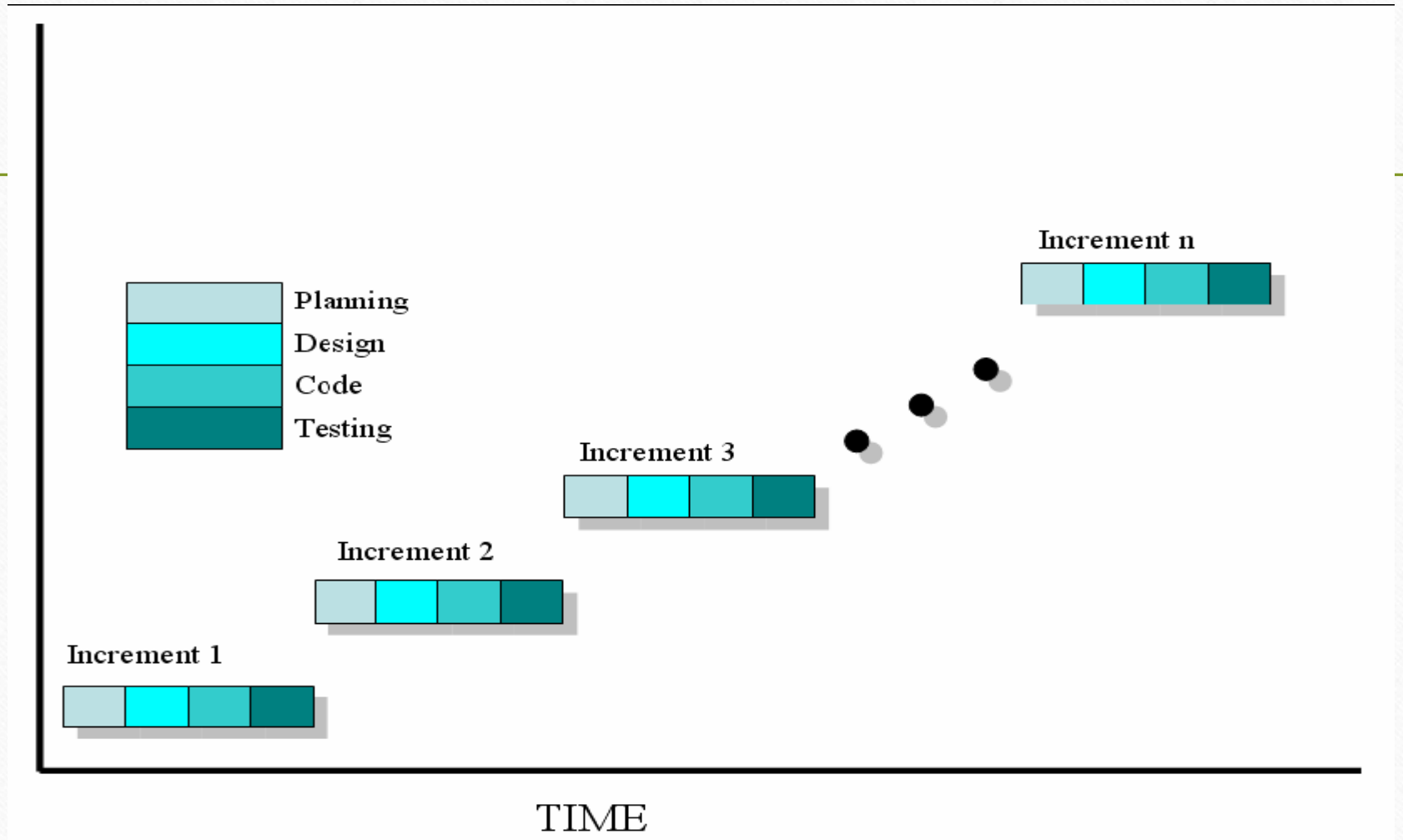
2) engineers did not always understand the requirements.

Some Agile Methodologies

- Extreme Programming (XP)-With XP

programming the steps are only planning, design, coding and testing.

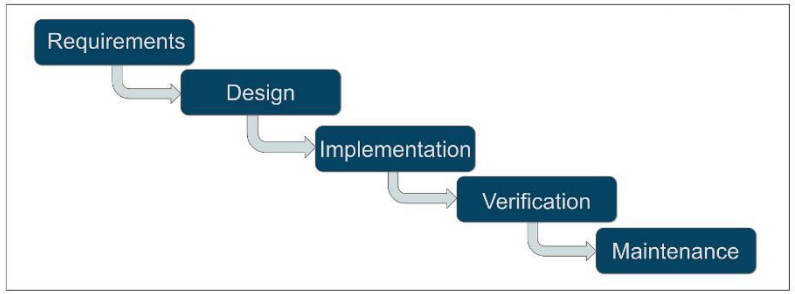
e.g. pair programming


Planning
Design
Code
Testing
Increment 1
Increment 2
Increment 3
Increment n
TIME

- Scrum-it is iterative in nature
- Sprints- basic unit of development in scrum. It produces a working and tested software within given time limits.
- Sprints tend to last between on week to month

DevOps Life cycle

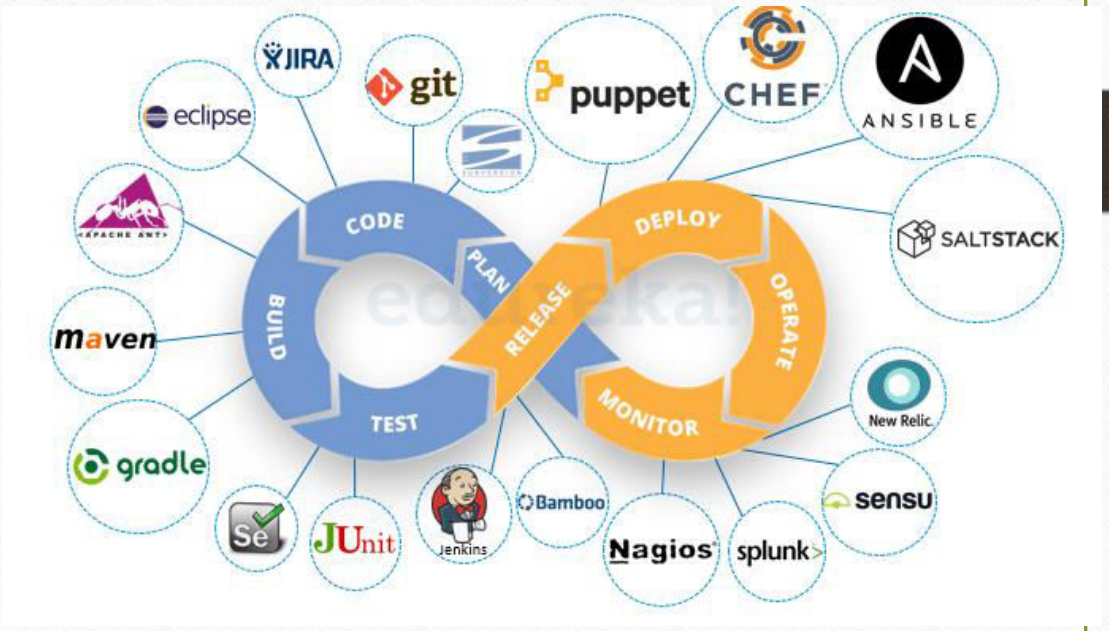
- **Why DevOps?**
- Before we know what DevOps is, it is very important to know how DevOps came into existence. Before DevOps, we had the waterfall model and the agile model for software development. So let us have a look at the waterfall model.
  - **Waterfall Model**



- **AGILE Methodology**

What is DevOps?

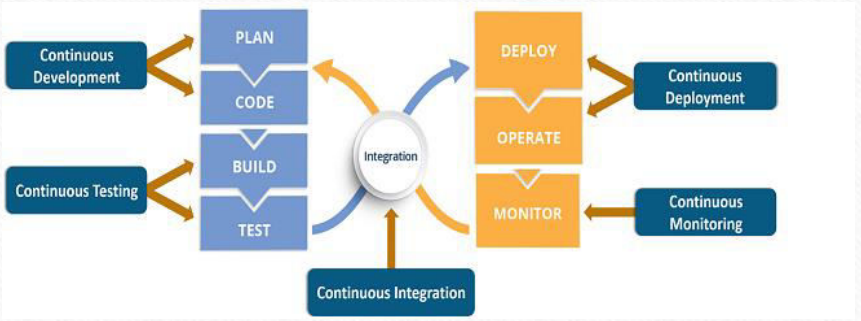
- The term DevOps is a combination of two words namely Development and Operations. DevOps is a practice that allows a single team to manage the entire application development life cycle, that is, development, testing, deployment, operations.
- The aim of DevOps is to shorten the system's development life cycle while delivering features, fixes, and updates frequently in close alignment with business objectives.
- DevOps is a software development approach through which superior quality software can be developed quickly and with more reliability. It consists of various stages such as continuous development, continuous integration, continuous testing, continuous deployment, and continuous monitoring.



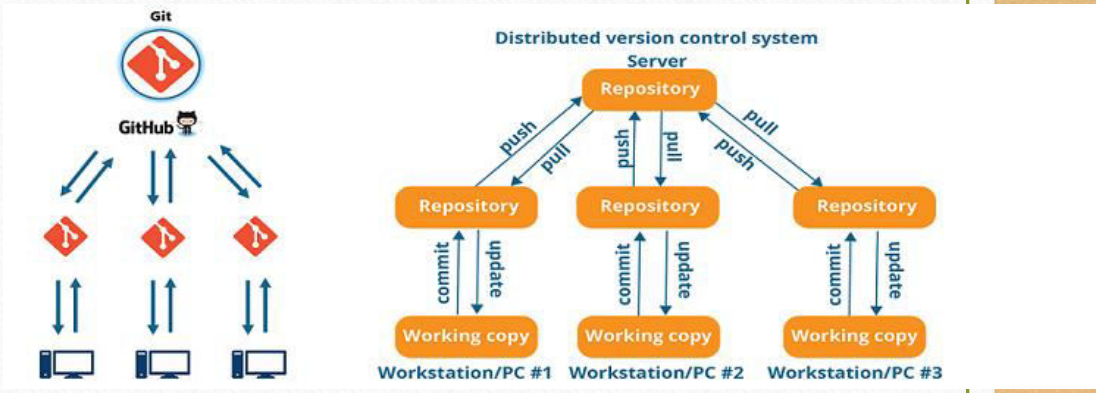
What is DevOps Life cycle?

- the various phases such as continuous development, continuous integration, continuous testing, continuous deployment, and continuous monitoring constitute DevOps Life cycle. Now let us have a look at each of the phases of DevOps life cycle one by one.

- **Continuous Development –**
- This is the phase that involves ‘planning‘ and ‘coding‘ of the software. The vision of the project is decided during the planning phase and the developers begin developing the code for the application. There are no DevOps tools that are required for planning, but there are a number of tools for maintaining the code.
- The code can be written in any language, but it is maintained by using Version Control tools. Maintaining the code is referred to as Source Code Management. The most popular tools used are Git, SVN, Mercurial, CVS, and JIRA. Also tools like Ant, Maven, Gradle can be used in this phase for building/ packaging the code into an executable file that can be forwarded to any of the next phases.



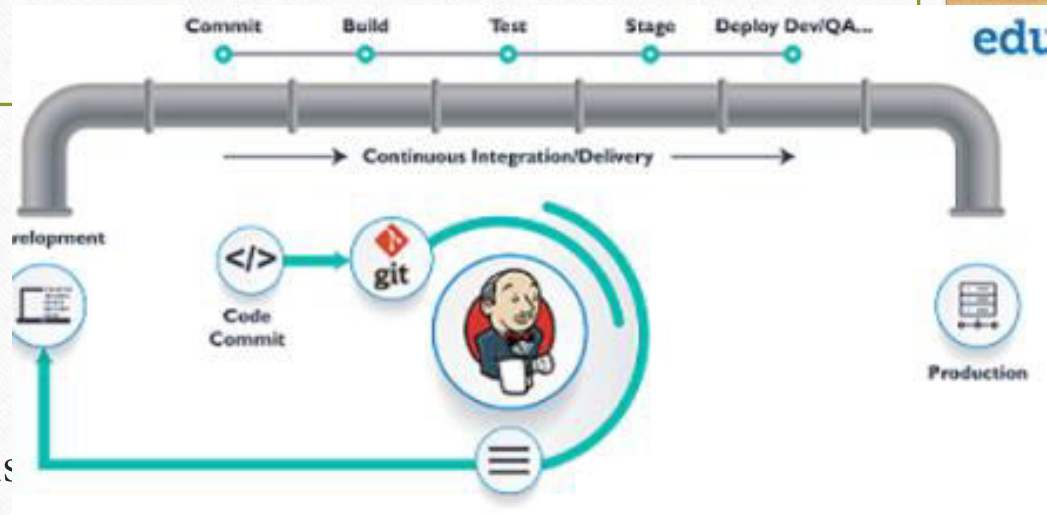
- Now let us try to know a bit more about Git.
- Git is a distributed version control tool that supports distributed non-linear workflows by providing data assurance for developing quality software. Tools like Git enable communication between the development and the operations team.
- When you are developing a large project with a huge number of collaborators, it is very important to have communication between the collaborators while making changes in the project.
- Commit messages in Git play a very important role in communicating among the team. Apart from communication, the most important reason to use Git is that you always have a stable version of the code with you.
- Hence, Git plays a vital role in succeeding at DevOps.



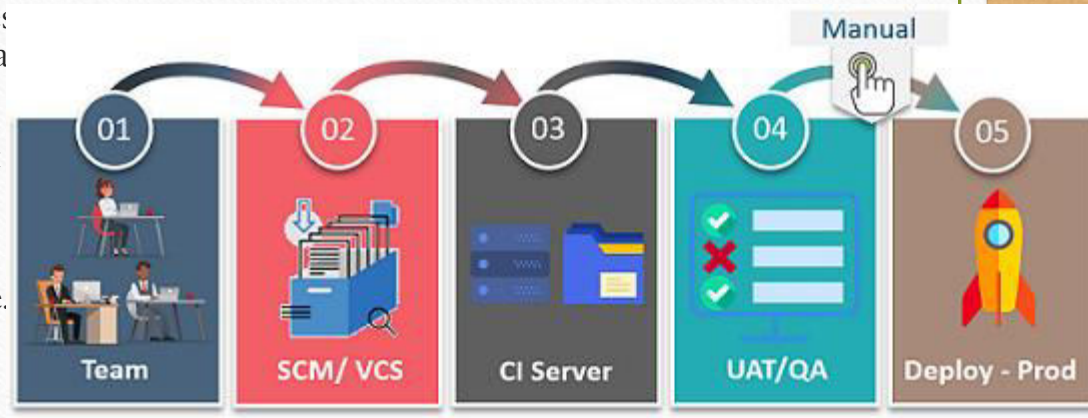
- **Continuous Testing –**
- This is the stage where the developed software is continuously tested for bugs. For Continuous testing, automation testing tools like Selenium, TestNG, JUnit, etc are used. These tools allow QAs to test multiple code-bases thoroughly in parallel to ensure that there are no flaws in the functionality. In this phase, Docker Containers can be used for simulating the test environment.
- Selenium does the automation testing, and the reports are generated by TestNG. This entire testing phase can be automated with the help of a Continuous Integration tool called Jenkins. Suppose you have written a selenium code in Java to test your application. Now you can build this code using ant or maven. Once the code is built, it is tested for User Acceptance Testing (UAT). This entire process can be automated using Jenkins.

- Automation testing saves a lot of time, effort and labor for executing the tests instead of doing this manually. Besides that, report generation is a big plus. The task of evaluating the test cases that failed in a test suite gets simpler. We can also schedule the execution of the test cases at predefined times. After testing, the code is continuously integrated with the existing code.

- **Continuous Integration –**
- This stage is the heart of the entire DevOps life cycle. It is a software development practice in which the developers require to commit changes to the source code more frequently. This may be on a daily or a weekly basis. Every commit is then built and this allows early detection of problems if they are present. Building code not only involves compilation but it also includes code review, unit testing, integration testing, and packaging.
- The code supporting new functionality is continuously integrated with the existing code. Since there is continuous development of software, the updated code needs to be integrated continuously as well as smoothly with the systems to reflect changes to the end-users.
- Jenkins is a very popular tool used in this phase. Whenever there is a change in the Git repository, Jenkins fetches the updated code and it prepares a build of that code which is an executable file in the form of a war or a jar. This build is then forwarded to the test server or the production server.



- **Continuous Deployment –**
- This is the stage where the code is deployed to the production servers. It is also important to ensure that the code is correctly deployed on all the servers. Before moving on, let us try to understand a few things about Configuration management and Containerization tools. These set of tools here help in achieving Continuous Deployment (CD).
- Configuration Management is the act of establishing and maintaining consistency in an application's functional requirements and performance. Let us put this in simpler words, it is the act of releasing deployments to servers, scheduling updates on all servers and most importantly keeping the configurations consistent across a the servers.
- Since the new code is deployed on a continuous basis, configuration management tools play an important role in executing tasks quickly and frequently. Some popular tools that are used here are Puppet, Chef, SaltStack, and Ansible.
- Containerization tools also play an equally important role in the deployment stage. Docker and Vagrant are the popular tools used for this purpose. These tools help produce consistency across Development, Test, Staging and Production environments. Besides this, they also help in scaling-up and scaling-down of instances swiftly.
- Containerization tools help in maintaining consistency across the environments where the application is developed, tested and deployed. Using these tools, there is no scope of errors/ failure in the production environment as they package and replicate the same dependencies and packages used in the development/ testing/ staging environment. It makes your application easy to run on different computers.

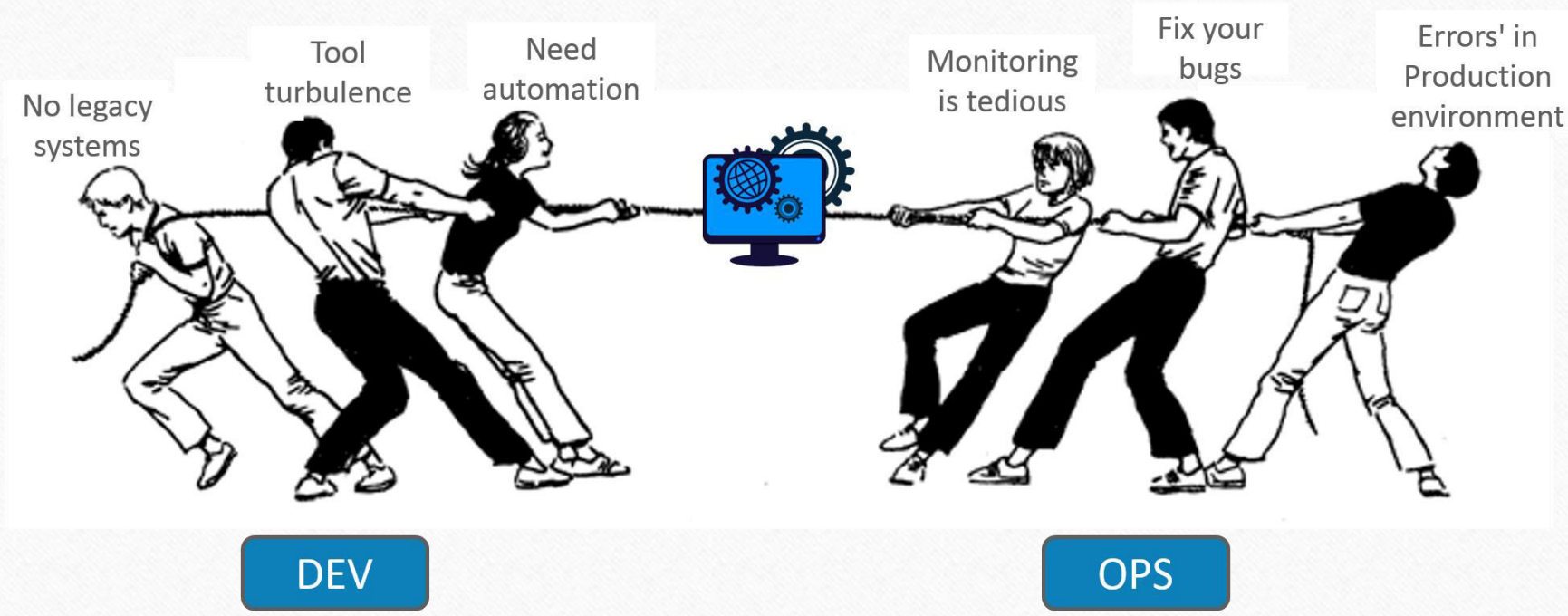


- **Continuous Monitoring –**
- This is a very crucial stage of the DevOps life cycle where you continuously monitor the performance of your application. Here vital information about the use of the software is recorded. This information is processed to recognize the proper functionality of the application. The system errors such as low memory, server not reachable, etc are resolved in this phase.
- The root cause of any issue is determined in this phase. It maintains the security and availability of the services. Also if there are network issues, they are resolved in this phase. It helps us automatically fix the problem as soon as they are detected.
- This practice involves the participation of the Operations team who will monitor the user activity for bugs or any improper behavior of the system. The popular tools used for this are Splunk, ELK Stack, Nagios, NewRelic and Sensu. These tools help you monitor the application's performance and the servers closely and also enable you to check the health of the system proactively.
- They can also improve productivity and increase the reliability of the systems, which in turn reduces IT support costs. Any major issues if found are reported to the development team so that it can be fixed in the continuous development phase. This leads to a faster resolution of the problems.
- These DevOps stages are carried out on loop continuously till you achieve the desired product quality. Therefore almost all of the major IT companies have shifted to DevOps for building their products.

DevOps vs Agile

Features	DevOps	Agile
Agility	Agility in both Development & Operations	Agility in only Development
Processes/ Practices	Involves processes such as CI, CD, CT, etc.	Involves practices such as Agile Scrum, Agile Kanban, etc.
Key Focus Area	Timeliness & quality have equal priority	Timeliness is the main priority
Release Cycles/ Development Sprints	Smaller release cycles with immediate feedback	Smaller release cycles
Source of Feedback	Feedback is from self (Monitoring tools)	Feedback is from customers
Scope of Work	Agility & need for Automation	Agility only

Why Is DevOps Better Than Agile?



- Let's understand this by first learning what were the challenges with Agile software development.
- Agile software development is about following a set of best practices for creating quality software in a timely manner. But the problem is, the best practices followed, involves people working in **Silos**.
- By Silos,We mean there are people who will be working as *Developers*, or as *Testers*, or as *ITOps* with very little communication between them. And since, there is very little communication between them, they are not aware what the others are working on despite being a part of the same process.
- This Silos-ed working of teams is the reason for the infamous "Blame Game" that goes about when a software fails or has major flaws.

- **The Blame Game**
- When a client has complains about a software, the blame is internally thrown at each other. The ‘Dev’ team would point fingers at the ‘QA’ team. ‘QA’ team will then point fingers at the ‘ITOps’ team, who would redirect the blame to the ‘Dev’ team.
- Irrespective of the problem residing in the code developed, or on the systems where the code is deployed, the problem remains in isolation, as nobody wants to take ownership for the screw-up.

- **Solution To This Everlasting Problem?**
- **DevOps**! You could have guessed this. But, can you guess how DevOps overcomes the Silos?
- Simple- DevOps breaks the Silos right through the middle. In DevOps, the 'Dev' team, the 'ITOps' team and 'QA' team are not independently working pieces of the gamut. But, they are 'one'.

- DevOps practice uses a *DevOps Engineer* – who does everything:- developing the code, testing that code and deploying the very same code to production. So, does the unification solve the problem?
- Yes, it solves one major aspect of the problem. Since the same DevOps Engineer is multi-skilled, he will be given ownership of the entire process: developing the code, unit testing/ functional testing the code and deploying that code to staging/ testing/ production sever.
- Since he is the sole owner, there are very few problems that will arise. And even if problems do arise, the person who knows the product best, will be on the job.
- Speaking of the best person, another issue that DevOps solves is the dependency problem. So, even if the 'ITOps' guy is not available, there won't be any delay. Because as DevOps Engineers, the role of 'ITOps' can be easily assumed by anybody else.

- **Is DevOps Performed By Only DevOps Engineers?**
- Well, that's the catch. It always seems like DevOps Engineers are the only folks involved. But, in the real world, DevOps Engineers are restricted to only performing a specified role even though they are capable of being involved throughout the entire lifecycle.

Technical Differences Between DevOps vs Agile Process Or Practices?

- **Key Focus Area?**
- Agile development focuses mainly on releasing quality software in a timely manner.
- DevOps goes one step further. It focuses on guaranteeing quality software in a timely manner. Quality is guaranteed by *Continuously Monitoring* the software application after its deployment.

- **Release Cycles/ Development Sprints**
- Agile focuses on smaller release cycles with incremental software delivery.
- DevOps focuses on smaller release cycles with incremental delivery & immediate feedback.

- **Who Gives Feedback?**
- In Agile, feedback is mostly given by customers.
- In DevOps, feedback is mostly measured by the internal team (by using Continuous Monitoring tools).

- **Scope Of Work**
- Agile mainly focuses on working with Speed or Agility.
- DevOps mainly focuses on achieving automation by orchestrating various DevOp tools.

Requirements Elicitation

- **Requirements elicitation** is perhaps the most difficult, most error-prone and most communication intensive software development. It can be successful only through an effective customer-developer partnership. It is needed to know what the users really need.

Requirements elicitation Activities

- Requirements elicitation includes the subsequent activities. Few of them are listed below –
- Knowledge of the overall area where the systems is applied.
- The details of the precise customer problem where the system are going to be applied must be understood.
- Interaction of system with external requirements.
- Detailed investigation of user needs.
- Define the constraints for system development.

Requirements elicitation Methods

- There are a number of requirements elicitation methods. Few of them are listed below –
  - Interviews
  - Brainstorming Sessions
  - Facilitated Application Specification Technique (FAST)
  - Quality Function Deployment (QFD)
  - Use Case Approach
- The success of an elicitation technique used depends on the maturity of the analyst, developers, users, and the customer involved.