PRESIDENCY UNIVERISTY, BENGALURU
School of Information Science

Bachelor of Computer Applications

Introduction to DevOps
(CSA 1007)
IV Semester 2022-23

Phases of Software Project

A software is developed by a series of phases as follows:

1) **Requirements gathering and analysis**
2) **Planning**
3) **Design**
4) **Development/Coding/Implementation**
5) **Testing**
6) **Deployment and Maintenance**

Requirements gathering and analysis

- **In this phase, the specific requirements of the software to be built are gathered and documented.**
- **The requirements get documented in the form of a System Requirements Specification (SRS) documented.**
- **SRS document acts as a bridge between the customer and software designers**

Components of SRS Document

- Functional Requirements
- Non - Functional Requirements
- Static Requirements
- Execution Constraints eg. Response time, throughput time
- Standards to be followed
- Security requirements
- Company Policies
- Interface with other external agents ie. persons, software or hardware

Planning

- **The purpose of Planning phase is to come up with a schedule, scope and resource requirements for a release.**
- **A plan explains requirements will be met and by what time.**
- **It needs to take into consideration the requirementns- what will be met and what will not be met**

Activities of Planning Phase of SDLC

14

- Define business problem and scope
- Produce detailed project schedule
- Confirm project feasibility
  - Economic, organizational, technical, resource, and schedule
- Staff the project (resource management)
- Launch project → official announcement

Design

- **Design Phase's purpose is to figure out how to satisfy requirements enumerated in the SRS document.**

- **It produces a representation (BluePrint) that will be used by the following phase, the development phase. This representation serves two purposes:**

  a) possible to verify all requirements are satisfies

  b) Gives info to development phases to proceed with coding.

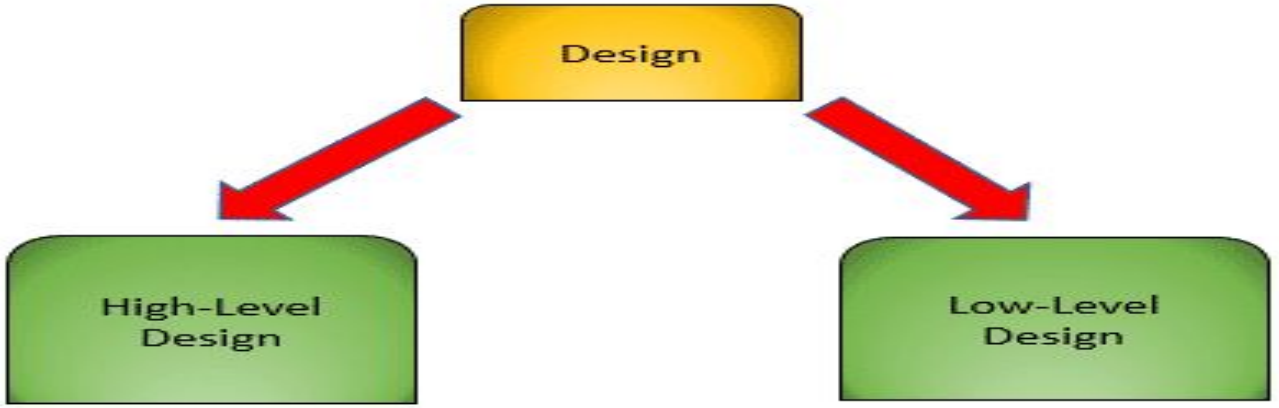
- **Design is split in High-Level Design (HDD) and Low-Level Design (SDD)**

- **This phase produces System Design Description (SDD) that will be used by development teams to produce the programs that realize the design**

Design

Design

- It is the most creative and challenging phase of SDLC.
- It defines the final system and refers to the technical specifications.
- DFDs are used to show the flow of system.
- Two phases: 1. Logical Design and Physical Design
- Logical Design: Specifies user needs.
- Physical Design: Tells the programmer what the candidate system must do.



Development or Coding

- **Design act as a blueprint for actual coding to proceed.**
- **This phase comprises coding the programs in choosen programming Language.**
- **In addition, this phase also involves the creation of product documentation**

Coding Phase:

1. The specifications from the detailed design phase are converted into programs.
2. Code is produced from the deliverables of the design phase during implementation, and this is the longest phase of the software development life cycle.
3. Design must be translated into a machine-readable form taking input as SRS.
    1. Done by Team of developers.
    2. Reviews after every 500 lines of code
        1. Code Inspection
        2. Code Walkthrough

Coding Phase:

Roles: Programmers/Developers

Task: Programming/Coding

Process:

Developer

TDD

SCD

1. Indentation
2. Proper color coding
3. Commenting

Coding Standards

Proof: SCD

DO NOT DISTRIBUTE - HIGHLY CONFIDENTIAL

Testing

As the programs are coded, they are also debugged and executed. After the coding is completed, the product is subjected to testing.

Testing is the process of exercising the software product in pre-defined ways to check if the behavior is the same as expected behavior.

Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect free in order to produce the quality product.

Software testing is a set of processes aimed at investigating, evaluating and ascertaining the completeness and quality of computer software. Software testing ensures the compliance of a software product in relation with regulatory, business, technical, functional and user requirements.

Deployment and Maintenance

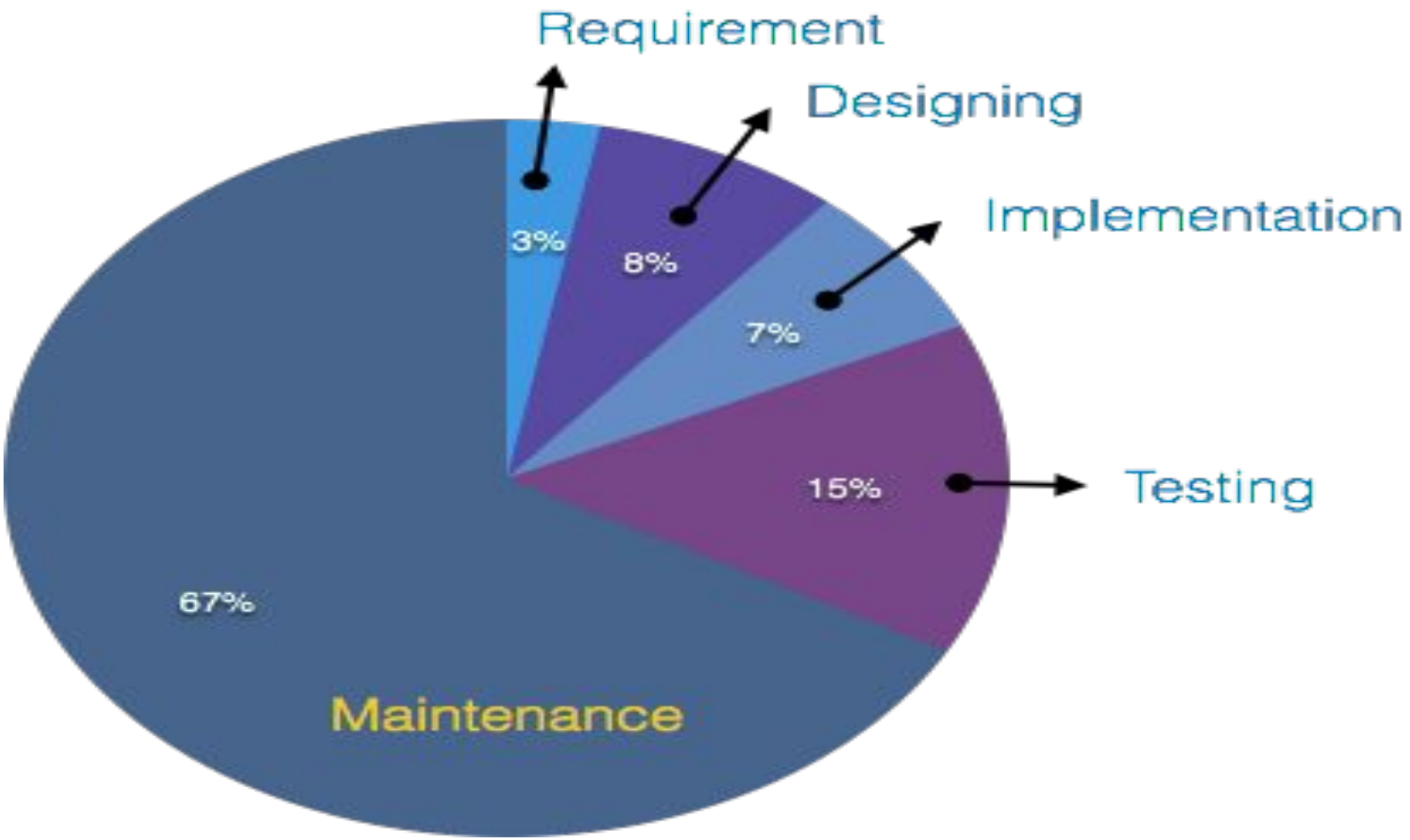
Once the software is tested, it is given to clients who deploy it in their environments.

As users starts using the product in their environment they may observe discrepancies between the actual behavior and what they were given to expect. Such defects need to be corrected.

The product now enters the maintenance phase, where the product is maintained or changed to satisfy customers expectations that arises from environmental changes.

Maintenance can be one among the following:
Corrective (Fixing customer related problems)
Adaptive (Making the S/w run on new version of OS or DB)
Preventive(Changing to code to avoid security threat)

Costs Involved in Various Phases of Software project



Quality, Quality assurance and Quality Control

Quality:
Quality is meeting the requirement, expectation, and needs of the customer is free from the defects, lacks and substantial variants. There are standards needs to follow to satisfy the customer requirements.

Quality Assurance is known as QA and focuses on preventing defect. Quality Assurance ensures that the approaches, techniques, methods and processes are designed for the projects are implemented correctly.

Quality assurance activities monitor and verify that the processes used to manage and create the deliverables have been followed and are operative.

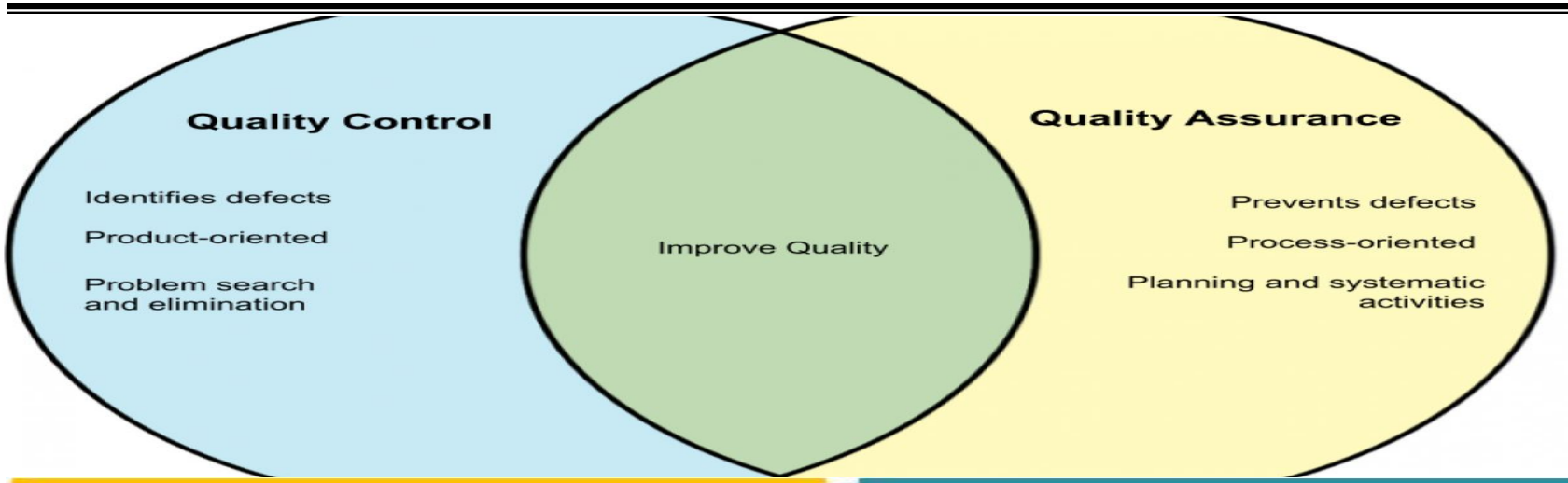
Quality Assurance is a proactive process and is Prevention in nature. It recognizes flaws in the process. Quality Assurance has to complete before Quality Control.

Quality, Quality assurance and Quality Control

Quality Control is known as QC and focuses on identifying a defect. QC ensures that the approaches, techniques, methods and processes are designed in the project are following correctly. QC activities monitor and verify that the project deliverables meet the defined quality standards.

Quality Control is a reactive process and is detection in nature. It recognizes the defects. Quality Control has to complete after Quality assurance

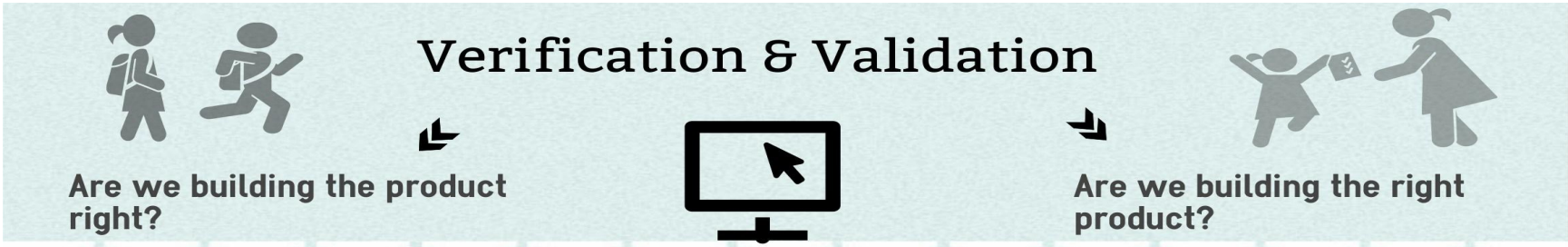
Quality, Quality assurance and Quality Control



Quality Control	Quality Assurance
Focused on Product	Focused on Process
Reactive	Pro-active
Line Function	Staff Function
Finds Defects	Prevent Defects
Testing	Quality Audits

Testing, Verification and Validation.

Verification focuses on: Are we building the product right?
Validation focuses on: Are we building the right product?



Verification	Validation
● Verify the intermediary products like requirement documents, design documents, ER diagrams, test plan and traceability matrix ● Developer point of view ● Verified without executing the software code ● Techniques used: Informal Review, Inspection, Walkthrough, Technical and Peer review	● Validate the final end product like developed software or service or system ● Customer point of view ● Validated by executing the software code ● Techniques used: Functional testing, System testing, Smoke testing, Regression testing and Many more

www.letzdotesting.com

Testing, Verification and Validation.

VERIFICATION	VALIDATION
It is a static practice of checking documents,design code and program	It is a dynamic practice of validating and testing the actual product
It doesnot involve code execution	It involve code exceution
It is human based checking of documents and files	It is computer based exection of Program
It uses walkthroughs,inspection and reviews	It uses blackbox testing,gray box testing and white box testing

Testing, Verification and Validation.

SOFTWARE VERIFICATION	SOFTWARE VALIDATION
1. It means, "Are we building any System/Product in a right manner?"	1. It means, "Are we building right System/Product ?".
2. Software Verification is the process of evaluating System/Products in the development phase to find whether they meet the specified requirements or not.	2. Software Validation is the process of evaluating software at the end of development process in order to determine whether software Product/System meets the customer expectations and requirements or not.
3. Software Verification process involves : a) Reviews. b) Meetings. c) Inspections.	3. Software Validation involves : a) Black Box Testing. b) White Box Testing. c) Grey Box Testing.
4. Verification of software is carried out by quality assurance team.	4. Validation of software is carried out by the testing team.
5. Execution of code is not done in case of Software Verification and is carried out before Validation process.	5. Code is executed in case of Software Validation and it is carried out after the Software Verification process.

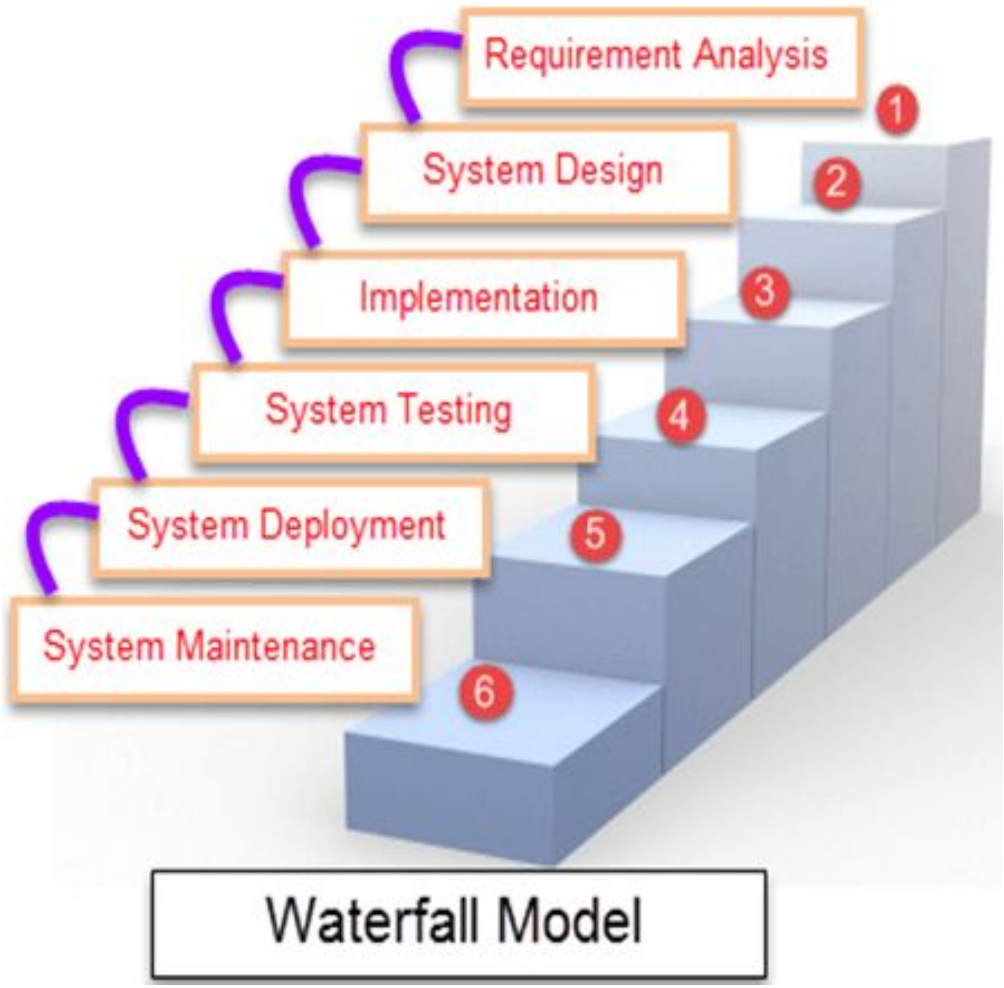
(Software Development Life Cycle (SDLC) Models

A framework that describes the activities performed at each stage of a software development project.

A software lifecycle model is a standardized format for planning organizing, and running a new development project

A (software/system) lifecycle model is a description of the sequence of activities carried out in an SE project, and the relative order of these activities.
Normally, a lifecycle model covers the entire lifetime of a product.

Waterfall Model



- **Requirements** – defines needed information, function, behavior, performance and interfaces.
- **Design** – data structures, software architecture, interface representations, algorithmic details.
- **Implementation** – source code, database, user documentation, testing.

Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

Waterfall Deficiencies

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Can give a false impression of progress
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

When to use the Waterfall Model

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.

Structured Evolutionary Prototyping Model

- Developers build a prototype during the requirements phase
- Prototype is evaluated by end users
- Users give corrective feedback
- Developers further refine the prototype
- When the user is satisfied, the prototype code is brought up to the standards needed for a final product.

Structured Evolutionary Prototyping Steps

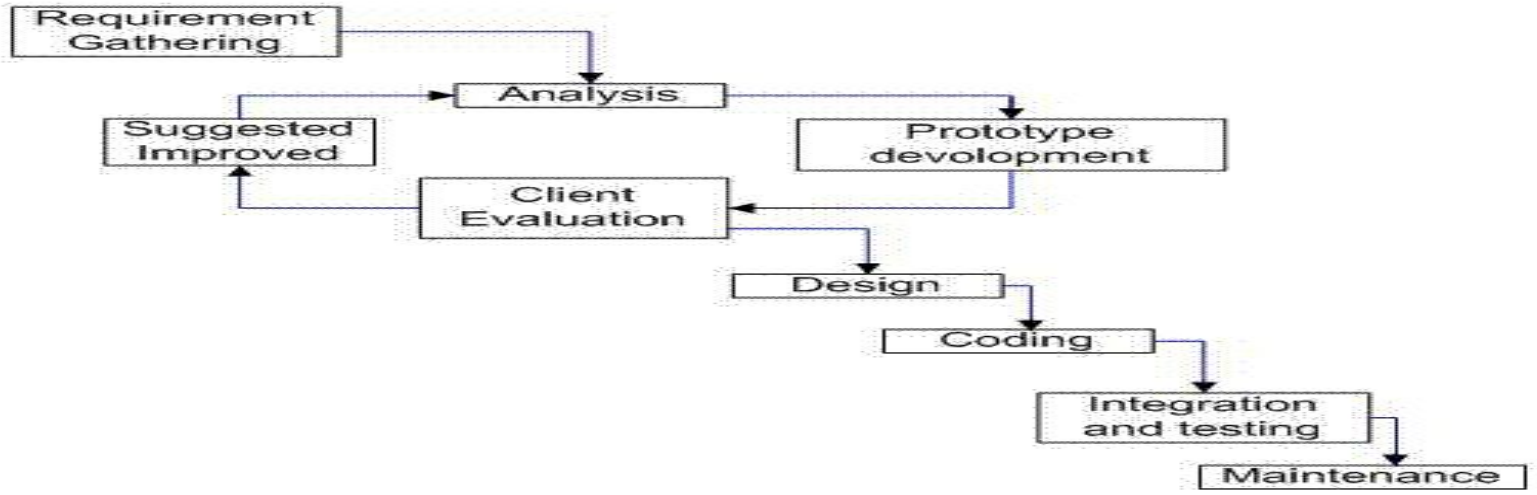
- A preliminary project plan is developed
- An partial high-level paper model is created
- The model is source for a partial requirements specification
- A prototype is built with basic and critical attributes
- The designer builds
  - the database
  - user interface
  - algorithmic functions
- The designer demonstrates the prototype, the user evaluates for problems and suggests improvements.
- This loop continues until the user is satisfied

Structured Evolutionary Prototyping Strengths

- Customers can "see" the system requirements as they are being gathered
- Developers learn from customers
- A more accurate end product
- Unexpected requirements accommodated
- Allows for flexible design and development
- Steady, visible signs of progress produced
- Interaction with the prototype stimulates awareness of additional needed functionality

Structured Evolutionary Prototyping Weaknesses

- Tendency to abandon structured program development for "code-and-fix" development
- Bad reputation for "quick-and-dirty" methods
- Overall maintainability may be overlooked
- The customer may want the prototype delivered.
- Process may continue forever (scope creep)

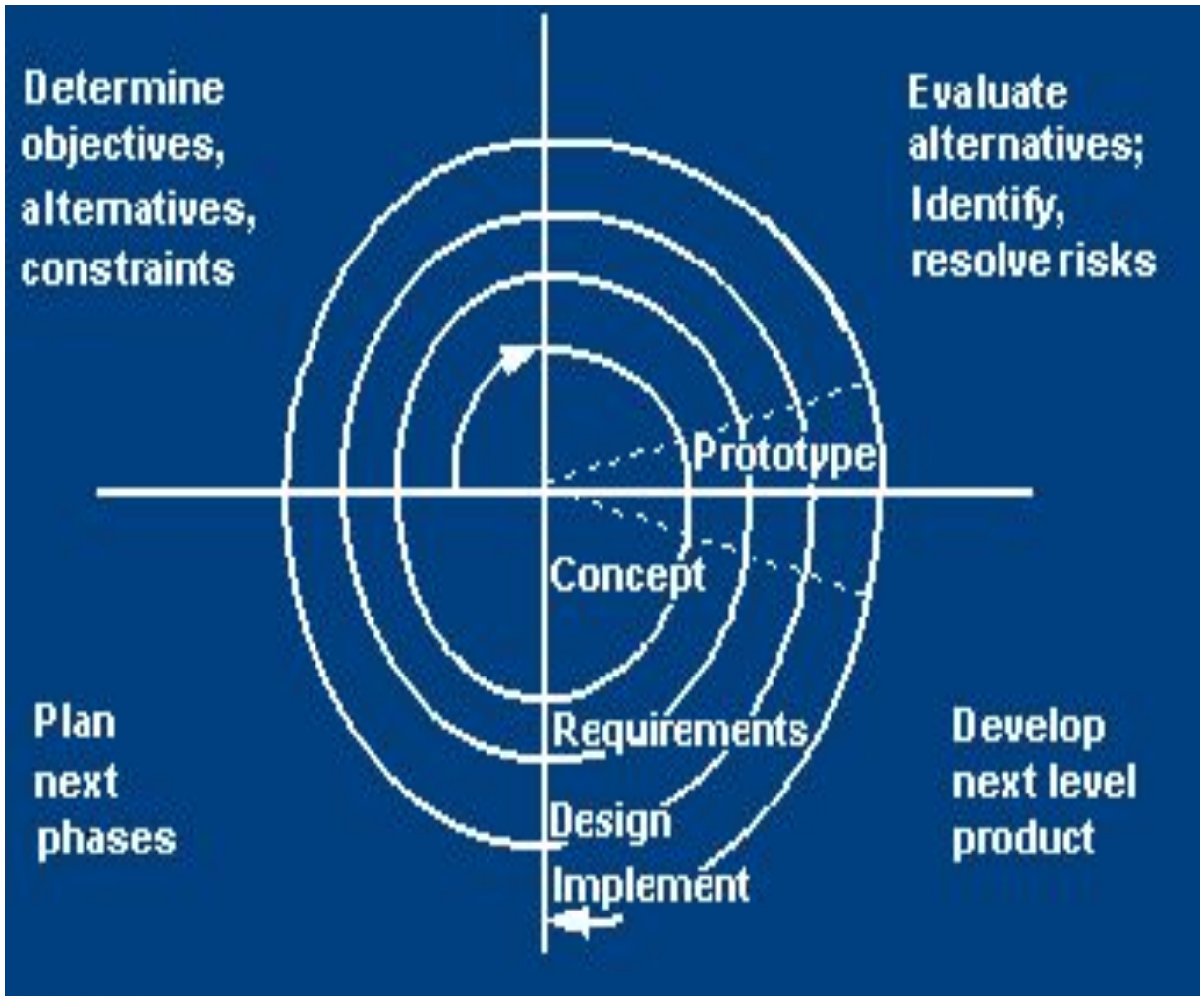


Evolutionary Prototyping Model

When to use Structured Evolutionary Prototyping

- Requirements are unstable or have to be clarified
- As the requirements clarification stage of a waterfall model
- Develop user interfaces
- Short-lived demonstrations
- New, original development
- With the analysis and design portions of object-oriented development.

Spiral or Iterative SDLC Model



- Adds risk analysis, and 4gl RAD prototyping to the waterfall model
- Each cycle involves the same sequence of steps as the waterfall process model

Spiral Quadrant Determine objectives, alternatives and constraints

- Objectives: functionality, performance, hardware/software interface, critical success factors, etc.
- Alternatives: build, reuse, buy, sub-contract, etc.
- Constraints: cost, schedule, interface, etc.

Spiral Quadrant Evaluate alternatives, identify and resolve risks

- Study alternatives relative to objectives and constraints
- Identify risks (lack of experience, new technology, tight schedules, poor process, etc.
- Resolve risks (evaluate if money could be lost by continuing system development

Spiral Quadrant Develop next-level product

- Typical activites:
  - Create a design
  - Review design
  - Develop code
  - Inspect code
  - Test product

Spiral Quadrant Plan next phase

- Typical activities
  - Develop project plan
  - Develop configuration management plan
  - Develop a test plan
  - Develop an installation plan

Spiral Model Strengths

- Provides early indication of insurmountable risks, without much cost
- Users see the system early because of rapid prototyping tools
- Critical high-risk functions are developed first
- The design does not have to be perfect
- Users can be closely tied to all lifecycle steps
- Early and frequent feedback from users
- Cumulative costs assessed frequently

Spiral Model Weaknesses

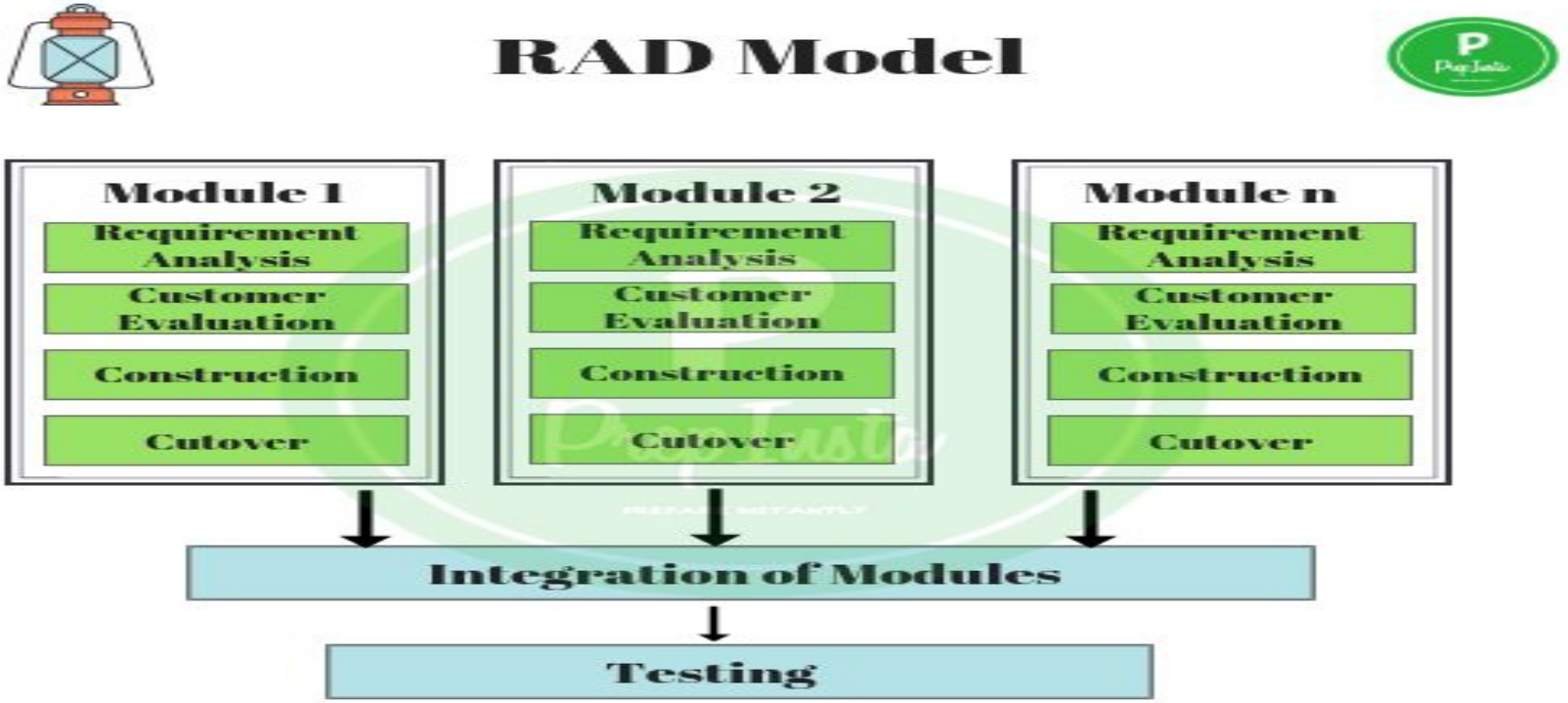
- Time spent for evaluating risks too large for small or low-risk projects
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- The model is complex
- Risk assessment expertise is required
- Spiral may continue indefinitely
- Developers must be reassigned during non-development phase activities
- May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

When to use Spiral Model

- When creation of a prototype is appropriate
- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

Rapid Application Model (RAD)

- Requirements planning phase (a workshop utilizing structured discussion of business problems)
- User description phase – automated tools capture information from users
- Construction phase – productivity tools, such as code generators, screen generators, etc. inside a time-box. ("Do until done")
- Cutover phase -- installation of the system, user acceptance testing and user training



RAD Strengths

- Reduced cycle time and improved productivity with fewer people means lower costs
- Time-box approach mitigates cost and schedule risk
- Customer involved throughout the complete cycle minimizes risk of not achieving customer satisfaction and business needs
- Focus moves from documentation to code (WYSIWYG).
- Uses modeling concepts to capture information about business, data, and processes.

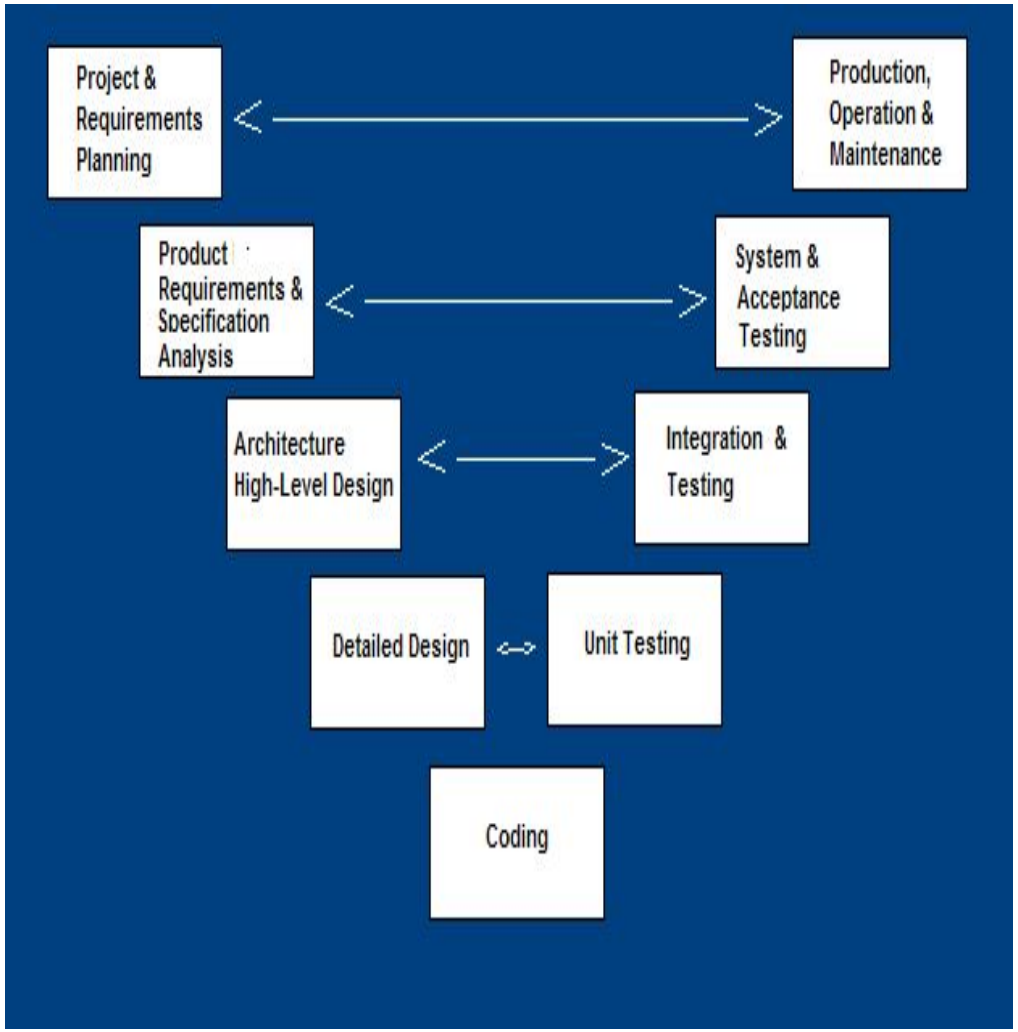
RAD Weaknesses

- Accelerated development process must give quick responses to the user
- Risk of never achieving closure
- Hard to use with legacy systems
- Requires a system that can be modularized
- Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.

When to use RAD

- Reasonably well-known requirements
- User involved throughout the life cycle
- Project can be time-boxed
- Functionality delivered in increments
- High performance not required
- Low technical risks
- System can be modularized

V-Shaped SDLC Model



- A variant of the Waterfall that emphasizes the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

V-Shaped Steps

- Project and Requirements Planning – allocate resources
- Product Requirements and Specification Analysis – complete specification of the software system
- Architecture or High-Level Design – defines how software functions fulfill the design
- Detailed Design – develop algorithms for each architectural component
- Production, operation and maintenance – provide for enhancement and corrections
- System and acceptance testing – check the entire software system in its environment
- Integration and Testing – check that modules interconnect correctly
- Unit testing – check that each module acts as expected
- Coding – transform algorithms into software

V-Shaped Strengths

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

V-Shaped Weaknesses

- Does not easily handle concurrent events
- Does not handle iterations or phases
- Does not easily handle dynamic changes in requirements
- Does not contain risk analysis activities

When to use the V-Shaped Model

- Excellent choice for systems requiring high reliability – hospital patient control applications
- All requirements are known up-front
- When it can be modified to handle changing requirements beyond analysis phase
- Solution and technology are known

Comparison Of SDLC Models

		Water fall	V Model	Increm-ental	Spiral	Proto type	RAD
1	Well Defined Requirement	Yes	Yes	No	No	No	Yes
2	Domain Knowledge of Team member	Adeq-uate	Adeq-uate	Adeq-uate			Adeq-uate
3	Expertise of Users in Problem Domain	Very Less	Very Less	Adeq-uate	Very Less	Adeq-uate	Adeq-uate
4	Availability of Reusable Components	No	No	No	Yes	Yes	Yes
5	User involvement in all SDLC Phase	No	No	No	No	Yes	Yes
6	Complexity of the System	Simple	Simple	Complex	Complex	Complex	Medium

Comparison of Life cycle models

S.N	Waterfall model	Iterative model	Spiral model	V shaped Model
1.	Simple and easy to use.	More flexible than the basic waterfall model.	High amount of risk analysis	Simple and easy to use.
2.	Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.	If there is personnel continuity between the phases, documentation can be substantially reduced.	Good for large and mission-critical projects	Each phase has specific deliverables.
3.	Phases are processed and completed one at a time.	Implementation of easy areas does not need to wait for the hard ones.	Software is produced early in the software life cycle.	Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.
4.	Works well for smaller projects where requirements are very well understood.	Works well for smaller and moderate size projects	Works well for projects where risk analysis contains higher priority.	Works well for small projects where requirements are easily understood

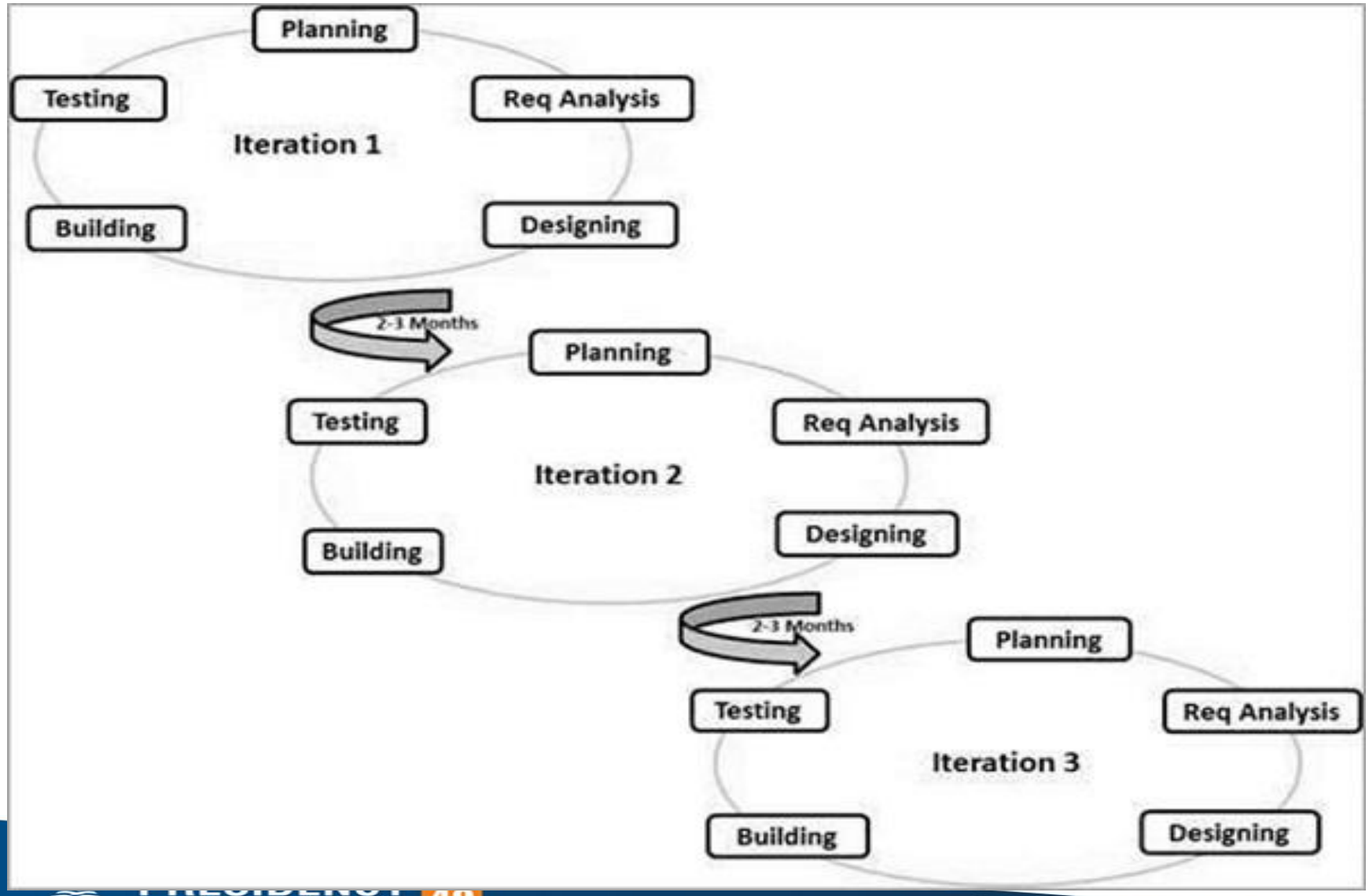
SDLC - Agile Model

- Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.
- Agile Methods break the product into small incremental builds.
- These builds are provided in iterations. Each iteration typically lasts from about one to three weeks.
- Every iteration involves cross functional teams working simultaneously on various areas like −
    - Planning
    - Requirements Analysis
    - Design
    - Coding
    - Unit Testing and
    - Acceptance Testing.
- At the end of the iteration, a working product is displayed to the customer and important stakeholders.

What is Agile?

- Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements.
- In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.
- Iterative approach is taken and working software build is delivered after each iteration.
- Each build is incremental in terms of features; the final build holds all the features required by the customer.

Graphical illustration of Agile Model



- The Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.
- The most popular Agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995).
- These are now collectively referred to as **Agile Methodologies**, after the Agile Manifesto was published in 2001.
- Following are the Agile Manifesto principles −
  - **Individuals and interactions** − In Agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
  - **Working software** − Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation.
  - **Customer collaboration** − As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
  - **Responding to change** − Agile Development is focused on quick responses to change and continuous development.

Agile Vs Traditional SDLC Models

- Agile is based on the **adaptive software development methods**, whereas the traditional SDLC models like the waterfall model is based on a predictive approach.
- Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle.
- Predictive methods entirely depend on the **requirement analysis and planning** done in the beginning of cycle.
- Any changes to be incorporated go through a strict change control management and prioritization.
- Agile uses an **adaptive approach** where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed.
- There is feature driven development and the team adapts to the changing product requirements dynamically.
- The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.
- **Customer Interaction** is the backbone of this Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment.
- The agile teams work in close collaboration with each other and are most often located in the same geographical location.

AGILE – Pros and Cons

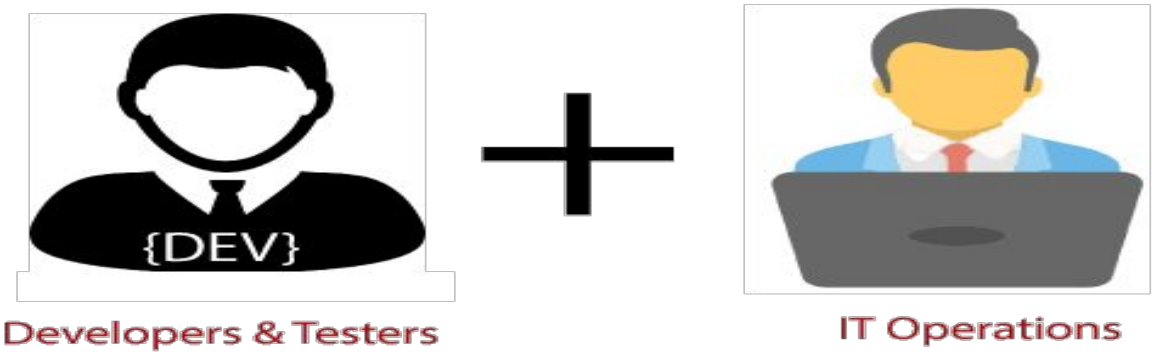
- **Pros**
  - **Project is divided into short and transparent iterations.**
  - **It has a flexible change process.**
  - **Quick release of the first product version.**
  - **The correctness of functional requirement is implemented into the development process.**
  - **Customer can see the result and understand whether he/she is satisfied with it or not**
- **Cons**
  - **The development team should be highly professional and client-oriented.**
  - **New requirement may be a conflict with the existing architecture.**
  - **With further correction and change, there may be chances that the project will cross the expected time.**
  - **There may be difficult to estimate the final coast of the project due to constant iteration.**
  - **A defined requirement is absent**

Introduction to DevOps

The DevOps is a combination of two words, one is software Development, and second is Operations. This allows a single team to handle the entire application lifecycle, from development to testing, deployment, and operations.

DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators.

What is DevOps?



DevOps promotes collaboration between Development and Operations team to deploy code to production faster in an automated & repeatable way.

DevOps helps to increase organization speed to deliver applications and services. It also allows organizations to serve their customers better and compete more strongly in the market.

DevOps can also be defined as a sequence of development and IT operations with better communication and collaboration.

DevOps is nothing but a practice or methodology of making "Developers" and "Operations" folks work together.

DevOps represents a change in the IT culture with a complete focus on rapid IT service delivery through the adoption of agile practices in the context of a system-oriented approach.

DevOps has become one of the most valuable business disciplines for enterprises or organizations.

With the help of DevOps, quality, and speed of the application delivery has improved to a great extent.

Why DevOps?

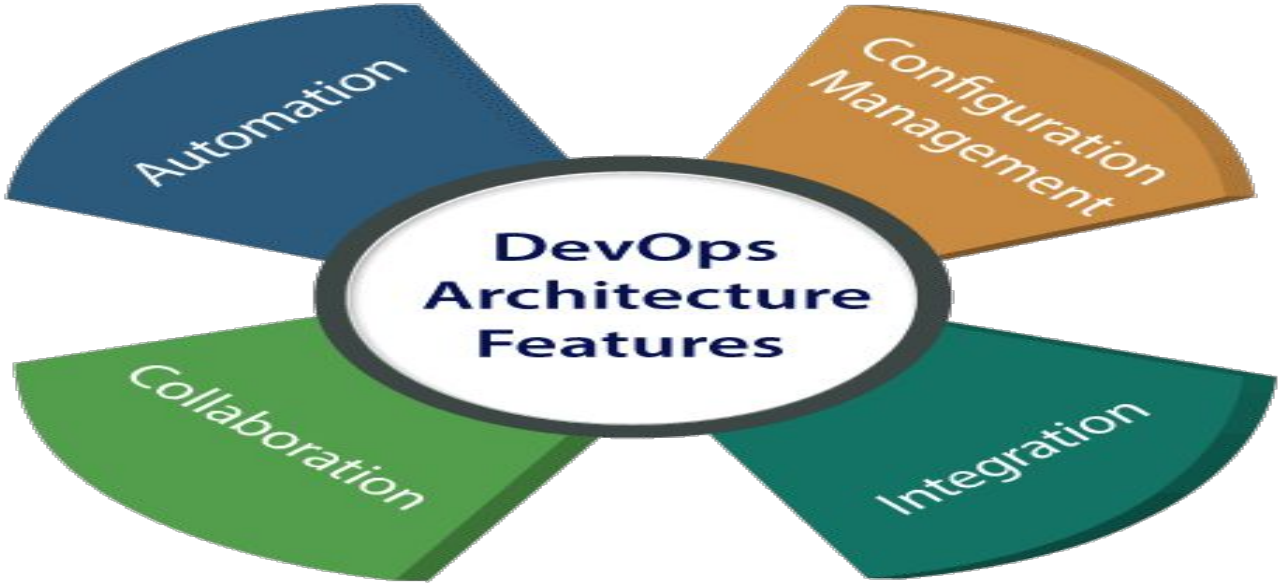
Before going further, we need to understand why we need the DevOps over the other methods.

- The operation and development team worked in complete isolation.
- After the design-build, the testing and deployment are performed respectively. That's why they consumed more time than actual build cycles.
- Without the use of DevOps, the team members are spending a large amount of time on designing, testing, and deploying instead of building the project.
- Manual code deployment leads to human errors in production.
- Coding and operation teams have their separate timelines and are not in synch, causing further delays.

DevOps History

- In 2009, the first conference named DevOpsdays was held in Ghent Belgium. Belgian consultant and Patrick Debois founded the conference.
- In 2012, the state of DevOps report was launched and conceived by Alanna Brown at Puppet.
- In 2014, the annual State of DevOps report was published by Nicole Forsgren, Jez Humble, Gene Kim, and others. They found DevOps adoption was accelerating in 2014 also.
- In 2015, Nicole Forsgren, Gene Kim, and Jez Humble founded DORA (DevOps Research and Assignment).
- In 2017, Nicole Forsgren, Gene Kim, and Jez Humble published "Accelerate: Building and Scaling High Performing Technology Organizations".

DevOps Architecture Features



DevOps Architecture Features

1) Automation

Automation can reduce time consumption, especially during the testing and deployment phase. The productivity increases, and releases are made quicker by automation. This will lead in catching bugs quickly so that it can be fixed easily. For contiguous delivery, each code is defined through automated tests, cloud-based services, and builds. This promotes production using automated deploys.

2) Collaboration

The Development and Operations team collaborates as a DevOps team, which improves the cultural model as the teams become more productive with their productivity, which strengthens accountability and ownership. The teams share their responsibilities and work closely in sync, which in turn makes the deployment to production faster.

DevOps Architecture Features

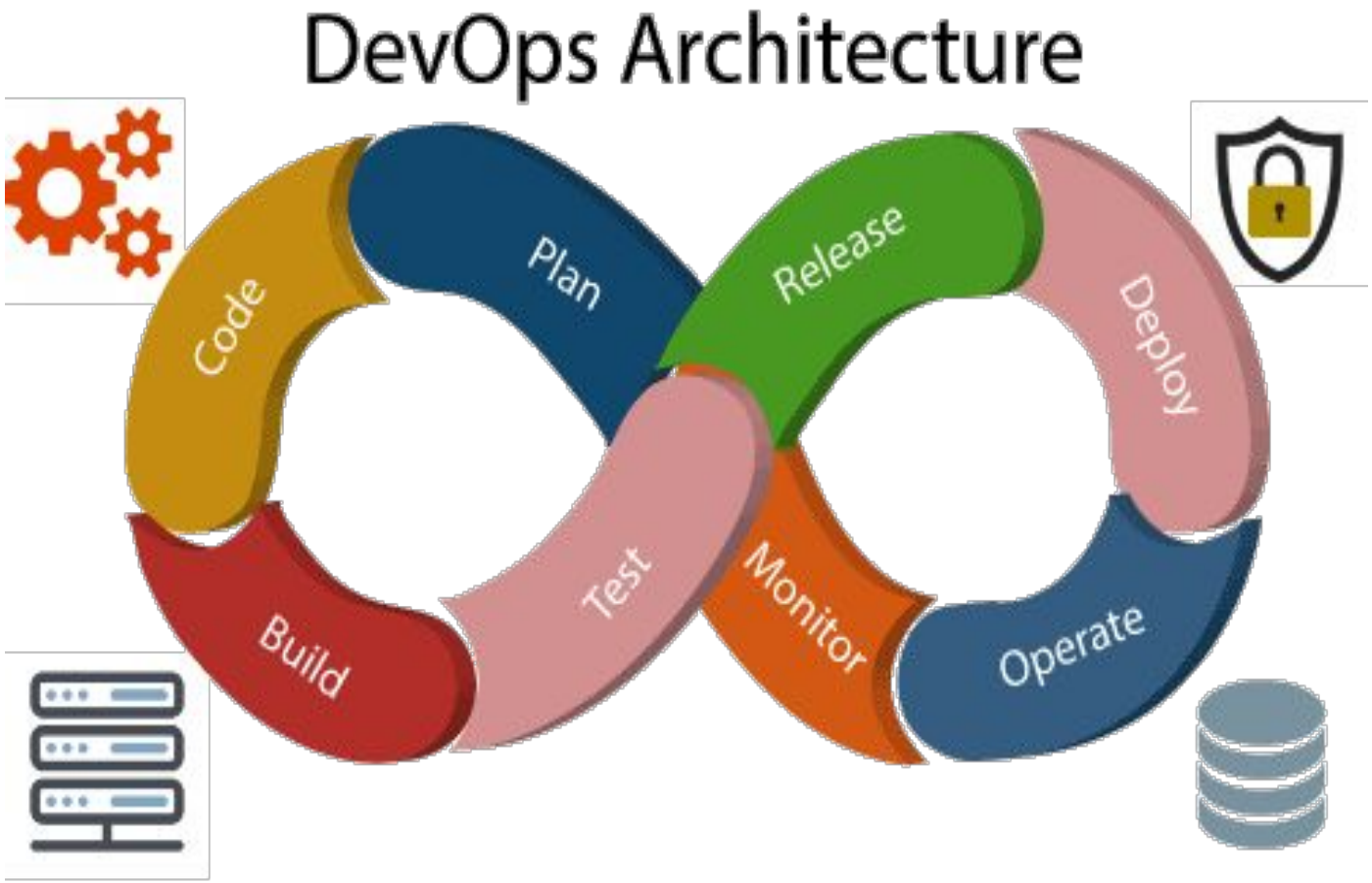
3) Integration

Applications need to be integrated with other components in the environment. The integration phase is where the existing code is combined with new functionality and then tested. Continuous integration and testing enable continuous development. The frequency in the releases and micro-services leads to significant operational challenges. To overcome such problems, continuous integration and delivery are implemented to deliver in a quicker, safer, and reliable manner.

4) Configuration management

It ensures the application to interact with only those resources that are concerned with the environment in which it runs. The configuration files are not created where the external configuration to the application is separated from the source code. The configuration file can be written during deployment, or they can be loaded at the run time, depending on the environment in which it is running

DevOps Architecture



DevOps Architecture

- Development and operations both play essential roles in order to deliver applications.
- The deployment comprises analyzing the requirements, designing, developing, and testing of the software components or frameworks.
- The operation consists of the administrative processes, services, and support for the software.
- When both the development and operations are combined with collaborating, then the DevOps architecture is the solution to fix the gap between deployment and operation terms; therefore, delivery can be faster

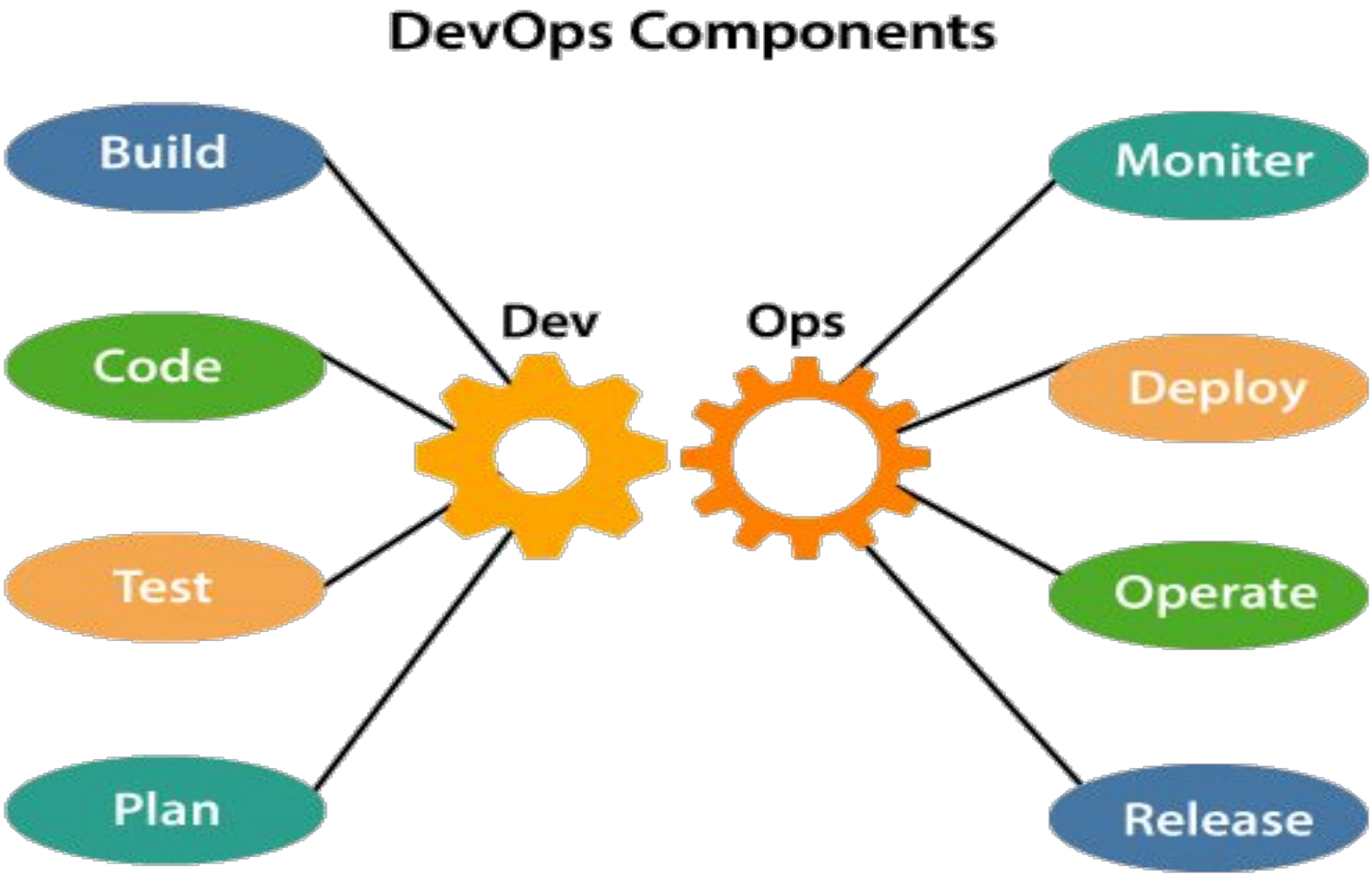
DevOps Architecture

DevOps architecture is used for the applications hosted on the cloud platform and large distributed applications.

Agile Development is used in the DevOps architecture so that integration and delivery can be contiguous. When the development and operations team works separately from each other, then it is time-consuming to design, test, and deploy.

And if the terms are not in sync with each other, then it may cause a delay in the delivery. So DevOps enables the teams to change their shortcomings and increases productivity

DevOps Components



Dev Components

1) Build

Without DevOps, the cost of the consumption of the resources was evaluated based on the pre-defined individual usage with fixed hardware allocation.

And with DevOps, the usage of cloud, sharing of resources comes into the picture, and the build is dependent upon the user's need, which is a mechanism to control the usage of resources or capacity.

2) Code

Many good practices such as Git enables the code to be used, which ensures writing the code for business, helps to track changes, getting notified about the reason behind the difference in the actual and the expected output, and if necessary reverting to the original code developed.

The code can be appropriately arranged in **files, folders**, etc. And they can be reused.

Dev Components

3) Test

The application will be ready for production after testing. In the case of manual testing, it consumes more time in testing and moving the code to the output.

The testing can be automated, which decreases the time for testing so that the time to deploy the code to production can be reduced as automating the running of the scripts will remove many manual steps.

4) Plan

DevOps use Agile methodology to plan the development. With the operations and development team in sync, it helps in organizing the work to plan accordingly to increase productivity.

Ops Components

5) Monitor

Continuous monitoring is used to identify any risk of failure. Also, it helps in tracking the system accurately so that the health of the application can be checked.

The monitoring becomes more comfortable with services where the log data may get monitored through many third-party tools such as **Splunk**.

6) Deploy

Many systems can support the scheduler for automated deployment. The cloud management platform enables users to capture accurate insights and view the optimization scenario, analytics on trends by the deployment of dashboards.

Ops Components

7) Operate

DevOps changes the way traditional approach of developing and testing separately. The teams operate in a collaborative way where both the teams actively participate throughout the service lifecycle.

The operation team interacts with developers, and they come up with a monitoring plan which serves the IT and business requirements.

8) Release

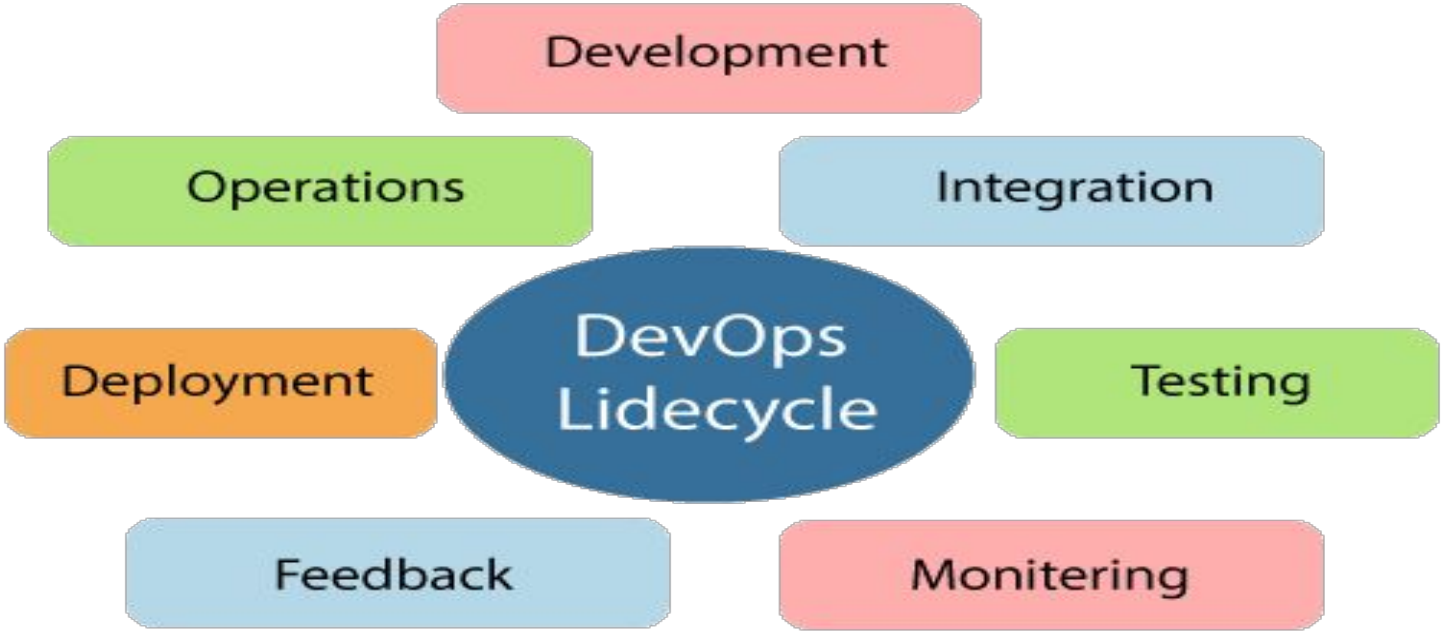
Deployment to an environment can be done by automation. But when the deployment is made to the production environment, it is done by manual triggering.

Many processes involved in release management commonly used to do the deployment in the production environment manually to lessen the impact on the customers.

DevOps Lifecycle

DevOps defines an agile relationship between operations and Development.

It is a process that is practiced by the development team and operational engineers together from beginning to the final stage of the product.



Learning DevOps is not complete without understanding the DevOps lifecycle phases. The DevOps lifecycle includes seven phases as given below:

1) Continuous Development

This phase involves the planning and coding of the software.

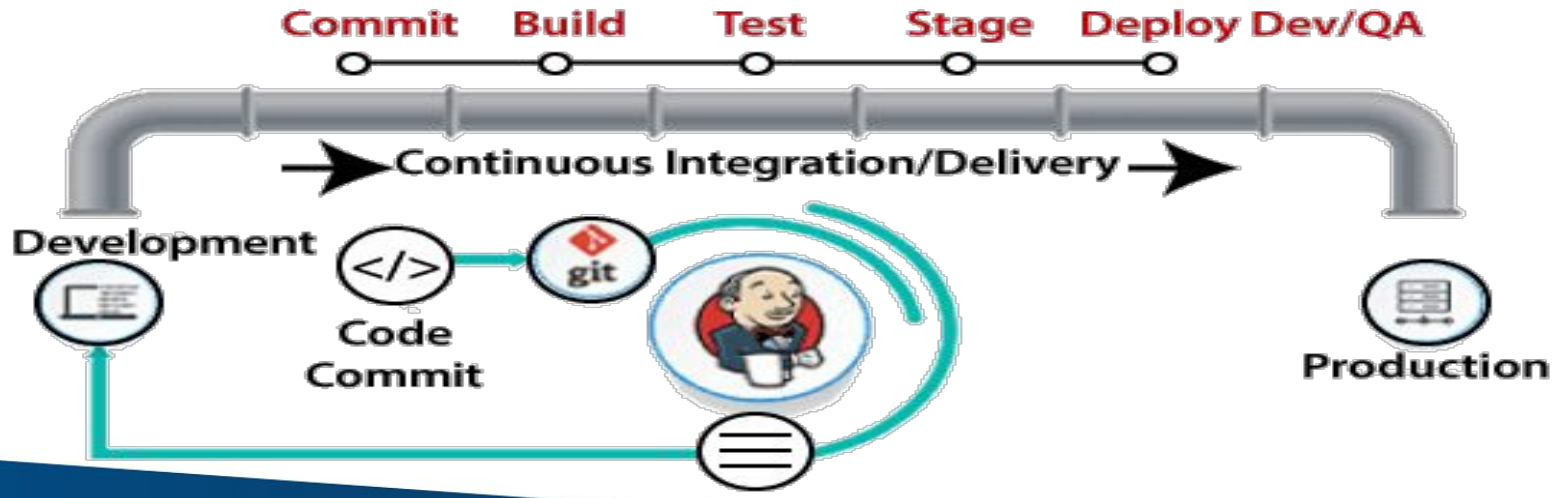
The vision of the project is decided during the planning phase.

And the developers begin developing the code for the application.

There are no DevOps tools that are required for planning, but there are several tools for maintaining the code.

2) Continuous Integration

- This stage is the heart of the entire DevOps lifecycle. 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 weekly basis. Then every commit is built, and this allows early detection of problems if they are present. Building code is not only involved compilation, but it also includes unit testing, integration testing, code review, and packaging.
- The code supporting new functionality is continuously integrated with the existing code. Therefore, there is continuous development of software. The updated code needs to be integrated continuously and smoothly with the systems to reflect changes to the end-users.



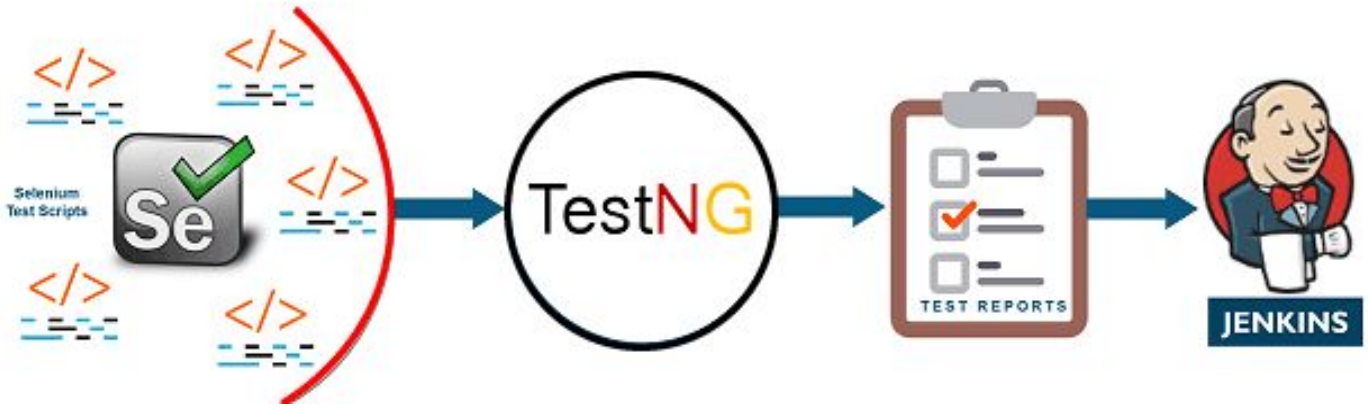
2) Continuous Integration

- Jenkins is a popular tool used in this phase. Whenever there is a change in the Git repository, then Jenkins fetches the updated code and prepares a build of that code, which is an executable file in the form of war or jar.
- Then this build is forwarded to the test server or the production server

3) Continuous Testing

This phase, where the developed software is continuously testing for bugs. For constant testing, automation testing tools such as TestNG, JUnit, Selenium, etc are used.

These tools allow QAs to test multiple code-bases thoroughly in parallel to ensure that there is no flaw in the functionality. In this phase, Docker Containers can be used for simulating the test environment.



4) Continuous Monitoring

- Monitoring is a phase that involves all the operational factors of the entire DevOps process, where important information about the use of the software is recorded and carefully processed to find out trends and identify problem areas.
- Usually, the monitoring is integrated within the operational capabilities of the software application.
- It may occur in the form of documentation files or maybe produce large-scale data about the application parameters when it is in a continuous use position.
- The system errors such as server not reachable, low memory, etc are resolved in this phase. It maintains the security and availability of the service

3) Continuous Testing

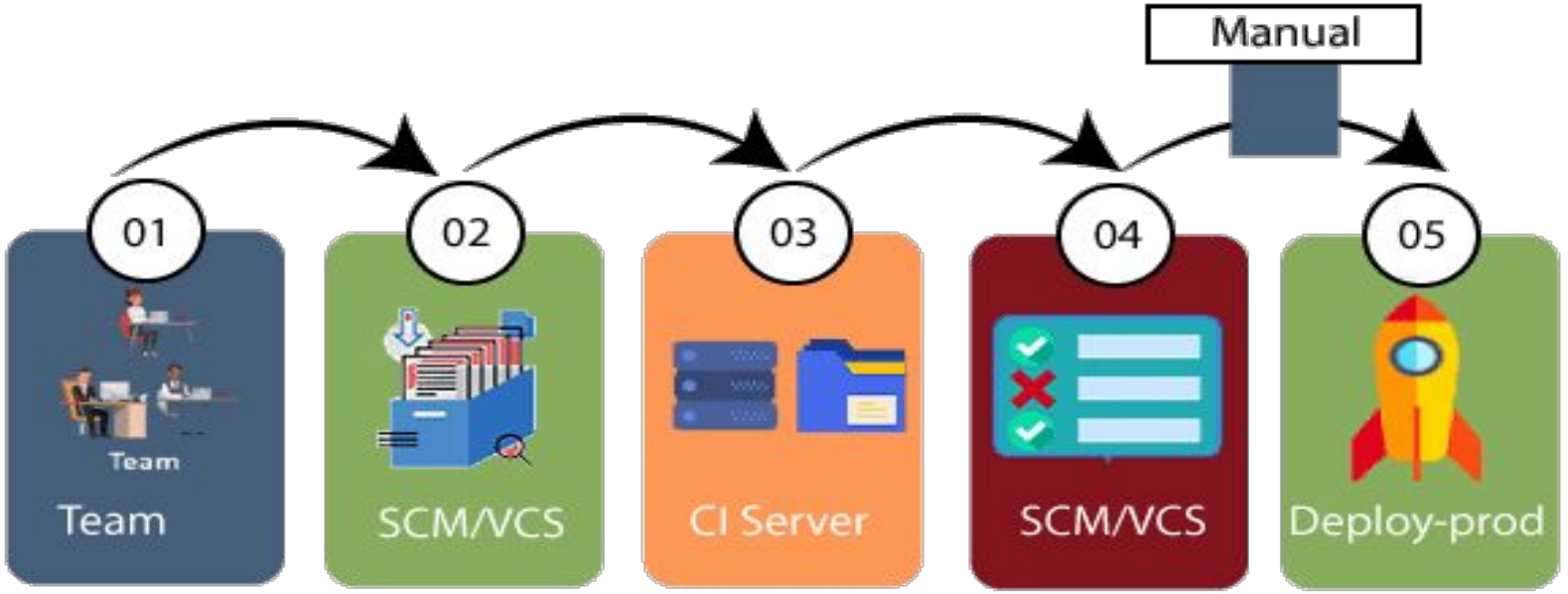
- Selenium does the automation testing, and TestNG generates the reports. This entire testing phase can automate with the help of a Continuous Integration tool called Jenkins.
- Automation testing saves a lot of time and effort for executing the tests instead of doing this manually. Apart from that, report generation is a big plus. The task of evaluating the test cases that failed in a test suite gets simpler.
- Also, we can schedule the execution of the test cases at predefined times. After testing, the code is continuously integrated with the existing code.

5) Continuous Feedback

- The application development is consistently improved by analyzing the results from the operations of the software.
- This is carried out by placing the critical phase of constant feedback between the operations and the development of the next version of the current software application.
- The continuity is the essential factor in the DevOps as it removes the unnecessary steps which are required to take a software application from development, using it to find out its issues and then producing a better version.
- It kills the efficiency that may be possible with the app and reduce the number of interested customers.

6) Continuous Deployment

In this phase, the code is deployed to the production servers. Also, it is essential to ensure that the code is correctly used on all the servers.



6) Continuous Deployment

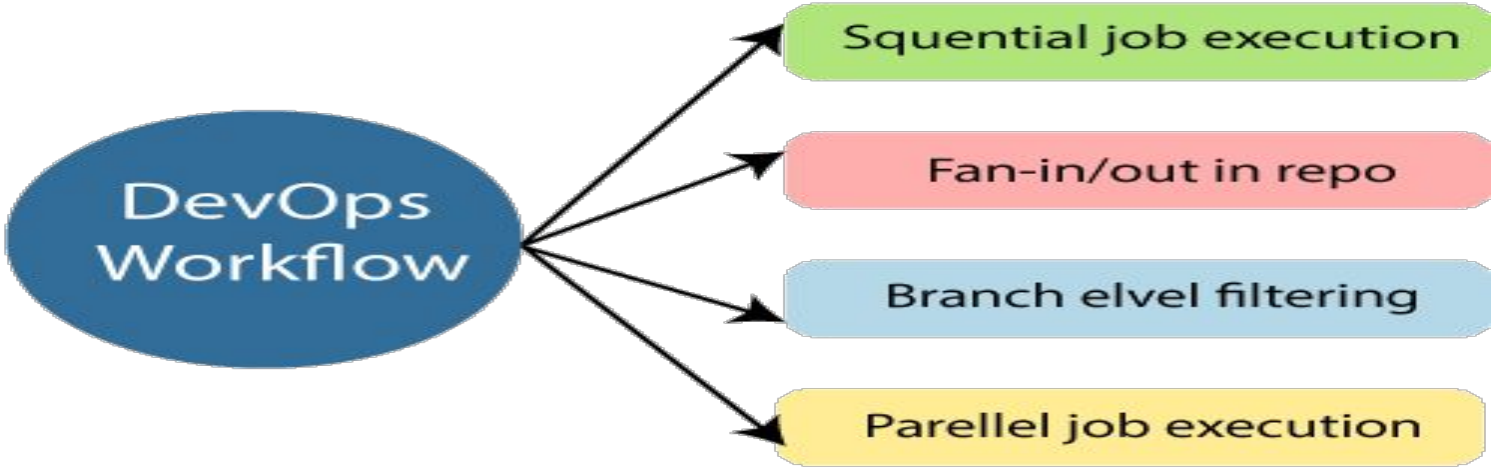
- The new code is deployed continuously, and configuration management tools play an essential role in executing tasks frequently and quickly. Here are some popular tools which are used in this phase, such as Chef, Puppet, Ansible, and SaltStack.
- Containerization tools are also playing an essential role in the deployment phase. Vagrant and Docker are popular tools that are used for this purpose.
- These tools help to produce consistency across development, staging, testing, and production environment. They also help in scaling up and scaling down instances softly.
- Containerization tools help to maintain consistency across the environments where the application is tested, developed, and deployed.
- There is no chance of errors or failure in the production environment as they package and replicate the same dependencies and packages used in the testing, development, and staging environment. It makes the application easy to run on different computers

7) Continuous Operations

- All DevOps operations are based on the continuity with complete automation of the release process and allow the organization to accelerate the overall time to market continuingly.
- It is clear from the discussion that continuity is the critical factor in the DevOps in removing steps that often distract the development, take it longer to detect issues and produce a better version of the product after several months.
- With DevOps, we can make any software product more efficient and increase the overall count of interested customers in your product.

DevOps Workflow

DevOps workflow provides a visual overview of the sequence in which input is provided. Also, it tells about which one action is performed, and output is generated for an operations process.



DevOps workflow allows the ability to separate and arrange the jobs which are top requested by the users. Also, it gives the ability to mirror their ideal process in the configuration jobs

DevOps Principles

The main principles of DevOps are Continuous delivery, automation, and fast reaction to the feedback.

1. End to End Responsibility: DevOps team need to provide performance support until they become the end of life. It enhances the responsibility and the quality of the products engineered.

2. Continuous Improvement: DevOps culture focuses on continuous improvement to minimize waste. It continuously speeds up the growth of products or services offered.

3. Automate Everything: Automation is an essential principle of the DevOps process. This is for software development and also for the entire infrastructure landscape.

4. Custom Centric Action: DevOps team must take customer-centric for that they should continuously invest in products and services.

5. Monitor and test everything: The DevOps team needs to have robust monitoring and testing procedures.

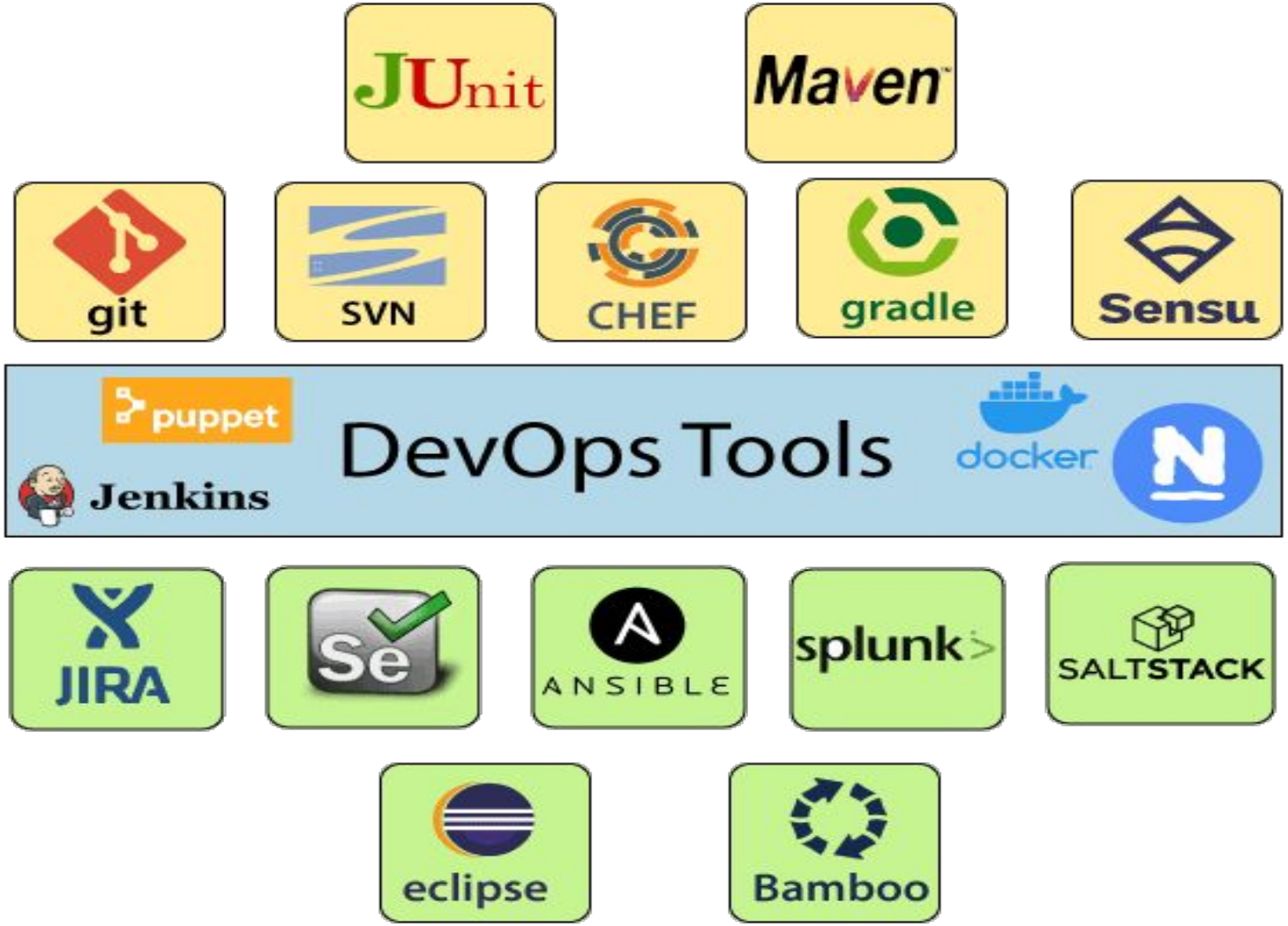
6. Work as one team: In the DevOps culture role of the designers, developers, and testers are already defined. All they needed to do is work as one team with complete collaboration.

DevOps Practices

Some identified DevOps practices are:

- **Self-service configuration**
- **Continuous build**
- **Continuous integration**
- **Continuous delivery**
- **Incremental testing**
- **Automated provisioning**
- **Automated release management**

DevOps Tools



DevOps Engineer Roles and Responsibilities

Below are some roles, responsibilities, and skills which are expected from DevOps engineers, such as:

- Manage projects effectively through an open standard based platform.
- Increases project visibility through traceability.
- Improve quality and reduce the development cost with collaboration.
- DevOps should have the soft skill of problem solver and a quick learner.
- Analyze, design, and evaluate automation scripts and systems.
- Able to perform system troubleshooting and problem-solving across the platform and application domains.
- Ensuring the critical resolution of system issues by using the best cloud security solution services.

DevOps Pipeline

- A pipeline in software engineering team is a set of automated processes which allows DevOps professionals and developer to reliably and efficiently compile, build, and deploy their code to their production compute platforms.
- The most common components of a pipeline in DevOps are build automation or continuous integration, test automation, and deployment automation.

A pipeline consists of a set of tools which are classified into the following categories such as:

- Source control
- Build tools
- Containerization
- Configuration management
- Monitoring

DevOps Methodology

- We have a demonstrated methodology that takes an approach to cloud adoption. It accounts for all the factors required for successful approval such as people, process, and technology, resulting in a focus on the following critical consideration:
- The Teams: Mission or project and cloud management.
- Connectivity: Public, on-premise, and hybrid cloud network access.
- Automation: Infrastructure as code, scripting the orchestration and deployment of resources.
- On-boarding Process: How the project gets started in the cloud.
- Project Environment: TEST, DEV, PROD (identical deployment, testing, and production).
- Shared Services: Common capabilities provided by the enterprise.
- Naming Conventions: Vital aspect to track resource utilization and billing.
- Defining Standards Role across the Teams: Permissions to access resources by job function.

DevOps Advantages and Disadvantages

Advantages

- DevOps is an excellent approach for quick development and deployment of applications.
- It responds faster to the market changes to improve business growth.
- DevOps escalate business profit by decreasing software delivery time and transportation costs.
- DevOps clears the descriptive process, which gives clarity on product development and delivery.
- It improves customer experience and satisfaction.
- DevOps simplifies collaboration and places all tools in the cloud for customers to access.
- DevOps means collective responsibility, which leads to better team engagement and productivity.

Disadvantages

- DevOps professional or expert's developers are less available.
- Developing with DevOps is so expensive.
- Adopting new DevOps technology into the industries is hard to manage in short time.
- Lack of DevOps knowledge can be a problem in the continuous integration of automation projects.

DevOps Engineers Salary

- The DevOps Engineers salary estimates are based on two reports of salaries, wages, bonuses, and hourly pay.
- Here is a list of DevOps engineers salary according to the most recent DevOps engineer salary report, such as:

Salary	Area	Experiences	Company
₹4,20,000	A DevOps engineer in the Bengaluru area reported making ₹4,20,000 per year.	0-year experience	Private
₹3,00,000	A DevOps engineer in the Bengaluru area reported making ₹3,00,000 per year.	1-2 year experience	Public
₹5,00,000	A DevOps engineer in the Hyderabad area reported making ₹5,00,000 per year.	3-4 year experiences	Public
₹3,00,000	A DevOps engineer in the Hyderabad area reported making ₹3,00,000 per year.	3-4 year experience	Private
₹6,00,000	An Azure DevOps engineer in the Chennai area reported making ₹6,00,000 per year.	1-2 year experience	Public
₹4,80,000	A DevOps engineer in the Pune area reported making ₹4,80,000 per year.	3-4 year experience	Public
₹11,06,561	A DevOps engineer in the New Delhi area reported making ₹11,06,561 per year.	3-4 year experience	Private

End of Module-1

THANK YOU !!!!