**SDLC**

**What is SDLC?**

The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software.

SDLC is the process of planning, developing, testing, and deploying software.

The SDLC methodology focuses on the following phases of software development:

* Requirement analysis
  + The business team (mostly comprised of business analysts) perform a requirement analysis of their project's business needs. The requirements can be internal to the organization, or external, from a customer.
  + The project cost gets decided, and benefits are laid out. Then the project goals are defined.
* Planning & Software design such as architectural design.
* The system architects and the system designers formulate the desired features of the software solution and create a project plan. This plan may include process diagrams, overall interface, and layout design, along with a vast set of documentation.

Software development

* + - The developers develop the code depending on the tasks and goals defined in the design phase. This phase may last from a few months to a year, depending on the project.
* Testing
* When all the decided features are developed, the testing team takes over. For the next few months, all features are thoroughly tested. Every module of the software is collected and tested. Defects are raised if any bugs or errors occur while testing. In the event of a failure, the development team quickly acts to resolve the failures. The thoroughly tested code is then deployed into the production environment
* Deployment
* Maintenance
* Feedback from the users/customers is analysed, and the whole cycle of developing, testing, and releasing the new features and fixes in the form of patches or upgrades repeats.

**SDLC Models**

* Waterfall model
* Agile model
* Spiral model
* V-model
* Incremental Approach

**Waterfall method**

* In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

**Agile Method**

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 stakeh**olders.**

|  |  |
| --- | --- |
| **Waterfall** | **Agile** |
| Development of the software flows sequentially from start point to end point. | It follows an incremental approach |
| Software development will be completed as one single project. | The agile process is broken into individual models that designers work on |
| The cust006Fmer can only see the product at the end of the project | The customer has early and frequent opportunities to look at the product and make decision and changes to the project |
| All sorts of project can be estimated and completed. | Small projects can be implemented very quickly. For large projects, it is difficult to estimate the development time. |
| Waterfall model are more secure because they are so plan oriented | Agile model is considered unstructured compared to the waterfall model |
| Only at the end, the whole product is tested. If the requirement error is found or any changes have to be made, the project has to start from the beginning | Error can be fixed in the middle of the project. |
| The development process is phased, and the phase is much bigger than iteration. Every phase ends with the detailed description of the next phase. | Development process is iterative, and the project is executed in short (2-4) weeks iterations. Planning is very less. |
| Documentation is a top priority and can even use for training staff and upgrade the software with another team | Documentation attends less priority than software development |
| Only after the development phase, the testing phase is executed because separate parts are not fully functional. | Every iteration has its own testing phase. It allows implementing regression testing every time new functions or logic are released. |
| Testers work separately from developers | Testers and developers work together |
| User acceptance is **performed** at the end of the project. | At the end of every sprint, user acceptance is performed |

**Agile Methodology**

* Agile is a process by which a team can manage a project by breaking it up into several stages and involving constant collaboration with stakeholders and continuous improvement and iteration at every stage. The Agile methodology begins with clients describing how the end product will be used and what problem it will solve. This clarifies the customer's expectations to the project team. Once the work begins, teams cycle through a process of planning, executing, and evaluating — which might just change the final deliverable to fit the customer's needs better. Continuous collaboration is key, both among team members and with project stakeholders, to make fully-informed decisions.

**Agile Advantages**

* Functionality can be developed and demonstrated rapidly
* Promotes teamwork and cross-training
* Suitable for projects where requirements frequently change
* Minimalistic documentation
* Parallel development

**Agile Has 4 Values**

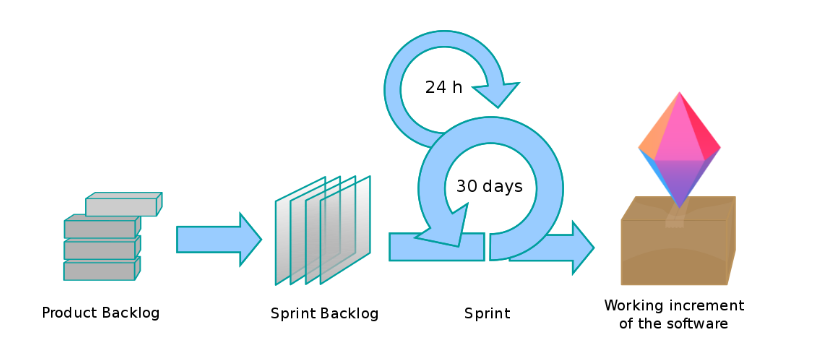
* Individuals and interactions over processes and tools
* Working software over comprehensive documentation
* Customer collaboration over contract negotiation
* Responding to change over following a plan

**Agile Principles**

* Customer satisfaction through early and continuous software delivery
* Accommodate changing requirements throughout the development process
* Frequent delivery of working software
* Collaboration between the business stakeholders and developers throughout the project
* Support, trust, and motivate the people involved
* Enable face-to-face interactions
* Working software is the primary measure of progress
* Agile processes to support a consistent development pace
* Attention to technical detail and design enhances agility
* Simplicity
* Self-organizing teams encourage great architectures, requirements, and designs
* Regular reflections on how to become more effective

**What is agile project management?**

* [Agile project management](https://zenkit.com/en/blog/agile-project-management-a-beginners-guide/) is a methodology that is commonly used to deliver complex projects due to its adaptiveness. It emphasizes collaboration, flexibility, continuous improvement, and high quality results. It aims to be clear and measurable by using six main “deliverables” to track progress and create the product.

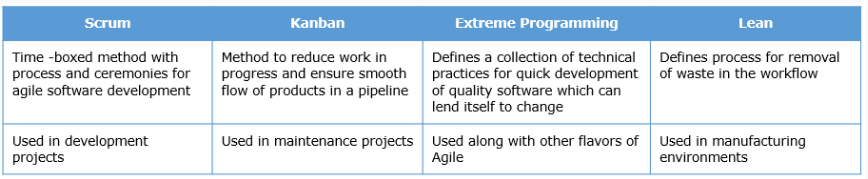


#### **The deliverables**

* Product vision statement: A summary that articulates the goals for the product.
* Product roadmap: The high-level view of the requirements needed to achieve the product vision.
* Product backlog: Ordered by priority, this is the full list of what is needed to be done to complete your project.
* Release plan: A timetable for the release of a working product.
* Sprint backlog: The user stories (requirements), goals, and tasks linked to the current sprint.
* Increment: The working product functionality that is presented to the stakeholders at the end of the sprint, and could potentially be given to the customer.

**Agile Methodologies Frameworks**

* Scrum
* Kanban
* Extreme Programing
* Lean

****

**Scrum Framework**

A PM methodology where a small team is led by a Scrum Master whose main job is to clear away all obstacles to completing work. Work is done in short cycles called sprints, but the team meets daily to discuss current tasks and roadblocks that need clearing.

Scrum is a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value.

Scrum is an agile project management methodology or framework used primarily for software development projects with the goal of delivering new software capability every 2-4 weeks. It is one of the approaches that influenced the [Agile Manifesto](http://agilemanifesto.org/), which articulates a set of values and principles to guide decisions on how to develop higher-quality software faster.

* Scrum Master
  + Master is responsible for setting up the team, sprint meeting and removes obstacles to progress
* Product owner
  + The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration
* Scrum Team
  + Team manages its own work and organizes the work to complete the sprint or cycle
* Product Backlog

This is a repository where requirements are tracked with details on the no of requirements (user stories) to be completed for each release. It should be maintained and prioritized by Product Owner, and it should be distributed to the scrum team. Team can also request for a new requirement addition or modification or deletion

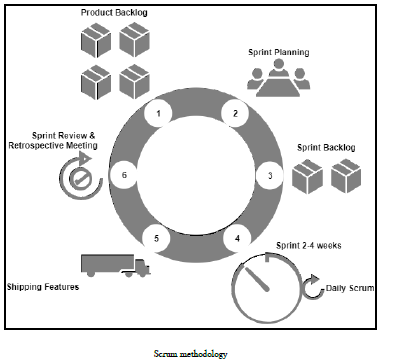
* Sprint: Iterative time boxes where a goal is accomplished. Time frame does not exceed one calendar month and are consistent throughout the development process.
* Sprint planning: Where the entire Scrum team get together — at the beginning of every Sprint — to plan the upcoming sprint.
* Daily Scrum: 15 minute time boxed meeting held at the same time, every day of the Sprint, where the previous day’s achievements are discussed, as well as the expectations for the following one.
* Sprint review: An informal meeting held at the end of every Sprint where the Scrum team present their Increment to the stakeholders, and discuss feedback.
* Sprint retrospective: A meeting where the Scrum team reflect on the proceedings of the previous Sprint and establish improvements for the next Sprint.

**How does Scrum work?**

The Product Owner, the Scrum Master, and the Scrum Team together follow a set of

stringent procedures to deliver the software features. The following diagram explains the

Scrum development process:



**Sprint Planning**

* Scrum Team to plan the features in the current Sprint cycle.
* The plan is created mainly by the developers. Once the plan is created, it is

explained to the Scrum Master and the Product Owner. The Sprint Planning is a timeboxed activity, and it is usually around eight hours in total for a one-month Sprint cycle

**Sprint cycle**

During the Sprint cycle, the developers simply work on completing the backlogs decided in the Sprint Planning. The duration of a Sprint may last from two weeks to one month, depending on the number of backlogs.

**Daily Scrum meeting**

This happens on a daily basis. During the Scrum meeting, the Development Team discusses what was accomplished yesterday, and what will be accomplished today. They also discuss the things that are stopping them from achieving their goal. The Development Team does not attend any other meeting or discussion apart from the Scrum meeting.

**Monitoring Sprint progress**

The Daily Scrum is a good opportunity for a team to measure its progress. The Scrum Team can track the total work remaining.

**Sprint Review**

In the Sprint Review, the Development Team demonstrates the features that have been accomplished. The Product Owner updates on the Product Backlog status to date. The Product Backlog list is updated depending on the product performance or usage in the market. Sprint Review is a four-hour activity altogether for a one-month Sprint.

**Sprint Retrospective**

In this meeting, the team discusses the things that went well, and the things that need improvement. The team then decides the points on which it has to improve to perform better in the upcoming Sprint. This meeting usually occurs after the Sprint Review and before the Sprint Planning.

**Continuous Integration**

Continuous Integration (CI) is a software development practice where developers

frequently integrate their work with the project's Integration branch and create a build. Integration is the act of submitting your private work (modified code) to the common work area (the potential software solution). This is technically done by merging your private work (personal branch) with the common work area (Integration branch). Or we can say, pushing your private branch to the remote branch.

CI is necessary to bring out issues encountered during the integration as early as possible. This can be understood from the following diagram, which depicts various issues encountered during a single CI cycle. A build failure can occur due to either an improper code or a human error while doing a build (assuming that the tasks are done manually). An integration issue can occur if the developers do not rebase their local copy of code frequently with the code on the Integration branch. A testing issue can occur if the code does not pass any of the unit or integration test cases.

**Kanban**

[Kanban](https://zenkit.com/en/blog/kanban-explained-what-youve-always-wanted-to-know/) is a highly visual method popularly used within agile project management. It paints a picture of the workflow process, with an aim to identify any bottlenecks early on in the process, so that a higher quality product or service is delivered.

A visual approach to project management where teams create physical representations of their tasks, often using sticky notes on whiteboards (or via online apps). Tasks are moved through predetermined stages to track progress and identify common roadblocks.

#### Its [six general practices](https://en.wikipedia.org/wiki/Kanban_%28development%29#Principles) are:

1. Visualization
2. Limiting work in progress
3. Flow management
4. Making policies explicit
5. Using feedback loops
6. Collaborative or experimental evolution

|  |  |  |
| --- | --- | --- |
| **Kanban** |  | **Scrum** |
| No prescribed roles | kanban-different-from-scrum-icon | Pre-defined roles of Scrum master, Product owner and team member |
| Continuous Delivery | kanban-different-from-scrum-icon | Timeboxed sprints |
| Work is pulled through the system (single piece flow) | kanban-different-from-scrum-icon | Work is pulled through the system in batches (the sprint backlog) |
| Changes can be made at any time | kanban-different-from-scrum-icon | No changes allowed mid-sprint |
| Cycle time | kanban-different-from-scrum-icon | Velocity |
| More appropriate in operational environments with a high degree of variability in priority | kanban-different-from-scrum-icon | More appropriate in situations where work can be prioritized in batches that can be left alone |

# **What is refactoring?**

* The process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure.
* Code Refactoring is the process of clarifying and simplifying the design of existing code, without changing its behavior. Agile teams are maintaining and extending their code a lot from iteration to iteration, and without continuous refactoring, this is hard to do. This is because un-refactored code tends to rot. Rot takes several forms: unhealthy dependencies between classes or packages, bad allocation of class responsibilities, way too many responsibilities per method or class, duplicate code, and many other varieties of confusion and clutter.
* Refactoring code ruthlessly prevents rot, keeping the code easy to maintain and extend
* But note that it is only "safe" to refactor the code this extensively if we have extensive unit test suites of the kind we get if we work Test-First
* If you are doing true [Test-Driven Development (TDD)](https://www.versionone.com/agile-101/agile-software-programming-best-practices/test-first-programming/), in which the design evolves continuously, then you have no choice about regular refactoring, since that's how you evolve the design.

# **How can refactoring be done?**

Modern day IDE's have features that help programmers to refactor code. There are many refactoring patterns that need to be understood by developers and with the help of tools, use it in code.

# Practical tips

* Buffer time for refactoring every sprint
* Do not refactor very close to release
* You may refactor code that is being newly developed rather than already existing code (legacy code)
* Sensitize customers on the need for time to be allocated for refactoring

# Additional References

Read: “Refactoring: Improving the Design of Existing Code” by Martin Fowler

# What is TDD?

Test Driven Development & test automation

Is a software engineering approach that involves short development iterations consists of writing failing test case(s) first covering a new functionality and then implementing the necessary code to pass the tests and finally refactor the code without changing external behavior.

# Need

It is required for the following reasons -

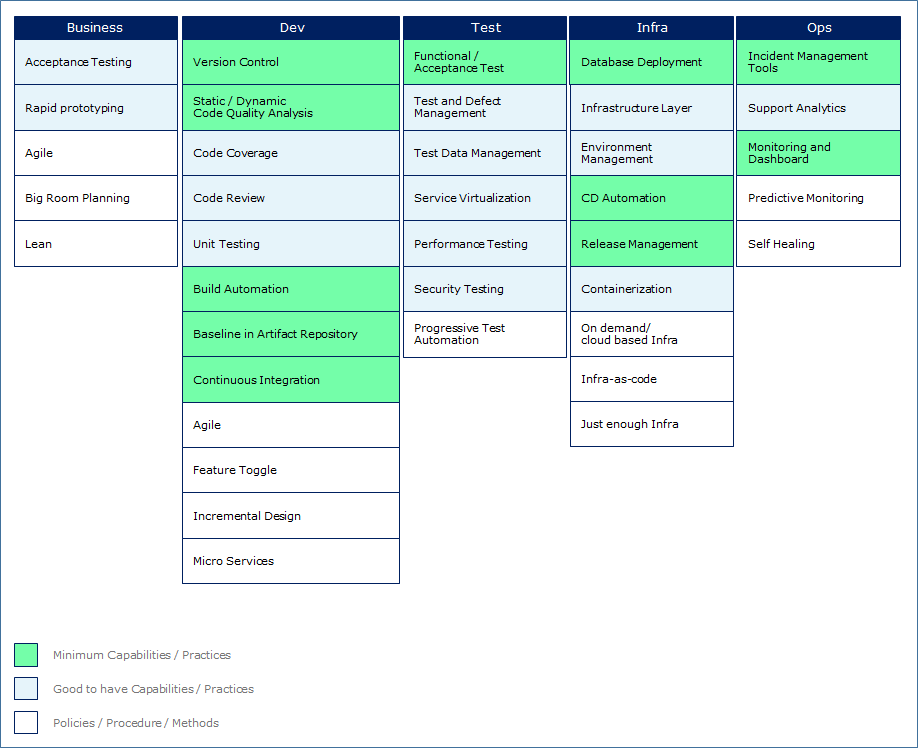
* Increased Code coverage
* Reduced duplication/redundancy
* Better consideration of reusability & design
* Modular and Stable Code

# What is pair programming?

Two programmers coding together on a single machine is called pair programming. One of the programmer plays the role of a driver who writes clean code, compiles and execute the code. The other plays the role of navigator who strategizes on design, tests, reviews and refactors code. These pairs self-organize and switch partners and roles during the day

# Need

Enables higher productivity and quality of work of the individuals doing pair programming than if done alone



# Policies/procedures/methods

**Agile**

* **Why?** - A rapid development approach like Agile helps in faster delivery of valuable software
* **What?** - Agile is a time boxed, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end
* **Benefits** - Rapid development of valuable software

**Big room planning**

* **Why?** - For effective planning and execution literally together in the same room real time
* **What?** - This brings all the stakeholders who are responsible for delivery of software (business, dev, test, program management teams) together in a single room for about two days
* **Benefits** - Improves communication between the teams and promotes a collaborative working relationships which is the basis for Agile

**Lean**

* **Why?** - Elimination of waste and bottlenecks in automated software development and delivery
* **What?** - This is achieved by collaboration, by ‘shifting left’ operational concerns early in the development lifecycle, by eliminating waste, rework and over-production i.e. using Lean principles in DevOps
* **Benefits** - Teams eliminate the bottlenecks in the DevOps pipeline, making it more efficient and productive

# Minimum capabilities/Practices

**Automated acceptance tests**

* **Why?** - Acceptance tests help identifying building the right code
* **What?**- These are meant for business teams to check if the functionality is working for the user. These can be automated using tools like Fitnesse
* **Benefits** - Brings transparency and speed to delivery of software

# Good to have capabilities/Practices

**Rapid prototyping**

* **Why?**- For rapid and iterative design and delivery of software
* **What?**- It involves using prototyping tools and collaborative review by stakeholders and refinement based on feedback
* **Benefits** - Provides a feel of the product early, room for customization, saves cost and time (if tools are used) and minimizes design flaws

# Role and responsibilities

**Business teams should**

* be ready to adopt Agile approach and collaborate with the development teams for faster development and delivery of valuable software
* adopt lean practices and eliminate wasteful processes, documentation and the like
* be ready to allow iterations to happen and provide early feedback
* Ensure availability and clarity on requirements are on time

If business teams align with the capabilities mentioned, the following issues faced by “Pura vida” would get alleviated.

**Alleviated issues**

* Lack of automation in the software development lifecycle and hence loss of quality due to error prone repeatability of steps – in this case automation of Acceptance Tests
* Well defined  and automated acceptance criteria and clarity on requirements

DEV TEAM

# Policies/procedures/methods

**Agile**

* **Why?**- A rapid development approach like Agile helps in faster delivery of valuable software
* **What?** - Agile is a time boxed, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end
* **Benefits** - Early feedback, work division into small units, transparency in the process – all these motivate teams and improve their productivity

**Feature toggle**

* **Why?**- When new features /enhancements need to be released to all/chosen user base into a live system without needing to change the code, feature toggling is used
* **What?**- Let us take an example where there are four teams A-D working on a release. Team A is working on a feature that will take considerable time to develop and test. Instead of team members B, C and D waiting for A to complete, they introduce a feature toggle in the code

**Types of toggles**

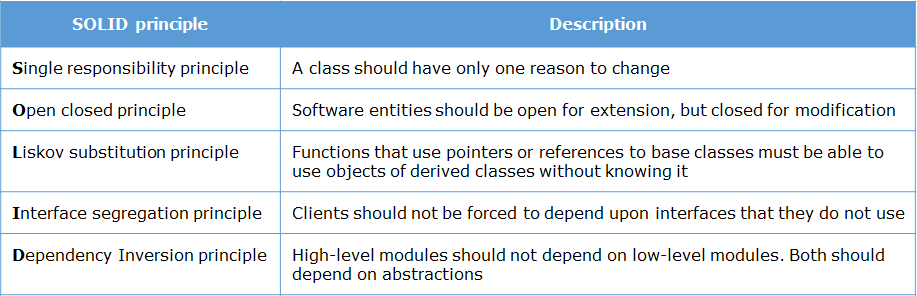
* release toggles
* experiment toggles
* Ops toggles
* Permission toggles

**Benefits of toggle**

* release features to a selected cohort of users to get their responses prior to release to the entire user base
* provide permissions to access certain features to a selected user base

**Incremental design**

* **Why?**- We need to design solutions that can lend itself to change as the code adapts to changing requirements
* **What?**- Incremental design is based on SOLID principles. S.O.L.I.D is an acronym for the first five object-oriented design(OOD) principles by Robert C. Martin



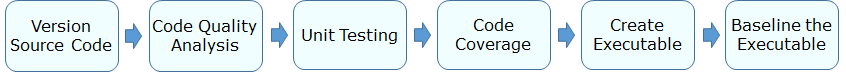
**Micro services**

* **Why?**- Traditionally, enterprise Applications are often built in three main parts. The server side application becomes a “monolith”, a single logical executable. Any changes to the system will involve building and deploying this monolith. In fully automated deployable environments, it becomes necessary to develop small deployable components instead of one single process which then ties up the change cycles together.
* **What?**- It is an architectural style to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery
* **Benefits**
  + Improves isolation of faults. Large application may not be very much affected by failure of a single module
  + Commitment to a single stack of tools can be eliminated. Dependencies may be lighter compared to monolithic applications
  + Very useful in fully automated deployment pipeline for faster delivery of components

# C:\Users\VENKTESH\Desktop\MicroFunction.png

**Continuous integration pipeline**

* **Why?**-Continuous integration practice is mandatory to ensure that defects are captured early and a working version of the software is always available by automating the different development stage through integration of tools. This is a must-to-have to ensure continuous, frequent and automated delivery & deployment of software to customers
* **What?**- Is a software development practice adopted as part of extreme programming (XP). The image below shows the activities in the development phase that need to be automated to form the Continuous Integration pipeline. It helps in automating the build process, enabling frequent integration, code quality checks and unit testing without any manual intervention by use of various open-source, custom-built and/or licensed tools. The sequence and whether an activity will be done will be decided by the orchestrator – i.e. the continuous integration tool based on the gating criteria set for the project



* **Benefits**
  + Reduced risks and fewer integration defects
  + Helps detect bugs and remove them faster
  + Less integration time due to automation
  + Avoids cumulative bugs due to frequent integration
  + Helps in frequent deployment

Note: The details of these activities and tools involved will be discussed in the subsequent sections.

If Dev teams align with the capabilities mentioned, the following issues faced by “Pura vida” would get alleviated.

**Issues**

* Lack of automation in the software development lifecycle and hence loss of quality due to error prone repeatability of steps – here the automation of activities in the build cycle activities
* Brittle point-to-point integration between modules – resolved due to continuous integration of modules

TESTING TEAM

**Progressive test automation**

* **Why?**- In order to expedite and automate the testing right from the beginning and to ensure consistent delivery of valuable software, progressive testing is required. It also helps to detect bugs early and quickly
* **What?** - It is one of the automation methods where the test modules are tested in various stages of software development. The test code is written simultaneously with development code. Helps test new and evolving functionality suggested by customers and hence has gained significance in software development using an Agile Approach. Testing teams are involved right from the beginning. They write end-end automated test cases that are run continuously. Mocking and service virtualization is used when the required components are not available (more on this in the later section). Some service providers provide end-end automated testing services
* **Benefits**
  + Expedites the testing process, helps detect and fix bugs early
  + Increased testing coverage and shorter testing cycle time
  + Testers are involved right from the beginning and hence the test cases are close to real world and requirements of end users

**Continuous and automated testing**

* **Why?**- In an automated development and delivery pipeline, integration will be done frequently as we saw in the previous section. Hence test cases need to be run frequently. The gate for developed software meeting the functionality are the test cases. This is possible only if the tests are automated
* **What?**- Involves using automated testing tools which can execute tests, provide outcomes of the tests in the form of reports and can be run repeatedly. Test automation frameworks helps in this. Ex. JUnit

In the automated development and delivery pipeline, the following tests and their management are automated mandatorily. These tests are invoked in an automated fashion by the orchestrator – i.e. the continuous integration tool. The testing related automation is required.

**Test and test data management automation**

* Functional test
* Test and Test data management
* Performance test
* Security test
* **Benefits**
  + Increases the depth and scope of test coverage which helps in improving software quality
  + Ensures repeatability of running tests whenever required and helps in continuous integration of valuable software

**Service virtualization**

* **Why?** - In traditional software development, the testing starts post the integration of all the components that are needed – ex. Performance testing is delayed and testers may skip this for want of time. This leads to finding of defects late in the cycle and are costly to fix. It also impacts the delivery speed. Hence service virtualization is required to “shift-left” and detect defects early in the cycle (say during unit testing) by simulating the non-available components of the application
* **What?**- Service virtualization helps in simulating application dependencies and begin testing earlier. Virtual components can be data, business rules, I/O configurations etc
  + **Features**
    - Are light-weight and hence testing is inexpensive. Example: If we have a legacy system on top of which business logic enhancements are done, setting up the latter every time for testing is cumbersome and costly
    - Creates a virtual asset which simulates the behaviour of the components which are impossible to access or unavailable
    - Components can be deployed in virtual environments
    - Created by recording of the live communication among the components that are used for testing
    - Provide logs representing the communication among the components
    - Analysis of service interface specifications like WSDL
    - Assets listen for requests and return an appropriate response. Example: Listen to an SQL statement and return the source rows from the database as per the query
* **Benefits**
  + Reduce cost of fixing defects
  + Decreases project risk
  + Improves the speed of delivery
  + Helps emulate the unavailable components/environments and represents a more realistic behaviour (stubs/mocks help in skipping unavailable components)

If test teams align with the capabilities mentioned, the following issues faced by “Pura vida” would get alleviated

* Defects generated due to inconsistent environments for testing and deployment – here testing need for all the components to be made available, simulation helps remove defects early and testing is done in a “close-to-real” environment

**INFRA TEAM**

**n- demand infrastructure**

* **Why?** - There are fluctuating demands for infrastructure due to the changing demands and requirements. For example, in an automated software development and delivery pipeline, testing environments will be required on a continuous basis for short periods of time. Maintaining the environments to service peak requirements on a continuous basis may be costly for organizations, while maintaining minimal resources will lead to delay in delivery. Hence the shift towards on-demand infrastructure is prevalent in the industry today with HP,IBM,Microsoft and Sun-microsystems being the prominent vendors
* **What?** - This is a model where computing resources are made available to the user as needed. The resources may be maintained within the user's enterprise, or made available by a service provider. This is implemented through virtualization and cloud or dynamic allocation of VMs
* **Benefits**
  + Infrastructure is provided when required
  + Minimises the capex and revex tied up in IT infrastructure
  + Reduced carbon footprint and maintenance costs of infrastructure

**Infra-as-code**

* **Why?** - Configuration / re-configuration of servers done frequently (due to fast and frequent releases) and manually (with the help of scripts) is a time consuming and tedious process with lots of scope for errors
* **What?** - It is also called programmable infrastructure which involves writing code using a high level or descriptive language to manage configurations and also provision infrastructure and deployments in an automated fashion. This uses proven software development practices which are used to write code for application development. It is different from infrastructure automation where the steps for configuration of servers is repeated
* **Benefits**
  + Developers can also engage in writing code for infrastructure provisioning, deployment and configuration
  + Development and testing is faster and simpler which further aids in speed of deployment and delivery
  + Helps avoid “snowflake servers” (servers that are difficult to reproduce due to its complex configurations). Once the configuration is automated (using infra-as-code ), it can be used by anyone to create servers of same configuration. This will also ensure consistency in development, testing and deployment environments
  + Since standard development practices like version control is followed, it is easier to maintain the changes made to the environments

**Just enough infra**

* **Why?** - Often, software quality suffers due to lack of proper testing. The latter may be due to the fact that enough infrastructure is not available when required. Hence on-demand, just-enough infrastructure is required so that environment can be made available for testing and deployment and released when not needed
* **What?** - Involves provisioning infrastructure from a cloud or a virtual setup when and as much required. May be provided on demand (just enough infra) by Infra structure as services(IAAS) by vendors
* **Benefits**
  + Cost-effective to host and test applications
  + Aids in fast deliver, testing and deployments

**CD automation**

* **Why?** - Business requires frequent delivery of valuable software with efficiency. Continuous delivery helps create a repeatable, reliable and incrementally improving process for ensuring this
* **What?** - CD allows constant flow of changes into production through an automated software production pipeline called "CD Pipeline". This involves Continuous Validation(CV) followed by Continuous Delivery. Quality is built-in to the pipeline. The pipeline provides feedback to the team and visibility into the flow of changes to everyone involved in delivering the new feature/s. We have seen these CI and CV earlier. CD therefore is a series of practices to ensure that quality code can be deployed fast and safe to production by delivering every change/new feature to a production-like environment. Since automation is used, the confidence level that this would work well in production environment is high. With the push of a button, the change can be deployed to the production environment. This is called continuous deployment. Continuous deployment may not be practical in all organizations due to regulatory and other processes, though it should be the goal of every organization. It follows continuous delivery
* **Benefits**
  + Customers can realize early ROI
  + Since it is based on automation, repeatability is ensured and quality software is delivered to pre-production environment which ensures that the same will work well in production environment also

**Release management**

* **Why?** - Traditionally, release management involves the complex process of planning, designing, building, testing and deploying new software and hardware in production environment. Integrity needs to be maintained while releasing the correct version. Traditionally, this process is very stressful and inefficient involving a lot of manual work and co-ordination. Also, since there is isolation of ops and dev teams, hence there are surprises, delays and errors in releases. There is lot of documentation that needs to be read prior to deployment every time.  A path and workflow needs to be done through a system to allow for fast delivery of software
* **What?** - The automated tools for release management allow the integration of the management and execution of releases. They help teams to plan, track and execute the releases through an integrated interface. They allow the approvals and notification to the concerned for various stages in the delivery pipeline. Release plans can therefore be run quickly
* **Benefits**
  + Errors in releases can be reduced
  + The workflow is automated and can be tracked through a system
  + Helps in bringing speed to releases

**Database deploy**

* **Why?** - In the continuous delivery pipeline, since database deployment has fundamental differences with application deployment and processes, the former is done in a manual fashion. Hence the benefits of continuous delivery pipelines are not optimized and may result in delays.  To standardize database deployments to the CD delivery practices, automated database deployments are required
* **What?** - The automated database deployment tools generate a single deployment script that contains the meta data and structure changes. It also contains the details of the changes in terms of configuration management
* **Benefits**
  + High levels of visibility into database deployments
  + Prevents errors due to manual scripts for database deployments
  + Provides an interface to package, verify, deploy, and promote database changes as done with application code and integration of database deployments to the CD pipeline

# Infrastructure layer and environment management

* **Why?** - The various steps that we saw earlier (including environment provisioning, testing, deployments etc.) need an infrastructure to be in place for implementation. This layer is responsible for infrastructure management. The environment management tools help in this
* **What?** - The infrastructure layer can be managed by tools like Chef which help spinning of virtual machines, syncing them, help make changes across multiple servers etc. The virtual machines mimic servers including the complete operating system, drivers, binaries etc. They run on top of a hypervisor system which in turn runs on top of another operating system
* **Benefits** - Provides the necessary hardware for automated deployments and the environment management tools help in managing and maintaining them

**Containerization**

* **Why?** - A virtual machine as seen in the earlier section has its own operating system. Hence precious operating system resources are wasted across virtual machines. In order to ensure that the virtual machines share the same resources, containerization is required
* **What?** - It allows virtual machines to share a single host operating system and relevant binaries, drivers etc. This is called operating system level virtualization
* **Benefits**
  + Containers are smaller in size, easier to migrate and requires less memory
  + Allows a server to host multiple containers instead of virtual machines being spun

**OPS TEAM**

**Predictive monitoring**

* **Why?**- In case of automated deployment and delivery it becomes important to help support teams predict an issue before it arises. This is why predictive monitoring is required
* **What?** - This is done through predictive monitoring tools which analyze various elements of the IT environment in a way that enables the IT teams to predict issues before it turns into a full-fledged issue and disrupts the services
* **Benefits** - Will help support teams identify potential issues proactively before it results in disruption of services

**Self-healing**

* **Why?** - Systems that are created are not perfect. Services may fail due to increase in load or induced bugs. Making systems resilient to recuperate from failures and predict one in the near future is required and hence self-healing systems need to be developed
* **What?** - This involves making the system take decisions based on continuous checking and optimization of its state and adapt to changing conditions. This creates a responsive system that is capable of responding to changes and recuperate from failures. Self-healing systems can be divided into three levels - Application level, System level and Hardware level
* **Benefits** - Helps support teams by monitoring and healing issues before they disrupt services and provide a way to heal themselves

**ncident management**

* **Why?** - With the large scale explosion of data centers and virtualization, the scale and fragmentation of IT alerts have increased dramatically. Hence the manual way of resolving alerts like constantly filtering through noisy alerts, connect them to get the bigger issue, prioritize and escalate to concerned, and manually managing the alerts should be avoided
* **What?** - Centralized incident management solutions avoids redundant alerts. It combines all the monitoring systems and provides an easy tracking mechanism by which support teams can respond
* **Benefits**
  + Helps support teams to respond to alerts quick and easy
  + Since the automated pipelines may have several tools and layers, incident management tools help centralize the alerts and hence faster responses to them

**Support analytics**

* **Why?** - Faster release cycles demand automated deployment to get applications out faster and they demand discovering and diagnosing production issues gaining insight quickly and through actionable analytics. Focusing on business metrics is important in DevOps environments. To derive these metrics and the data to meet the key performance indicators becomes essential and hence the need for support analytics
* **What?** - Tools for support analytics do a deep search for data, do centralized logging and parsing and display the data in a neat way
* **Benefits** - These tools help in collaboration across teams and provide exactly what is happening to the business from the data that is stored and logged

**Monitoring dashboard**

* **Why?** - In order to measure the success of DevOps adoption and also measure the health of the pipelines, monitoring dashboards are required
* **What?** - A dashboard provides a complete view of the pipeline. The dashboard can be based on different perspectives. Some examples -
  + Business performance dashboard – May depict the revenue, speed of deployment, defect status etc. This can be for both technical and non-technical teams
  + End user dashboard – may provide code and API specific metrics like error rates, pipeline status etc.
* **Benefits** - Single point where teams get visibility of the DevOps implementation

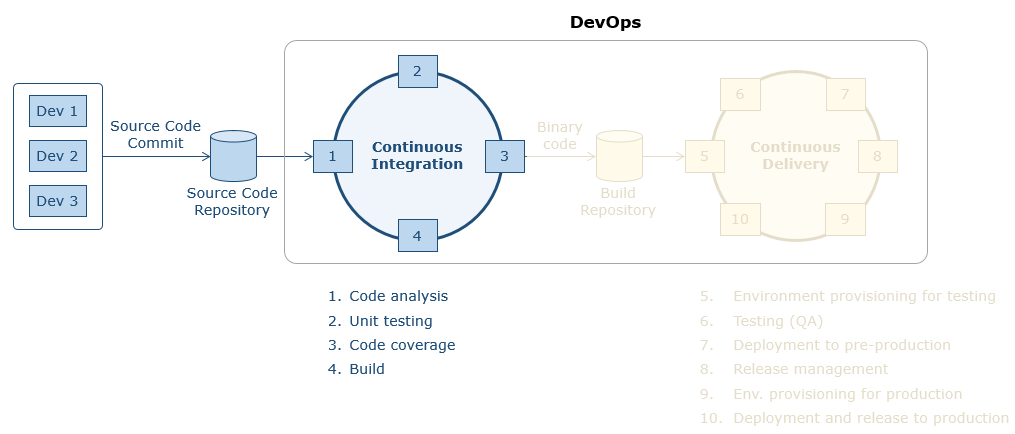
# DevOps

* a set of capabilities, across the IT Value Stream (plan – build – run) which enhances throughput, quality & business value
* relevance of capabilities needs to be considered along with the ROI, while deciding upon the DevOps blueprint
* Eco-system comprises a set of capabilities / tools / automation and will evolve continuously and needs to be governed
* ways of working (operating model & associated processes) have to be suitably adapted, to overcome functional silos and complexities posed by a multi-vendor environment

# What is Continuous Integration (CI) ?

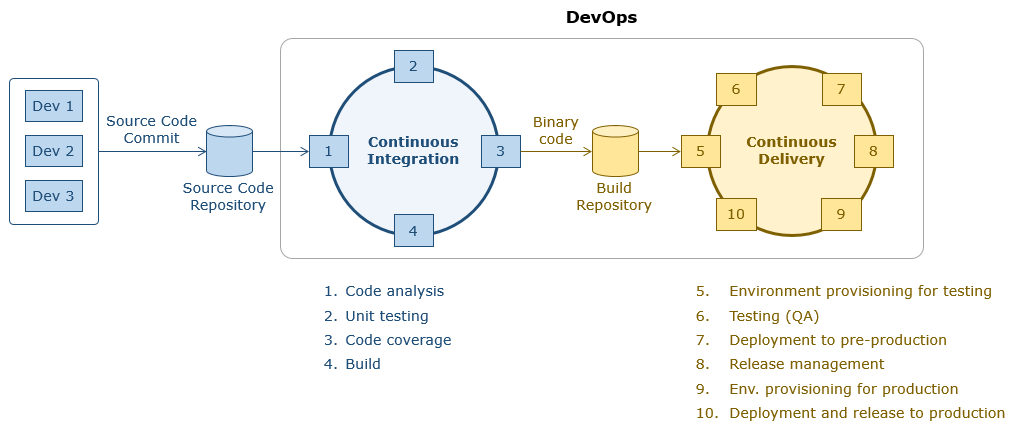
A software development practice that requires members of the team to integrate their work frequently. Integration is done at least daily. It leads to multiple integrations each day.

The workflow for CI is as given below.



The developers(ex. Dev1 and 2 shown in figure) commit code to a source code repository. Post this the build phase activities of static code analysis, unit testing, code coverage by the unit tests and building the executable file are automated through tools and these steps are called in sequence by the continuous integration server. The executable file is placed in the artifact repository for further testing and release. The continuous integration server is configured to orchestrate frequently to ensure continuous integration of code.

Devops



CICD Pipeline



vzxvbx