The Software Development Process

Software Development – Implementation

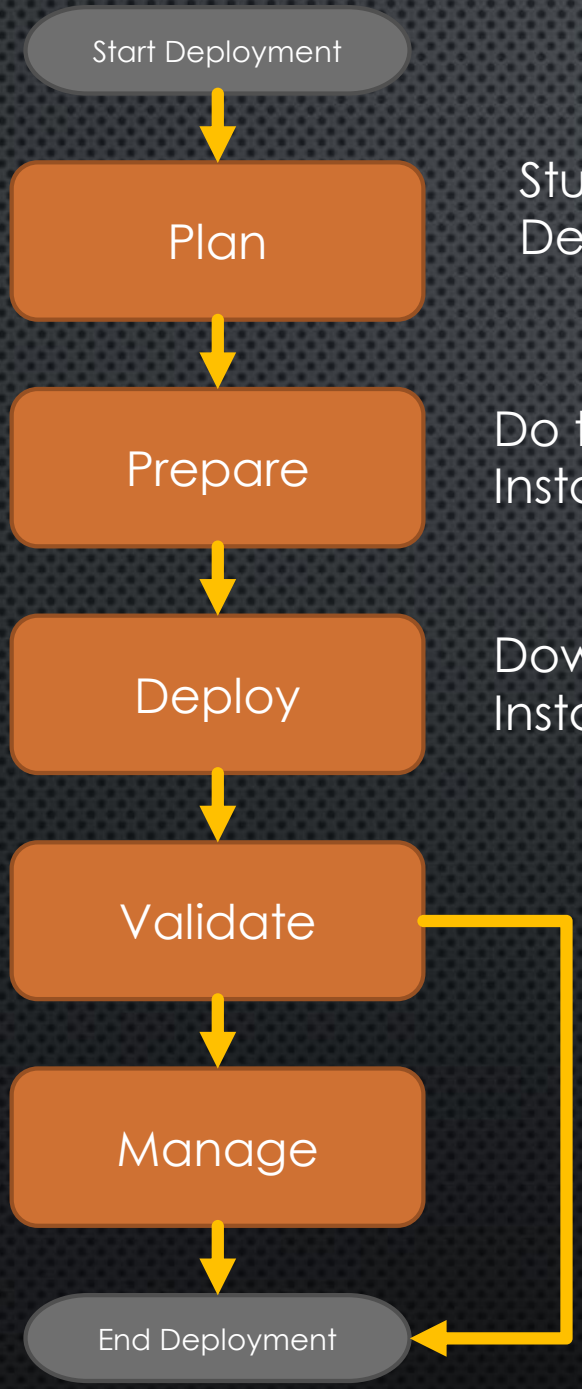
The Deployment Process

THE DEPLOYMENT PROCESS - DEPLOYMENT

- Software deployment refers to all the activities aimed to make a software system available for use
- For use it is intended by testers, final users, developers, customers, etc.
- The deployment refers to make a system available no matter which environment (production, testing, development,…)

THE DEPLOYMENT PROCESS

- The deployment process is like a recipe for deploying a software
- You define the recipe by adding steps and variables to a project
- Each step contains a specific action (or set of actions) that is executed as part of the deployment process each time your software is deployed
- After the initial setup, your deployment process shouldn't change between deployments even though the software being deployed will change as part of the development process
- The process depends on your organization and on the type of software solution you are delivering



Study the environment
Design the architecture and zone structure

Do the configurations
Install deployment suites

Download software package
Install the package

Verify credentials and user rights
Verify the main functionalities with a deployment test case

Add roles
Add group polices
Add users

AGILE SOFTWARE DEPLOYMENT

- In Agile project the critical phase is the deployment phase
- This is a phase quite often neglected because many are not confident with deployment which in Agile is rapid deployment
- Rapid deployment, one of agile's greatest strengths, can also be a major weakness if testing and automation are not properly developed
- Moving too fast without the right processes in place can lead to errors, downtime, and poor user experience
- Agile deployment must be done correctly for the success of a project

AGILE SOFTWARE DEPLOYMENT

- A correct deployment process will make agile faster and leads to fewer failures and quicker recovery time
- That's why so many companies today have adopted agile development or DevOps
- If you're trying to improve your deployment process, consider the 8 best practices of agile deployment
- Keep in mind that every company and every application is different
- Use these best practices as a guide to developing a unique deployment process for your team

Use a deployment checklist

A deployment checklist reminds you to complete critical tasks both before and after deployment
A checklist must suit your team's needs and project

Use a deployment checklist

Choose the right deployment tools

Deployment at the speed of DevOps is made possible by a set of tools that let you automate key stages of the software development life cycle

The set of tools to use depends on the type of project and team

Generally, you want tools that work natively with your application infrastructure and integrate with your other tools

Jenkins, a continuous integration/continuous delivery server, is popular because the open-source platform supports thousands of plugins with other tools

Use a deployment checklist

Choose the right deployment tools

Use a continuous integration server

One of the most important tools for successful agile deployment is a continuous integration (CI) server
CI servers pull in the source code from all developers and test it together in real time
This prevent the use of local PC to host the code, not together in the main branch
CI servers are sometimes called “build servers” because they take the source code and package it into an application artefact (this is called “building”)
Continuous integration also incorporates principles of continuous testing, where teams constantly collect feedback in order to catch problems as soon as possible

- Use a deployment checklist
- Choose the right deployment tools
- Use a continuous integration server
- Adopt continuous delivery

Continuous integration and continuous delivery (CD) are often mentioned together, but they are two different practices
Once the code is integrated and the application is built, CD involves packaging and preparing the code for deployment
The application is put in a pre-production environment, or a replica of the actual production server
Rigorous testing takes place during the delivery phase to ensure the application will work once deployed
The goal of continuous delivery is to have applications that are always ready to deploy
Not only does this speed up software deployment, it has also been proven to produce higher quality software
Often, CI and CD can be automated together with tools like **Jenkins**, or you can find two tools that support each other, like **TeamCity** and **Octopus Deploy**

continuous delivery

Develop

Integrate

Deploy

Deliver

- Use a deployment checklist
- Choose the right deployment tools
- Use a continuous integration server
- Adopt continuous delivery
- Automate your deployment process

If you're manually deploying new versions of your software, you're doing it wrong
Deployment is a complex process, and doing it by hand leaves too much room for human error
Deployment automation reduces errors, speeds up deployments, and makes the process so easy, practically anyone on your team can do it
The simplest form of deployment automation is the use of scripts to deploy specific actions in a specific environment in a specific context
For more advanced automation, use any one of the software deployment tools on the market today
Many CI/CD tools also support automated deployment

- Use a deployment checklist
- Choose the right deployment tools
- Use a continuous integration server
- Adopt continuous delivery
- Automate your deployment process
- Is continuous deployment right for you?

Continuous deployment takes continuous integration and continuous deployment a step further
It involves automatically deploying new versions of the software to customers, typically every time a developer makes a change
For a change to be automatically deployed, it would have to pass all tests in the CI and CD stages
Continuous deployment requires advanced testing automation and an acceptance of bugs in the production version
Continuous deployment benefits from incremental rollouts, which can be rolled back if an error occurs
Continuous deployment isn't for every team
While it speeds up your release rate, it requires high upfront investment in CI and CD testing to ensure deployments are as error-free as possible

- Use a deployment checklist
- Choose the right deployment tools
- Use a continuous integration server
- Adopt continuous delivery
- Automate your deployment process
- Is continuous deployment right for you?
- Monitor your KPIs

Once you deploy a new version of your software, keep an eye on your key metrics

Common deployment KPIs include server utilization, exception rates, log volume, and database performance

If there's a performance issue, it's also important to know where the issue is coming from

APM tools like Retrace let you monitor all of your KPIs and track deployment so you know the source of problems when they arise

- Use a deployment checklist
- Choose the right deployment tools
- Use a continuous integration server
- Adopt continuous delivery
- Automate your deployment process
- Is continuous deployment right for you?
- Monitor your KPIs
- Have a rollback strategy

You want to be able to deploy with confidence, and that means always having a rollback strategy in case something goes wrong
When an error occurs upon deploying a new version, sometimes the best solution is to switch back to the last known working version of the software
There are release automation tools like **Automic** that support rollbacks, or you can keep a backup copy of your latest version until you know the new one is working properly

REGRESSION TESTING

- During a deployment changes are introduced into your system
- When any modification or changes are done to the application, you can encounter unexpected issues
- Along with the new changes it becomes very important to test whether the existing functionalities are behaving as implemented previously
- This can be achieved by doing the **regression testing**
- The purpose of the regression testing is to find the bugs which may get introduced accidentally because of the new changes or modification

REGRESSION TESTING

- A change or even a fix to a previous bug may have affected another functionality or another part of the software code
- Regression testing is the way to detect these ‘unexpected side-effects’ of changes/fixes
- Usually the regression testing is done by automation tools
- While testing the new change or fixing the defect the same test is requested to be executed again and again and it will be very tedious and time consuming to do it manually

REGRESSION TESTING

- During regression testing the test cases are prioritized depending upon the changes and/or fixes
- The feature or module where the changes or modification is done that entire feature is taken into priority for testing
- These changes or enhancements should NOT introduce new issues in the existing tested code
- This helps in maintaining the quality of the product along with the new changes in the application

Example

- Let's assume that there is an application which maintains the details of a set of data, e.g. sales orders or users
- This application has three buttons: Add, Save, Delete
- All the buttons functionalities are working as expected
- Recently a new button '**Update**' is added in the application
- This 'Update' button functionality is tested and confirmed that it's working as expected
- But at the same time it becomes very important to know that the introduction of this new button should not impact the other existing buttons functionality
- Along with the 'Update' button all the other buttons functionality are tested in order to find any new issues in the existing code
- This process is known as **regression testing**

TYPES OF REGRESSION TESTING TECHNIQUES

We have four types of regression testing techniques
They are as follows:

Corrective Regression Testing	Corrective regression testing can be used when there is no change in the specifications and test cases can be reused
Progressive Regression Testing	Progressive regression testing is used when the modifications are done in the specifications and new test cases are designed
Retest-All Strategy	The retest-all strategy is very tedious and time consuming because here we reuse all tests which results in the execution of unnecessary test cases When any small modification or change is done to the application then this strategy is not useful

Selective Strategy

In selective strategy we use a subset of the existing test cases to cut down the retesting effort and cost
If any changes are done to the program entities, e.g. functions, variables etc., then a test unit must be rerun
Here the difficult part is to find out the dependencies between a test case and the program entities it covers

When to use it:

- Any new feature is added
- Any enhancement is done
- Any bug is fixed
- Any performance related issue is fixed

REGRESSION TESTING

- For regression testing to be effective, it needs to be seen as one part of a comprehensive testing methodology
- It should cover all the aspects e.g. frontend, to prevent any aspects of your software applications from going unchecked
- Many Agile work environments have regression testing as an essential aspect of a dynamic, iterative development and deployment schedule
- Automated regression testing can help prevent projects from going over budget, keep your team on track, and, most importantly, prevent unexpected bugs

HOW IS REGRESSION TESTING BEST IMPLEMENTED?

- **Retest All**
  - This method of regression testing simply re-tests the entirety of the software, from top to bottom
  - In many cases, the majority of these tests are performed by automated tools, but often that is neither feasible nor necessary
  - Moreover, purely using automation ignores the benefits of human testers and any chance for exploratory testing
  - Crowd testing provides the innate ability to run test cases in parallel, via numerous testers across many devices and platforms

HOW IS REGRESSION TESTING BEST IMPLEMENTED?

- **Regression Test Selection**
  - Rather than a full re-test process, this method allows the team to choose a representative selection of tests that will approximate a full testing of the test suite, but require far less time or cost to do so
- **Test Case Prioritization**
  - With a set of limited test cases, it is ideal to prioritize those tests
  - Try to prioritize tests which could impact both current and future builds of the software

REGRESSION TESTING – BEST PRACTICES

Maintain a Schedule	Choose a schedule of testing you can maintain throughout the software development life cycle, so testing is never placed on the back burner
Use a Test Management Tool	In order to properly track all the tests that are being performed on a regular basis, and have historical records of their performance over time, it's ideal to use a simple test management tool
Evaluate Test Prioritization	When using any form of prioritization to order your regression tests, try to find a sensible way to order them
Break Down and Categorize Tests	Just as with the refactoring of the code base itself, tests are often easier to understand and evaluate if they are broken down into smaller pieces Try to refactor your tests as often as necessary, then categorize them such that categories of smaller tests can be prioritized over others, making the sorting and execution much easier in the future

Consider Customer Risks

When developing and prioritizing regression tests, keep vigilant about considering the effects the test cases will have on the customer, or the business as a whole
Try to design regression tests that cover as many customer-oriented test cases as possible

Perform Exploratory Testing

Regression testing periods prior to a release to customers should also allow for an exploratory testing phase, in order to ensure not only that tests pass, but that the software functions as a user would expect, through in-depth examinations performed by professional testers