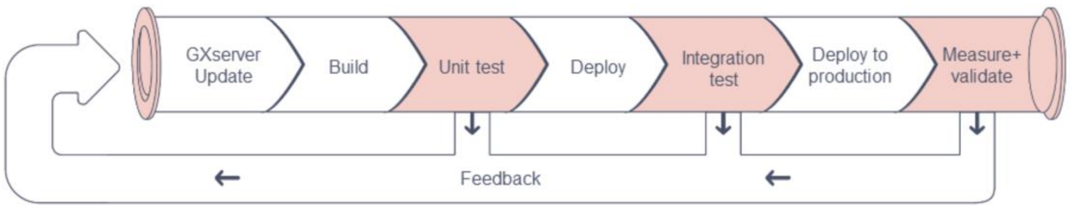
GeneXus™

by Globant

# Introduction to Pipelines

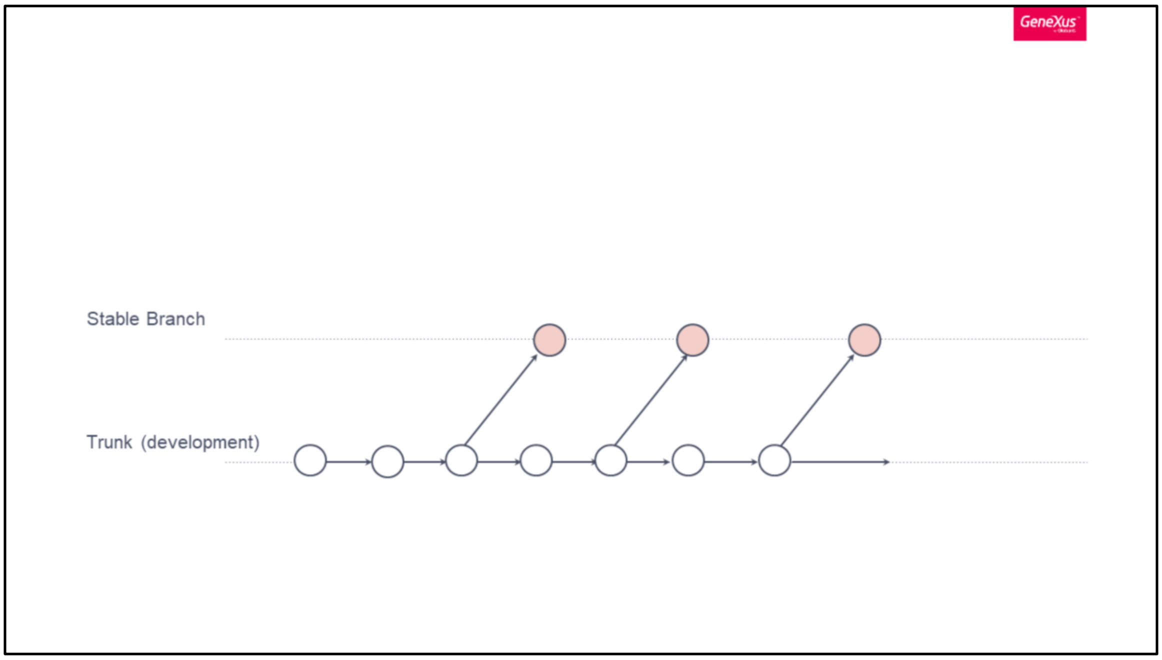
GeneXus by Globant



The goal of the pipeline is to automate the development process and validate frequently the last application version. In case issues are detected with automated tests, a quickly feedback is generated to the development team in each stage of the process.

Creating pipelines you will increase productivity and release software more frequently with more quality.

## Trunk-based development



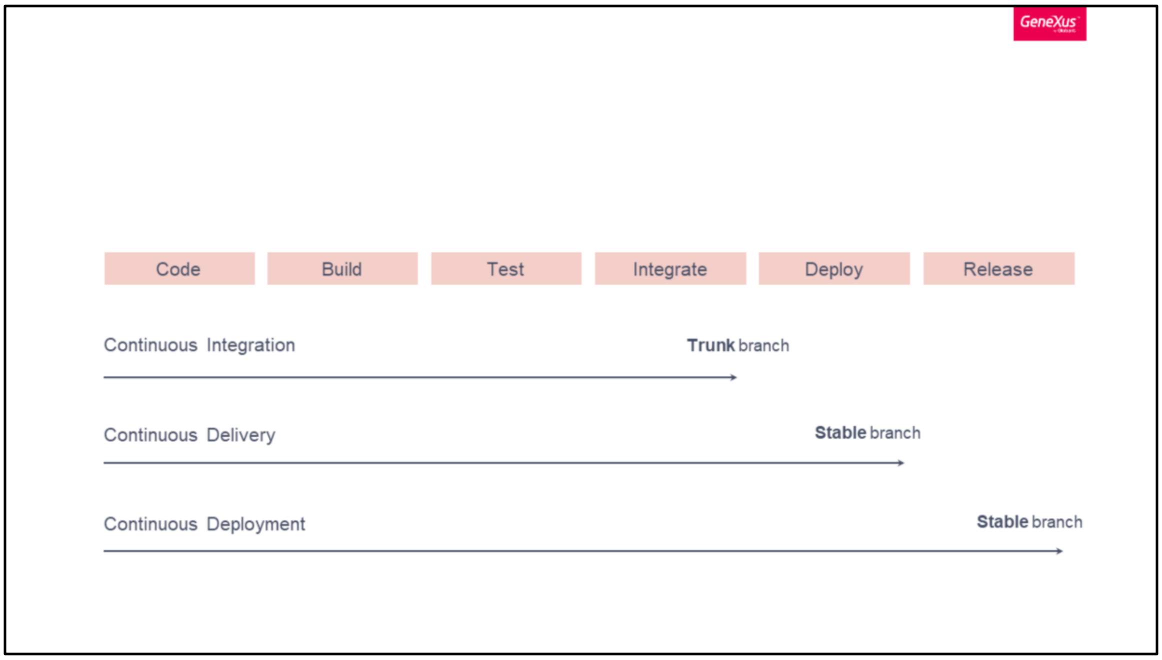
https://wiki.genexus.com/commwiki/servlet/wiki?38329,Typical+GeneXus+developme nt+cycle+for+Agile

Trunk-based development (TBD) refers to a process for managing a project within source control where all developers working on the project commit their code changes directly to the trunk (primary/master) branch. This model is recommended for small teams and experienced developers. It is a common practice to freeze the released versions.

We recommend to start using two different development branches: **Trunk** (Development) and **Stable** versions, where **Stable** is the consolidated Knowledge Base with all the features (already developed) that you decided to make part of the release. On the other hand, Trunk is the "beta" branch, where developers are working on new features and bug fixes every day.

The stable version will typically have the entire pipeline automated on each stage, so after you “merge” (bring changes) from Trunk, you want your tests to start running automatically under different stages to check the quality of the new version.

The names and amounts of versions are just a matter of consensus inside each team, what matters is to have a defined name for each type of branch and that the team is aware of the branching model.



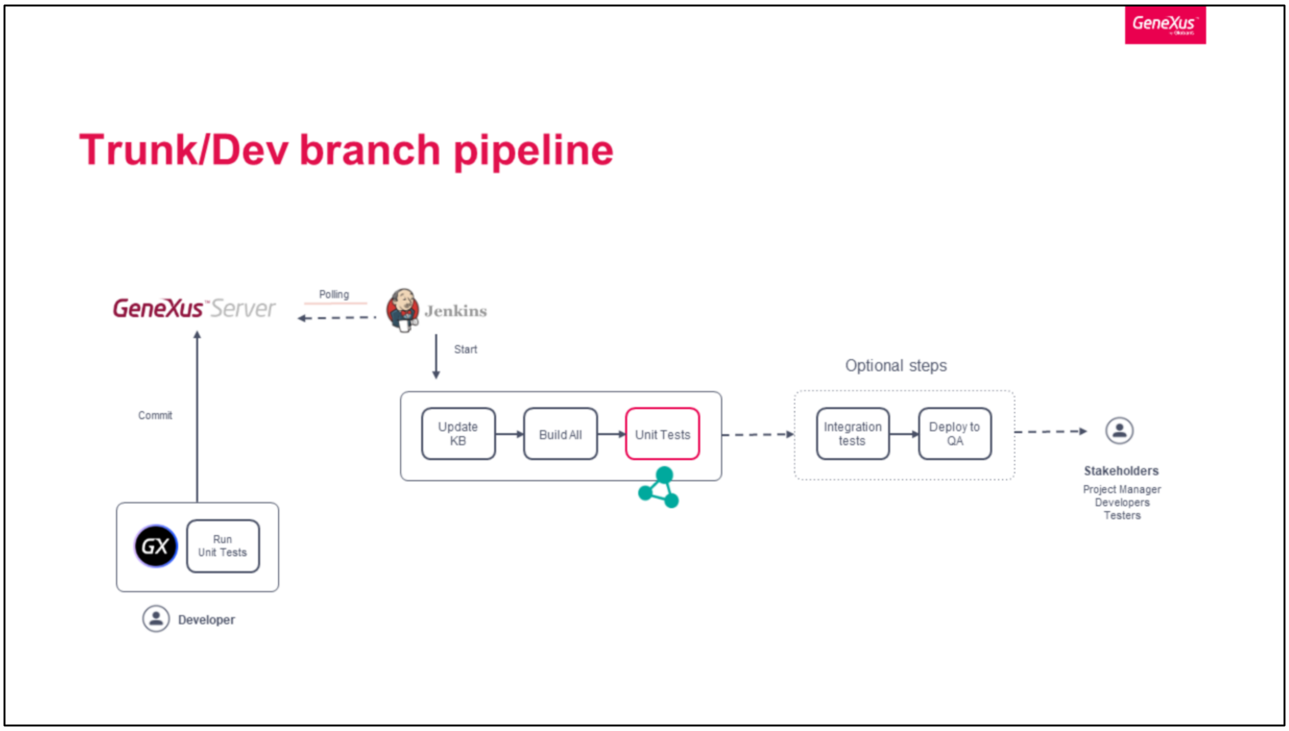
Source:
https://www.genexus.com/es/productos/genexus/versiones/genexus17/videos/cultura-devops-con-genexus (min 5)

A pipeline represents a chain of processing elements arranged so that the output of each element is the input of the next. This implies that the pipeline breaks down software delivery into several stages which verifies the quality of the new features and prevents bugs from affecting users.

Pipelines with automated tests allows to achieve continuous integration, continuous delivery and continuous deployment.

Continuous integration refers to integrating the code of each developer and validating it with unit tests. Continuous integration is typically configured in the Trunk branch.

When we automate the release, and with just one click we can put our application into production, we have continuous delivery. When the entire process is automated and the code is automatically put into production, that is where we achieve continuous deployment. Typically these last two schemes are configured for the Stable branch.



Trunk version is where developers are coding new features and/or fixing bugs. Those changes can be made over different objects by many developers, which means that they will make commits to this branch often.

The best way to work in this version is to make sure that all developer's changes are integrated frequently, to make sure that the system is always in a “buildable state” and the quality of the code in your Knowledge Base is respecting good development practices. We recommend to do it daily, and configure Continuous Integration in the Development branch to trigger automatic tasks when the developer commits changes to de GXserver. By doing so, your process will have an automatic check that ensures that recent changes don’t break the build and still respect quality standards.

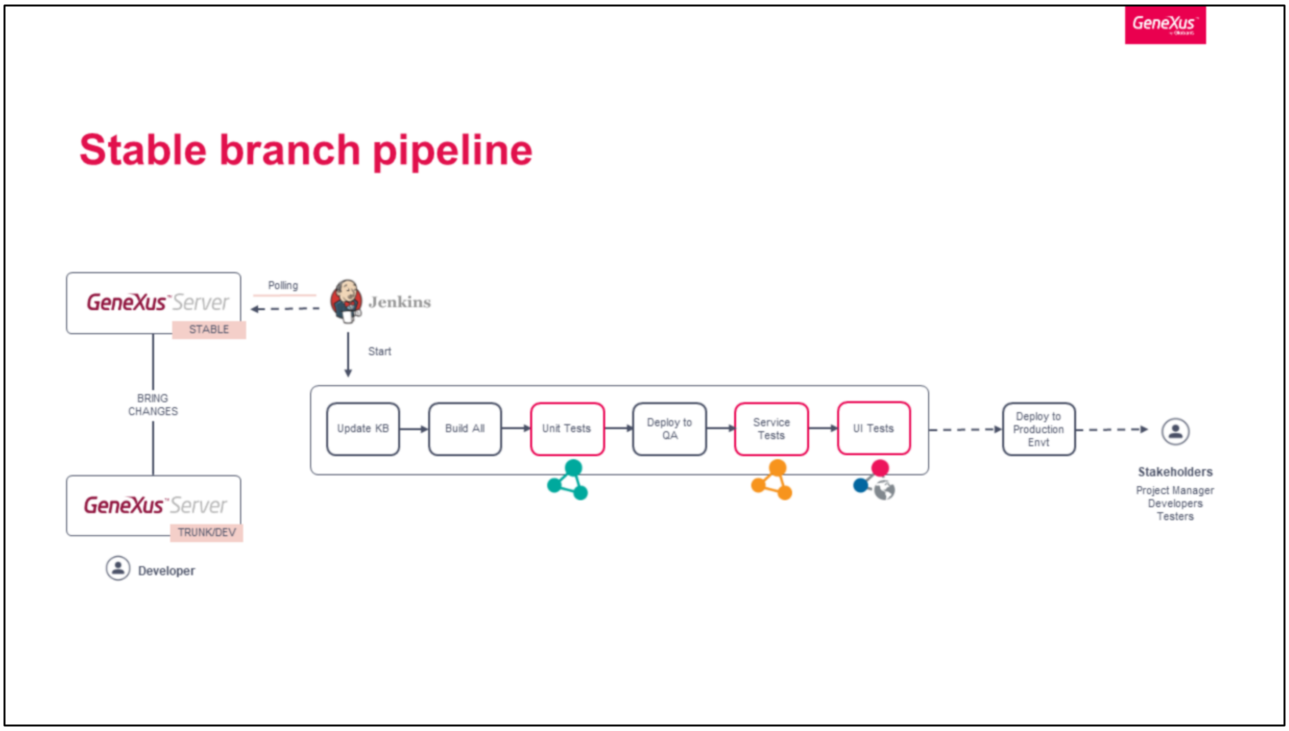
If you have a Continuous Integration server like Jenkins, the first thing to do is to add some listeners, that will be waiting for new changes. So, in this case, the pipeline should run (at least) the following steps:

1. Update latest changes from GXserver (Trunk Version)
   This will enable you to update the artifact to build (KB) to the latest commit.
2. Build All
   This will perform a build all to specify, generate and compile the KB objects.

3. Run Unit Tests

There are some optional (and recommended) steps which can be running static code checks to see if they are compliant with good practices (i.e. KB Doctor tool), running some basic automated integration tests and deploying to a QA environment for manual tests.

During this training, you will learn how to set this kind of pipeline.



## Stable branch pipeline

This version hosts the main changes that you decided to be part of the new consolidated version (candidate for a new release). Those changes are typically a set of commits from other versions, for example, the Development branch, made by one or more developers.

In the Stable branch, not only you want to automatically build your KB and check quality standards, but also to run different tests (API tests and UI tests). As a result, every time you introduce changes to this version, the pipeline will check that it is in a “deployable" state and it is functional.

The pipeline jobs are the same as defined in Development version plus adding some further steps to deploy the application to QA environments, and run services and UI tests after these. In the image, you can see the continuous delivery approach with the following automatic tasks: update, build, unit tests execution, deployment to QA, service tests execution and user interface tests execution.

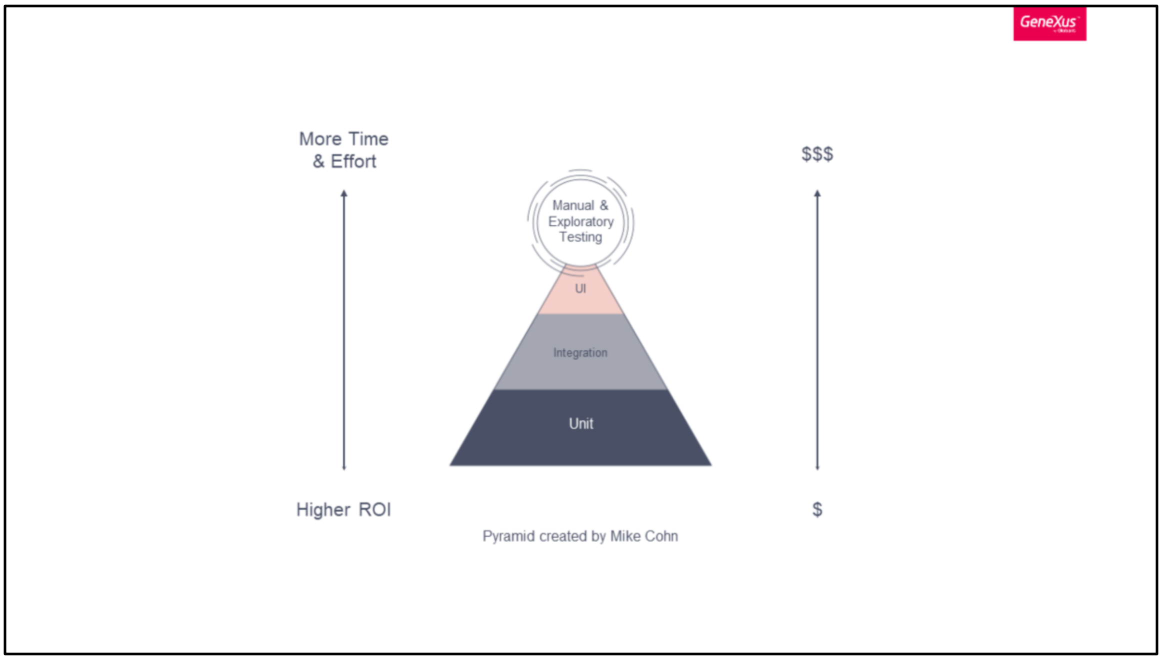
Note that in this approach the deployment to the production environment is manual. It is possible to automate this step, the approach is called continuous deployment. To achieve this approach it is necessary to have robust tests by which we make sure that the new version could be deployed to the customer.

There are several layers of test automation that add value to the pipeline, depending on factors like the cost of automation, how often and how fast tests are running, in which environment the test runs, and how valuable/effective checks are.

Identifying where the greatest risks are, how to mitigate them with tests at different levels and based on the automation pyramid, each organization decides what types of tests and how much to implement.

Automated testing allows to add value to the pipeline to perform quick validations and deliver value to the customers reducing the need for manual validations. This allows to deliver faster, continuously and with more quality.

## Test automation pyramid



Before automating an application, we must understand the different layers of testing represented in the ideal test automation pyramid by Mike Cohn.

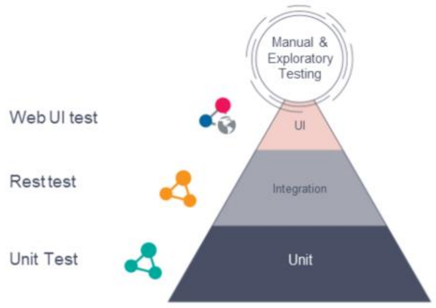
As you can see in the graphic, the most weight of test is in the unit testing capability. This capability is the fastest to run and the cheapest to maintain.

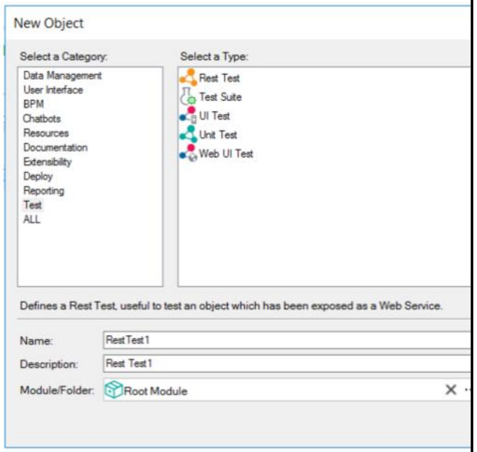
In the second testing capability, typically you can see integration tests by which we test different components of our application i.e. service testing.

In the third capability, we can find the UI test, such as end-to-end tests. They are slower to run and more expensive to maintain. This kind of tests are valuable for regression testing and user acceptance interface testing, but this kind of tests will require the application to be deployed and the testing environment is more expensive.

Finally, manual and exploratory tests will be reserved for functionalities that cannot be automated.

The exact mix of tests will depend again on each team, but the efforts invested on each specific test approach should follow this pyramid.




New Object

Select a Category:
- Data Management
- User Interface
- BPM
- Chatbots
- Resources
- Documentation
- Extensibility
- Deploy
- Reporting
- Test
- ALL

Select a Type:
- Rest Test
- Test Suite
- UI Test
- Unit Test
- Web UI Test

Defines a Rest Test, useful to test an object which has been exposed as a Web Service.

| | |
|---|---|
| Name: | RestTest1 |
| Description: | Rest Test1 |
| Module/Folder: | Root Module |

In GeneXus, the decision of which type of tests to start with depends on how the KB is structured and programmed, and the pipelines and applications needs.

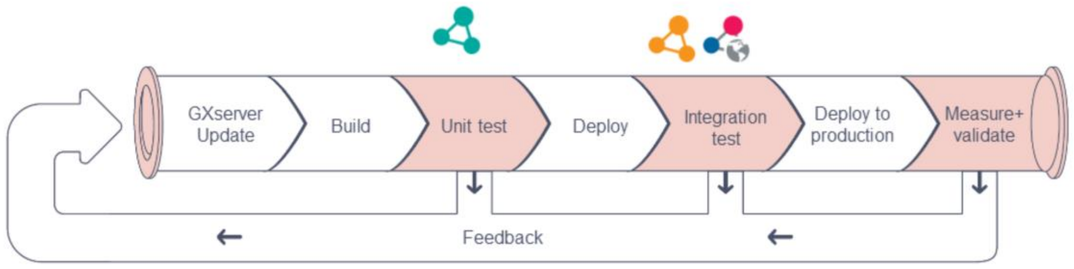
All kind of tests will be created from GeneXus IDE.

The unit tests are created with the Unit Test object. With this object it is possible to test the business logic encapsulated in other objects like procedures, data providers and business components.

With the Rest Test objects it is possible to test objects (procedures, data providers and business components) exposed as REST services by calling them via HTTP.

User interface tests in GeneXus are implemented with Web UI test object or UI test object (mobile).

You have also available a Test Suite object, in which it is possible to group different kind of tests.



The automated tests will be added in the pipeline to check the application behavior after each change in the different environments.

GeneXus™ by Globant

training.genexus.com
wiki.genexus.com