CONTINUOUS TESTING IS ON THE ROAD TO DEVOPS

EFFICIENT DEVOPS REQUIRE ADAPTED TESTING TOOLS

Keywords:
DevOps
Continuous Integration
Continuous Testing
Continuous Delivery

Abstract: *This paper explains why **Continuous Testing (CT)** (2) is the cornerstone for make **DevOps** (1) culture efficient in any organization.*

*We describe what it is, when you use it and how you build it along your projects; We then explain how you can make it happened with XStudio and how it works with **Continuous Integration (CI)** (3) and allows for **Continuous Delivery (CD)** (4). There's no magic, just relentless engineering and rigor to make it efficient and sustainable. But XStudio can tremendously ease the job of the CI loop tool and make CD become a reality.*

Isn't this one of the basis for an efficient DevOps organization?

Summary

Introduction

In any software driven industry, we hear a lot about Continuous Integration (CI) which, in fact, happens to be a relatively old paradigm. And in the last few years, Continuous Delivery (CD) seemed to become the new goal for most organization. We tend to agree with the need for more streamlined and faster pipeline from idea to consumption.

Help

Figure 1: CI, CT and CD

➔ Everyone can notice that Continuous Testing (CT) is seldom mentioned as if it was a normal consequence or a sub-component of the 2 other patterns. We believe that, to the contrary, CT is clearly a mandatory part of the picture! At least if your goal is to deliver high quality solutions and products and make your business sustainable on the long run.

CI and CD

Here we present basic notions of Continuous Integration and Continuous Delivery.

CI aims to do the following:
- Detect whenever a part of a product has been modified
- (Re)build and (re)assemble a product from its component
- Install that assembly in environments where it can be exercised or used
- Monitor these steps and report on the success/failures to the right stakeholders

It is often implemented as a loop or a graph that will launch multiple specialized tools and get their reports back, analyze them and present the summary information into a dashboard. Some of the known tools are Jenkins, Hudson, Bamboo and TeamCity.

CD aims to do the following:
- Detect whenever a product is ready
- Identify and prepare the target environments
- Install (or ship) the product
- Communicate to the right stakeholders, including the users
- Trigger the post install processes (marketing, support, maintenance, monitoring)

In most organizations, most of it is embedded into the CI loop.

➔ Many see CI as the way to make things happen and CD as a potential benefit one gain from CI. But this is not the entire story.

From the time you modify a part of your product to the time you put in production or ship it to consumers, there is a set of verifications and validations to need to happen. You need to answer some questions:
- Is your product still functioning as expected?
- Is it still safe to be used?
- Is the new feature of real value for the users?
- Are we prepared to support it?
- Are we good with our marketing campaigns?

Release Pipeline

So, another useful notion that became more prevalent is the one of release pipeline.

Said it rapidly, it is the process that a product will undergo each time you need release to your users and customers. Defining a release pipeline is just as for any process:
- Define the stages
- Determine the conditions to move from one stage to another
- Identify the steps in each stage
- Set the conditions (pre and post) and outcomes you expect from the steps
- And so onï¿½

Here we provide a usual stage pattern but one should not take this as a recommended best practice:

- Build and integrate within system (build & System integration)
- Test functions (Product Quality Tests)
- Check it is safe and secure to use (Security/Safety verification)
- Test integration with surrounding environments (End-to-End testing)
- Ensure users get real value out of it (User Acceptance Testing)
- As needed, verify it is robust, scalable...

Figure 2: A typical release pipeline

Note that the release pipeline does not dictate the set of test environments. System Integration and Product Functional tests can be done on a QA environment, while Security/safety, End to End and User Acceptance tests can be done on a staging environment, and '-ities' tests would be done on a pre-production.

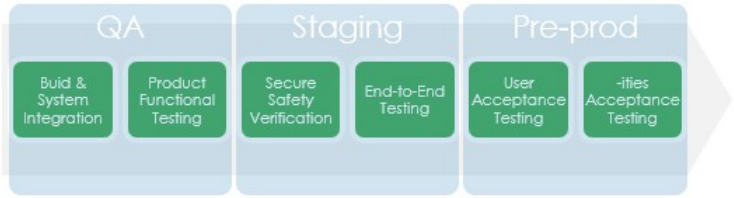

Figure 3: Mapping release stages to environments

➔ It all depends on the set of compliance you need to achieve (IEC 62304 & ISO 14971, FDA Title 21 CFR Part 11, ISO 14971, and IEC 60601, ISO 26262, DO-178C & DO-254), your own organizational barriers, andï¿½ how rich you can be.

So What About CT?

So, a release pipeline realizes 4 types of tasks:

- **Setting up** and cleaning environments when needed
- **Installing** products
- **Verifying and validating**
- **Taking decisions** or helping to make those between stages

In the software-driven industry, the most time-consuming set of tasks are (well...'should be') **Verifying and Validating**.

At each stage, you need to do specific tests to ensure your new or modified product is delivering on its quality attributes and expected values.

Each time you wish to deliver a new version of the product you must make sure it improves or at list maintain the same level of perceived quality and safety as the prior version while enabling new functionalities or enhancing existing ones.

This is what Continuous Testing brings to the game:

- Relentlessly testing and measuring quality levels
- Doing it as rapidly and effortlessly as possible
- Providing decision makers with the right information for managing the release pipeline
- Automating decision whenever possible

If we look at an example for the **Functional Testing** stage, we could see the following steps.

Figure 4: Functional testing stages

Note that the **Setting up Env.** and **Cleaning up** can be extremely time consuming and costly depending on the type of product you are addressing (a simple software is not as complex as an automobile for example...).

➔ But the game is to reduce the time of each of these steps. This allows to pass these steps more often. So, the key word and practice here is automation.

Every step is a Test

Any step (incl. setting up an environment), when it doesn't require too much manual interventions, should be automated and the result should be checked as for any test.

For example, if you are relying on Virtual Machines (VM), you may use **Vagrant** to load it up and then do some checks to ensure it is up and running. When installing the product, you could use **ANSIBLE** to deploy and configure and then do some verification on some files. Then you can launch some basic Smoke tests using **Robot Framework** or **SOAPUI** for API and so on.

➔ You need to use multiple tools and automation test frameworks within a single stage of your release pipeline.

This is where **XStudio** comes into play.

XStudio's ability to launch any kind of tests and to interpret and gather the results back, tremendously facilitates the management of a release Pipeline.

You design your campaigns for each step. Once executed, each test session will provide the tests results. Those are used for decision making on what to do as next step.

You use a third-party CI tool to orchestrate the various campaigns.

XQual's XContinuousIntegration is used by these CI tools to launch the various campaigns and to get the reports and test results back.

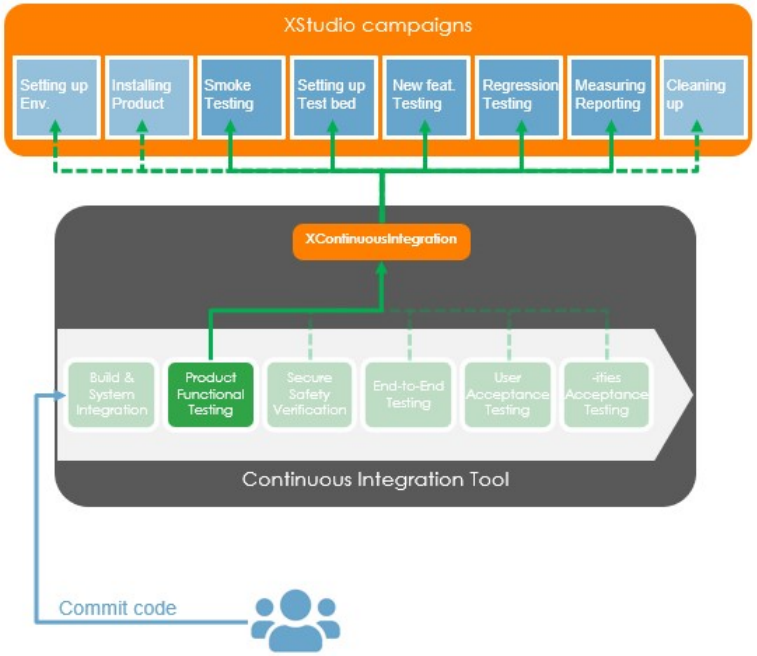

Figure 5: XContinuousIntegration integration

XContinuousIntegration provides all detailed reports and add a JUNIT formatted synthesis result file that can be parsed by the various CI tools. The CI tools remains the one to decide how to manage the loop (e.g. if the next step can be started or if the current stage should be stopped).

➔ This pattern allows you decoupling the implementation of your steps from the main CI loop. Detailed results are managed into the XStudio Database. All the results are consolidated through the smart KPI of XStudio

Whatever test management tool you use, automation is a key factor if you want to support faster delivery cycle.

XStudio supports nearly 90 test automation frameworks and you can add any new one in a matter of hours. But **nothing will happen unless you invest in automating the tests**.

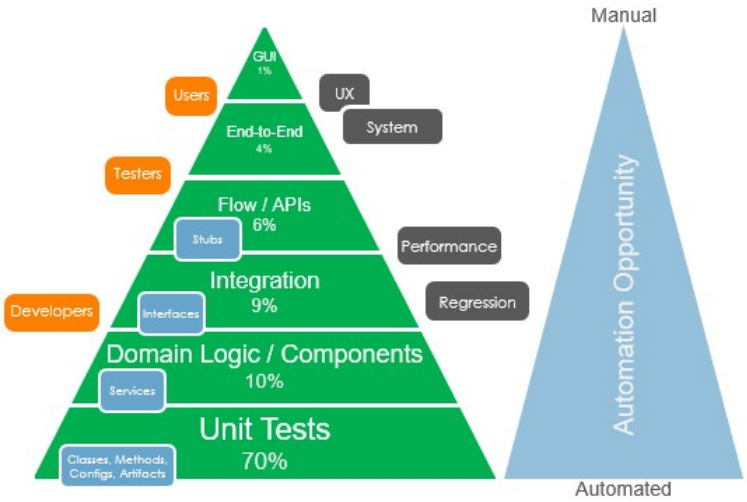

Figure 6: The_test_pyramid

➔ Given the large set of tools you can choose from when delivering on your test pyramid, getting the benefit of centralized and versatile test orchestration platform such as XStudio is a must to succeed.

Conclusion:

WHEN IT COMES TO TRANSFORM A SOFTWARE DRIVEN ORGANIZATION TO DEVOPS, ONE MUST BE CAREFUL TO BUILD THE RIGHT PRACTICES AND CULTURE. BUT TRANSFORMATION IS NOT REVOLUTION. TESTING ACTIVITIES ARE ATTHE HEART OF THE CONTINUOUS DELIVERY. YOU CAN'T CREATE, BUILD AND DELIVER FASTER, IF YOU DON'T TEST FASTER. AUTOMATION IS KEY, AS WELL AS INTEGRATING TEST ACTIVITIES INTO THE DELIVERY FLOW YOU MUST BUILD ON THE TESTING PRACTICES AND TOOLS THAT SIMPLIFY THE TRANSFORMATION. XSTUDIO IS A KEY ENABLER FOR THIS.

References:

1. DevOps: https://en.wikipedia.org/wiki/DevOps
2. Continuous Testing (CT): https://en.wikipedia.org/wiki/Continuous_testing
3. Continuous Integration (CI): https://en.wikipedia.org/wiki/Continuous_integration
4. Continuous Delivery (CD): https://en.wikipedia.org/wiki/Continuous_delivery
5. Agile: https://agilemanifesto.org/

Figures: