I’ve had the unique opportunity over the last 30+ years to be involved in multiple large, enterprisewide, mission critical application development projects. I’ve also had the opportunity to develop and bring several products to market. In looking back at these projects, the common thread in the successful ones (yes, some were not successful) was the ability to respond quickly and effectively to the changing needs of the end user. Whether these were because of vague or misunderstood requirements, a changing technology landscape or competitive pressures, the ability to react quickly and change directions withoutregressing the end product
was a critical factor in their success.

Getting these projects and organizations into an agile posture and reactive, adaptive position was not without its trials and tribulations. In every case, we had our share of setbacks and missteps along the way. And as more and more organizations embark on their journey to DevOps, each of them will undoubtedly encounter some of these same challenges.

I would like to share these past experiences and observations, both the success and particularly the failures with others so they can benefit and avoid these pitfalls and hopefully smooth out their road to a successful adoption of DevOps.

**What’s in Your Continuous Integration?**

So, what’s in your continuous integration, or CI, process? What does it do and what does it produce or provide as an end product? It’s interesting most projects don’t know what their CI process does, only that it “integrates” or “builds the product.” But you should really know more than that about a component so critical to your ability to deliver to your end users.

The CI process should be viewed as you would any other piece of software or application. You should develop and document clear functional requirements that dictate what it should do for the overall process as well as each phase of the process and exactly what it should produce.

I realize the requirements for CI will differ significantly from project to project. I’ve seen a range of CI processes —from very basic to everything plus the kitchen sink. What I’d like to understand though is, what are the basics requirements one would expect from a CI process? And more importantly, what should organizations expect as an end product from their CI process?
I’d like to propose a set of basic requirements or core capabilities that should be in your CI process. In doing so, I’m going to be looking at the CI process in terms of phases in the build life cycle. This life cycle proceeds from phase to phase with each contributing something to the end product and typically providing input into the next phase. The following is a list of those phases, along with a description and narrative I believe should serve as at least the starting point as requirements for your CI process.

**Initialize and Validate**: This phase provides for any initialization to occur for the build to complete successfully such as creating directories, setting variables and properties, etc. During this phase, formatting and style checks on the code according to project standards should be performed. Curiously, standardizing formatting seems to invoke some of the most heated debates and hotly contested discussions. I’m all for creativity and understand we all have our preferences when it comes to coding style. But for those who have to maintain the code long after the author is gone, I want it to be easily read and follow the code and be comfortable fixing and extending it. Establish projectwide formatting and style standards and stick with them.

**Manage Resources**: This phase is for any resources that need to be created, copied or processed in any way. This can include creation of test data, copying of files and directories, transformation of data and so on.

**Compilation**: Of course, we’re going to compile. While this seems too obvious to point out, make sure you’ve parameterized and standardized how the compilation will occur. For example, ensure the version of compiler and target platform are explicitly set and those values are used throughout the project. It really is important how your object or byte code is created. When the rubber hits the road, the bits and bytes need to line up.
**Test:** Here’s where we test our code at the unit and component level. And with some application frameworks, we’re able to go a bit further by testing how different “slices” of a system interact. The unit and component tests created by developers are run with each build. This includes the previous tests along with any new tests that cover any new or modified code. By running all the tests with each build and having them successfully complete, we are guarding or testing the modifications or additions to the code did not regress the functionality.

**Packaging:** The code is packaged along with any required resources into the required format — be a JAR, WAR, Bundle, etc. The end product is either the entire monolithic application or a component or part of the system or application to be deployed.

**Integration Test:** Strangely enough, despite its name, this phase is one of the most neglected and overlooked. This may be because it can be one of the most difficult and involved to implement. However, it’s where I believe you can achieve some of your highest return on investment. The integration phase is usually broken down into three separate steps.

The first is the pre-integration step, used to provision and set up the environment where the application will be deployed. When this phase is completed, the application should be deployed to the target platform, all resources provisioned and available and the application in a state where automated testing tools can exercise the functionality.

The second phase is the actual integration testing. Automated testing should drive the application from the user interface, application programming interfaces and any other access points that make sense for the application. This usually is the first opportunity where all the components and resources are brought together and must interact effectively.

This can pose a challenge, especially when considering the effort required to stand up some components such as large, distributed databases. But considering the value derived from the integration across the application, every effort should be made to create as effective and inclusive integration environment.

The third phase is used for cleanup. The application or servers should be shut down, all resources cleaned up and reclaimed and the environment reset as when the
testing started.

**Verify and Validate**: This phase should gather code metrics using static code analysis tools — many of which are available as open source. These code metrics should be aggregated and uploaded into tools such as the open source tool SonarQube, which provides management transparency on the overall health of the code. They also identify and track issues such as technical debt, code smells and test coverage via a drill-down graphical user interface. In addition, the code metrics should be gathered with each build or at least on a daily or nightly basis to support historical trends to determine if code quality is trending in a positive direction.

**Deploy Artifacts**: Not to be confused with a promotion of a release candidate to the next environment, this step will deploy the package or artifacts from the build into a remote artifact repository for use by other developers and build processes within the project and potentially across the organization. The artifact repository should allow for the versioning of each artifact so builds can have access to different versions of an artifact. For example, a build may want to use the most stable versions of most artifacts in the project but may want to use the very latest versions or snapshots of a particular artifact.

This isn’t an exhaustive list of what a CI process can do. But hopefully, this at least gets the conversation going concerning your project’s requirements for your continuous integration process. It’s one of the cornerstones of your delivery pipeline so it’s important you determine these requirements up front and understand not only what it does, but also what it produces for you. Don’t assume just because you’ve been told you have a CI process it does what you need to do. Be prepared to answer the question, “What’s in your CI?”

View printer friendly version
agile
Non-feature
The Agile Advocate
Standard