The Importance of Continuous Integration for Quality Assurance Teams

Without proper implementation, a continuous integration system will go from a competitive advantage for a software quality assurance team to a negative return on investment.

Continuous integration systems can help QA teams increase project visibility, increase confidence and maintain deployable software while reducing risks, repetitive manual processes, and overal cost. Because a continuous integration system can be used with virtually any type of software test, it is an essential part of an automation effort when properly installed.

Magenic®

White Paper
Aaron Humerickhouse

Continuous integration is a software development practice where members of a team integrate their work frequently.

What is CI?

"Continuous Integration (CI) is a software development practice where members of a team integrate their work frequently, leading to multiple integrations per day. Each integration is verified by an automated build, including tests, to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly."[1]

The entire process starts with new code. When an individual commits code to version control, the master node recognizes the change and creates its own build. If the submitted build passes, everything is good to move forward. If the build fails, it is broken and the problem needs to be addressed immediately, before anyone commits code to the version control. This process helps maintain the principle of always having a deliverable piece of software.

An Example of CI

Figure 1 provides an example of a CI system with child nodes. In the example, Tim and Jane are working on the same project and sharing a version control system. Tim is ready to commit his code changes and unit tests to the version control. He runs a private build to make sure his code will not break the build. His private build passes and he commits to version control. Because the master node is continuously polling the version control for changes made, the master node recognizes Tim's commit and builds it to make sure there are no errors. When it sees that there are no errors in the build, the master node then pushes code to the child nodes. Now the child nodes are ready to run all of the tests. The child nodes are then able to run any automated test, including, but not limited to: unit, user interface (UI), regression, smoke, load, stress, and performance tests. Everything passes and Tim's code is successfully committed.

Jane sees that new code has been committed and the CI system still reports passing results. She pulls the latest code from version control, knowing that it is the best practice to always be working with the latest code. Jane makes her code changes, forgets to run a private build, and then commits her changes to version control. While the master node is still polling for new changes, it recognizes Jane's commit and builds with the new

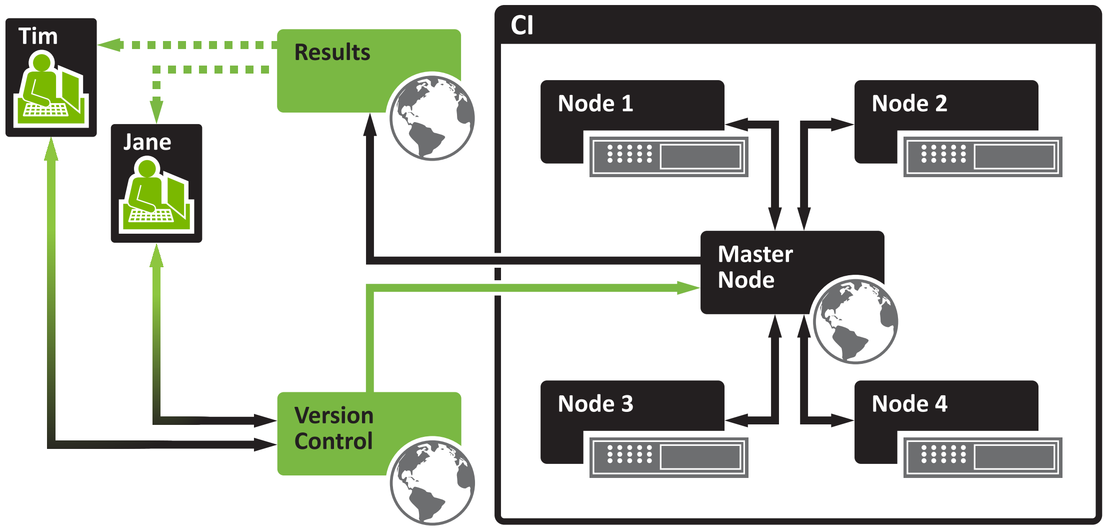

Figure 1 - A standard Continuous Integration system with child nodes.

code. The build fails, and the CI system now reports failure. Tim wants to pull the latest, sees that the build is broken and looks into it right away, as that is another best practice. Because of the CI system, he was able to avoid pulling broken code. He sees that Jane's last set of code is the source of the broken build and discusses it with her. She remembers she failed to run a private build and reverts her changes in version control. The CI system recognizes the change and builds. When Jane's build passes, the master node then pushes the code to the child nodes which are awaiting testing. Similar results will be shown if a private build is successful but the code does not correctly integrate into the rest of the software. This can be common if the private build does not include the entire project.

Individuals who commit code in a CI system will immediately see if the code integrates correctly or not.

General Team Advantages

Project Visibility

A CI system can have very powerful reporting capabilities. With the ability to store previous results, trending can be observed. With defined trends, effective decisions can be made, leading to reduced time in planning and execution. Stronger reporting also allows a more accurate understanding of progress can be sent to the client and management. Additionally, knowing the number of defects introduced, on average, greatly increases the accuracy of planning.

Confidence

Individuals who commit code in a CI system will immediately see if the code integrates correctly or not. When the code integrates correctly, there is a new deployable piece of software. When the code does not integrate correctly, the build fails and an individual can quickly revert their changes to fix the problem.

Maintain Deployable Software

The third principle behind the Agile Manifesto is "[d]eliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale." A CI system will allow a team to produce a deployable piece of software at any time as long as it is free from defects or errors. A CI system will point to the defects or errors and allow an individual to quickly identify the issue. If a team is continuously integrating their software changes, better software is produced and tested more often.

General Team Disadvantages

Increased Overhead

As with any new addition to a project, it takes time to learn the software and how to set it up. However, if the project is large enough, use of a CI system will justify time spent on the learning curve and set up. Once a CI system is up and running, maintenance time will be incurred. Some projects have multiple releases throughout their lives. Trying to juggle multiple releases across multiple nodes is cumbersome.

Initial Cost

Estimating the cost of a machine to be $2000 makes it easy to look past a network of machines and a CI system. However, the time and resources saved thanks to the system will justify the one-time cost commitment.   The cost of a CI system environment is much lower than that of a defect found too late in the development process.

Mental Factors

In addition to tangible limitations, there are mental factors as to why a team does not want to implement a CI system. Once a CI system is in place, there may a higher failure rate in testing than previously seen. Management can quickly be discouraged by this, but should not fret. This is how a CI system works! By design, a higher percentage of tests will fail because individuals will be committing changes more frequently, spending less time worrying about an integration being perfect. This is what makes a CI system effective. Even though the system will be performing these activities, developers will still be performing the manual build confirmations as expected.

Instead of the developer spending time searching every nook and cranny to make sure their code integrates, the CI system automates the process

Developers also get discouraged by the volume of changes to the code. Generally developers in traditional environments like to make sure large lumps of code are perfect before integration. This can be a time waster. If a change passes the private build and is pushed, the CI system will automatically communicate whether or not it has integrated correctly, most often within the hour. Instead of the developer spending time searching every nook and cranny to make sure their code integrates, the CI system automates the process over several nodes.
Another reason that teams shy away is because developers should already be running private builds before integrating, performing static analysis, etc. While this is true, humans make errors and can forget to perform the necessary actions before committing code. Running a CI system guarantees that these are performed every time code is committed and gives visibility to the action.

QA Advantages of using CI

Reduce Risks

A CI system creates the ability to focus on higher areas of risk while it runs regression tests or tests areas of the project that have been stable for many past iterations. Time is saved, allowing resources to find new defects that may have otherwise been overlooked due to time limitations. Another perk of a CI system is that defects can be found immediately. A CI system is the best way to run regression tests. While allowing time to continue work in other areas, the CI system simultaneously runs regression tests after each code integration. By continuously integrating software changes and running tests after each commit, assumptions – such as previously stabe areas remaining stable – can be minimized, as many will be verified as pass or fail.

A CI system creates the ability to focus on higher areas of risk while it runs regression tests or tests areas of the project that have been stable for many past iterations.

Maintain Deployable Software

An automation framework should be treated as a software development project. This includes version control and testing. If the automation framework is being built and tested through the CI system, it will also be deployable if it is free of defects or errors.

Reduce Repetitive Manual Processes

Without a CI system, every build requires time to be manually investigated. If a CI system is implemented, jobs will be kicked off once a new build is delivered. This can include smoke and regression testing. Depending on how often a build is delivered, a CI system can save time by running these tests and delivering the results. Assume for a minute that the client needs its web-based program to run on Internet Explorer 9 and Mozilla Firefox, and functionality on a few additional browsers as a perk. The CI system can be configured to run tests in multiple, randomly chosen, environments and test both required browsers throughout the day after a build. While beneficial, randomly choosing which environment is tested does not guarantee that every test is run in each browser throughout the day. To verify full functionality in both Internet Explorer 9 and Mozilla Firefox, a time trigger can be set to kick off the build with Internet Explorer 9 while everyone is out of the office, in order to limit resource time wasted. An event trigger can then be set up to run in Firefox after the Internet Explorer 9 tests are complete.

This verifies functionality in both browsers at night, while thoroughly testing each browser throughout the day.

Reduced Costs

Assume a QA team is assigned a build, on average, two and a half times per week. It takes about an hour to smoke test each deployed build on one configuration. If four configurations are required, that would be ten hours per week spent on smoke testing alone. If an automated smoke test takes half the time (often it will take less) and the CI system has four nodes to test on, smoke testing will take an hour and fifteen minutes per week. Not only do the testers not have to spend ten hours per week strictly smoke testing, but the smoke tests will be finished sooner, allowing earlier confirmation that the build is ready for testing. Extend this to regression, performance, load, and stress testing and the time savings continue to mount. (Table 1) Based on a hypothetical $85 hourly rate, the above scenario yields a savings of 142 hours, or $12,070, for just running smoke tests. If two identical teams worked on an identical project, one with a CI system and one without, the CI team will be delivering the same product with less man hours. The ability to deliver a quality product with less will allow for either fewer employees or more contracts throughout the year. If the estimate cost of a machine is $2,000, four child nodes and a master node result in a total cost of $10,000. This is only slightly less than the total cost saved, but this is a one-time cost. The next time a project uses these nodes the purchase costs are not incurred. The same costs saved would be true for a software company. They would be spending less on each project and they would be saving time being able to deliver software sooner.

	Manual	CI
Billing Rate	$85	$85
One-Time Costs		
CI Setup & Stabilization	$0	$3400
Total	**$0**	**$3400**
Manual Hours per Week		
Smoke Testing	10	0
Regression Testing	10	0
Performance Testing	10	0
Smoke Testing	10	0
Smoke Testing	10	0
Total	**$18,700**	**$5,610**
Project Length - 26 Weeks		
Total Cost*	**$486,200**	**$149,260**
One-Time Purchase of Five Nodes		
Total	**$0**	**$10,000**
CI Savings Over 26 Weeks		
Total		**$326,940†**

Table 1 - CI Cost Savings Table 1 Notes

All calculations, numbers & results are theoretical

* Total project cost is based on the following equation: One-Time Cost Total = Manual Hours per Week Cost Total x Project Length

† CI Total Cost - Manual Total Cost

How to Use CI in QA

Integrate with Development Team

The CI system should be set up to poll for changes in the development team's project. When code is committed, the jobs will start running as desired. Integrating the code and immediately running smoke tests will let the development team know that its project is testable and they are free to continue their work, passing the build off to QA.

know that its project is testable and they are free to continue their work, passing the build off to QA.

Separate Automation Repository and Development Repository

An automation framework should be treated as a software development project; it should have its own repository for code and build jobs. If the changes pass, the automation tests should be run as well. The tests would show whether or not the changes affected the stability of the automation framework. Having separate build jobs would also remove false negative results by eliminating confusion as to whether the development project is broken or the automation framework is broken.

Windows 7 32-bit	**Windows 7 64-bit**	**Windows 8 32-bit**	**Windows 8 64-bit**

Table 2- Suggested Minimum Nodes for Windows 7 & Windows 8 with 32 & 64 Bit Configurations

Multiple Nodes with Multiple Configurations

In order to take full advantage of a CI system, all configurations must be present while creating nodes. If the end-user has the requirements of working on two browsers, there must be nodes with at least those two browsers covered. If the end-user requires the program to work on Windows 7 and Windows 8, a minimum of 2GB of RAM, and 32-bit and 64-bit, there should at least four nodes all with different configurations. (Table 2) The more nodes present, the faster the throughput will be if the tests are run in parallel. This guarantees that all automated tests are run across all required configurations, taking the same time as testing one configuration on one node. Compare this to manual testing; testing everything on all configurations is difficult and time consuming.

Smoke Test Each Build

Every time a new build is deployed, it should always be smoke tested. This is true for both manual testing and automating the process. Developers are able to if their build is testable or if they need to revert their code changes and make some quick fixes. Time is saved by having the CI system do the smoke testing, freeing individuals to work on other parts of the build.

What Types of Tests Should Be Run in CI

The great thing about a CI system is that any test that can be automated can be run. Knowing this, smoke, regression, load, stress, and performance tests should all be run. Smoke and regression tests may include any type of tests, such as database or UI automated tests.

Use the Results to Create Bugs

A CI system can find bugs without human interaction; however these bugs may not be properly reported. An individual needs to investigate each failed job. If the failure is a defect, the individual should log the defect, or make an edit if the CI system is set up to automatically submit defects.

Use for Planning

Before each planning session, an individual needs to look back through the CI system's log in order to help decide how much time needs to be designated for defect work on the development team's side. This helps keep the project on track avoiding poor time planning to fix defects.

Conclusion

A CI system, with its multiple advantages, is a great addition to the automation effort on a QA team. A CI system helps reduce risks, maintains deployable software, generates project visibility, provides confidence, reduces repetitive manual processes, and most importantly saves money. Because a CI setup costs less than the cost of time saved throughout the project, including a CI system is essential to an automation effort.

References

[1] Martin Fowler

Principles behind the Agile Manifesto

http://www.martinfowler.com/articles/continuousIntegration.html

http://agilemanifesto.org/principles.html

Glossary of Terms

Build - A set of activities performed to show the quality of the source code.

Continuous Integration of CI- A software development system that ensures code changes are compiled and tested routinely.

Individual - These are members of the team writing code and committing to a version control.

Job - A designated action controlled by a CI, e.g. Automation Suite or Static Analysis.

Node - A virtual or physical machine that are reserved specifically for CI testing.

Private Build - A build performed on an individuals machine.

Results - The visual aspect of the CI; it shows whether or not a build is passing or failing.

User Interface (UI) - The means by which the user and a computer system interact, in particular the use of input devices and software.

Version Control - The management of source code for a team or project.

About the Author - Aaron Humerickhouse

Aaron Humerickhouse has been an associate QA tester with Magenic since January 2012. He works closely with QAT leads to ensure that Magenic's Adaptive Risk-Based Quality (MARQ) testing standards and guidelines are ensured on each project. Aaron holds a B.S. in Software Engineering from the University of Minnesota.

About Magenic

Founded in 1995 by the same technical minds that still run the company, Magenic focuses on the Microsoft stack and mobile application development.

Magenic®

Visit us at **magenic.com** or call us at **877.277.1044** to learn more or to engage Magenic today.