CollabNet

Building Value with Continuous Integration

ALM

http://www.collab.net

Abstract

Implementing continuous integration involves choosing the right tools and technology. It also involves examining the software delivery pipeline for waste, introducing automation, building feedback mechanisms into processes, and fostering cohesive teams that collaborate across functions. This paper is written to provide guidance for teams new to continuous integration or wanting to explore ways to improve their implementations. The topics explored in this paper include Value Stream Maps, Antipatterns, and Tools and Technology for Continuous Integration.

Table of Contents

Extending Agile's Benefits with Continous Integration

Agile methods use an iterative and incremental approach to software development. This approach has proven to be effective when it comes to dealing with all of the changing requirements inherent in the development phase of a software release. By working in sprints and focusing their efforts on the high risk and high value features first, agile teams are able to build business value quickly, while reducing much of a project's risk early on. Agile teams put emphasis on team collaboration, continuously aligning the software delivered to business and stakeholder needs, and adapting to changing requirements throughout the process.

Oftentimes, Agile is applied predominately to the requirements gathering phase, and only to a lesser extent to the rest of the software delivery pipeline. To harness Agile's true potential and to most effectively manage change, however, it's critical that teams extend Agile practices to the entire software delivery pipeline. One area that poses the largest challenge for Agile teams, and often the area that can slow down the go-to-market process most, is "build and test".

Continuous integration is a software development practice specifically designed to improve the build and test process and complement the speed with which Agile teams work. Teams integrate their working code early and often, usually daily, so as to avoid the pitfalls of deferring integration. As code is integrated it is verified in the build process to detect errors as quickly as possible. If a build fails, the information is fed back to the team through a feedback loop so that the cause can be resolved as quickly as possible. The objective of continuous integration is to eliminate waste and rework using rapid feedback. Teams find that this approach improves release quality and leads to significantly reduced integration problems – thereby accelerating the overall release process, and saving time and money.

For teams looking to improve their development processes with continuous integration – the first step is to critically examine the 'status quo' process, to identify areas that can be improved. Continuous integration is tied to the concepts of "eliminating waste" and "rapid feedback". Waste is defined as anything that does not add value to the customer. Let's look at these concepts, using Value Stream Maps.

A Value Stream Map is a visual model of the software development process that helps to uncover waste in the current process. It helps teams visualize what's going on as a "requirement" moves through the process from conception through release, and to identify the waste in the system and where value is being added. Value stream mapping is a lean manufacturing technique that originated at Toyota (Toyota Production System) but has been applied to the value chains of many different industries.

Lean Production

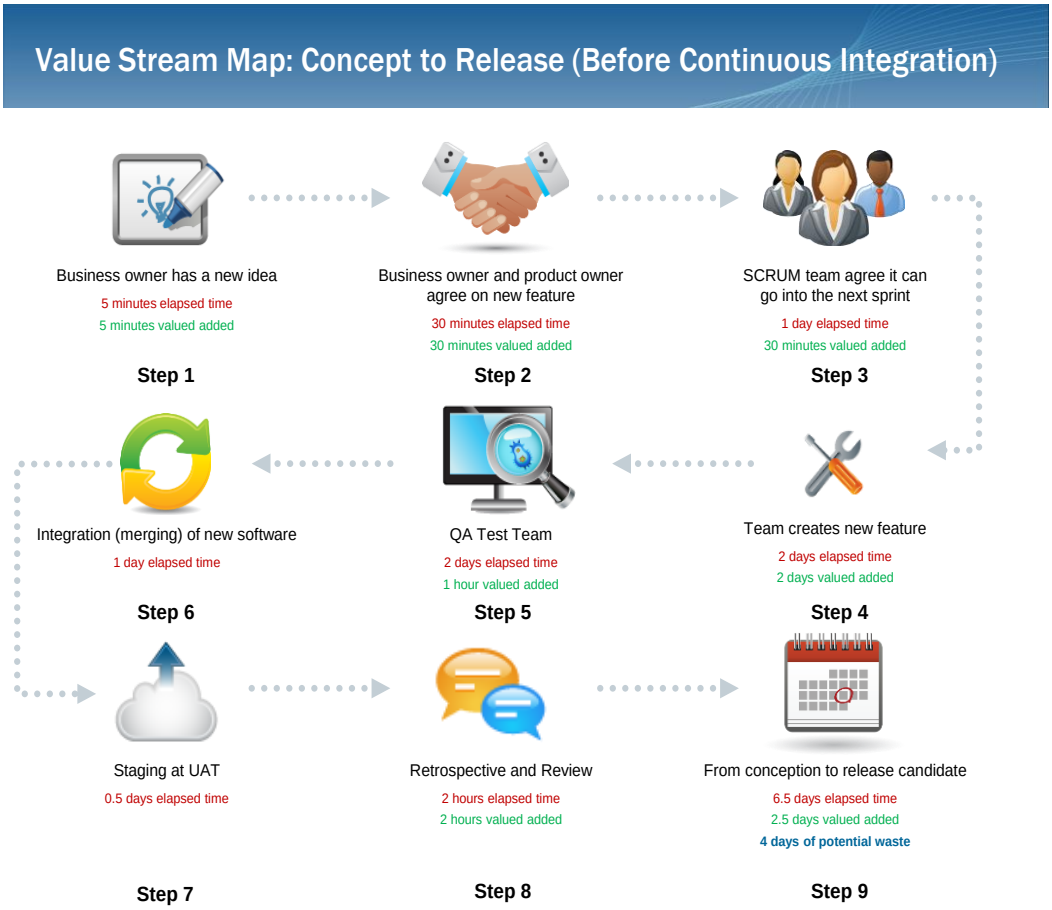
The software industry has adapted techniques from Lean which originated in the manufacturing industry. In the 1990's Toyota improved the mass production process by introducing methods that were considered lean and eliminated waste. In addition to Value Chain Mapping and Kanban, other techniques introduced were 'Just in Time' and 'Smarter Automation'. "Just in time' minimizes in-production inventory and associated carrying costs which helps enterprises maximize ROI in the production process. 'Smarter automation' is an automated process that helps to eliminate defects in production. When abnormalities are detected, production is stopped so that the root cause can be determined and the condition corrected. It's done in an automated way to ensure the highest product quality while keeping costs in check.

Value Stream Map: Concept to Release

To create a Value Stream Map, there are three fundamental steps:

1. Map the software development process, typically from inception to the delivery of the product. This process is sometimes referred to as "concept to cash"
2. Measure that process in terms of elapsed time versus value added time
3. Optimize the process and eliminate as much waste identified as possible

The following illustration shows a basic value stream map for the development of a new feature from concept to release. This value stream map shows the "Before state" of the software development process, meaning before it has been optimized with continuous integration. The table below the illustration provides details for each step in terms of activity description, elapsed time, waste and value added.



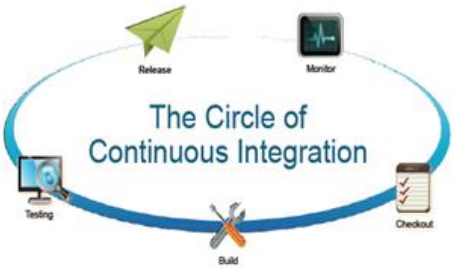
Scenario 1: Value Stream Map Before Continuous Integration			
		Time Elapsed	**Value Added**
Step 1	A business owner comes up with a new idea. It takes the business owner five minutes to develop that idea. That would be five minutes of value added time.	0.05 hours	+ 0.05 hours
Step 2	The business owner then goes to the product owner, and the two have a quick conversation and agree that the feature should be incorporated into the product. It's a 30 minute conversation: 30 minute of time elapses and 30 minutes of value is added.	0.5 hours	+ 0.5 hours
Step 3	The product owner then takes the concept to the Scrum team and the Scrum team discusses it to decide when they can incorporate this feature into a release. 1 day elapses and only 30 minutes of value is added.	8 hours	+ 0.5 hours
Step 4	The team then diligently works on the product. They deliver it within 2 days. They deliver 2 days of value within the 2 days of elapsed time.	16 hours	+ 16 hours
Step 5	Next, the quality team tests the product. Despite the fact that repetitive testing is required, the testing is not automated. The result is that 2 days elapse while only 1 hour of value is added.	16 hours	+ 1 hour
Step 6	The next step is the integration process which requires bringing in many parallel activities into one coherent running product. This is typically a painful process. It takes 1 day to integrate this product.	8 hours	0(1)
Step 7	Next the product is staged in a production like setting. In this case it's a somewhat dysfunctional process. The development team must work with the infrastructure team to obtain the resources to stage an application. This takes half a day, so half a day of elapsed time.	4 hours	0(1)
Step 8	After staging, a sprint review meeting is and the team demos the product for the product owner. 2 hours of time elapses and 2 hours of value is delivered.	2 hours	+ 2 hours
Total	Activities idea through release	54.55 hours (6.82 days)	20.05 hours (2.50 days)
		Equals 4 days potential waste	

(1)*The definition of "value" in Lean is that the activity must add value to the business. Even though integration and staging must be done – they don't in themselves add business value.*

When all of the activities are totaled, the team has identified 4.31 days of potential waste in the process from initial requirement through release. Now that the process has been mapped and the waste identified, the goal should now be to eliminate as much of that waste as possible.

Continuous Integration in the Software Delivery Process

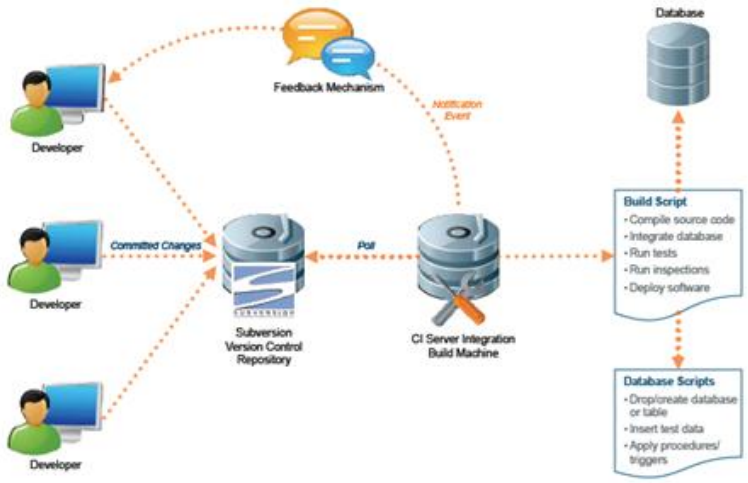
As mentioned earlier, one of the key benefits of implementing continuous integration is to provide rapid feedback so that if a defect is introduced into the code base, it can be identified and corrected as soon as possible. With continuous integration, isolated code changes are immediately tested and reported on when they are added to the larger code base. This helps to minimize risk by providing teams with real time notification and transparency, so that they understand what the build is doing and the current status of the software in terms of quality. Every time a change occurs that can potentially affect the product, the product is assembled, built, reassembled and then tested for functionality and quality. To be able to do this efficiently, the build and test processes must be automated.



The flow of continuous integration's iterative cycle at a high level is: "*monitor, check out, build, test and release - with a feedback mechanism*". . The flow of continuous integration requires that the team monitors the code base for any changes. Once a change occurs it needs to be checked out, built, and tested for quality. If successful and no defects are detected, it is released to a central storage repository or release area. If something does go wrong during the build process or a defect is detected, the team needs to know immediately that something went wrong so that this can be resolved quickly. Therefore there needs to be a built-in feedback mechanism. In that way, when a build breaks, the information feeds back to the central location or back to the developers that broke the build. This way they quickly understand what broke and how it broke, so that they can take quick action to resolve it. So, again at a high level it's "*monitor, check out, build, test and release - with a feedback mechanism built in*".

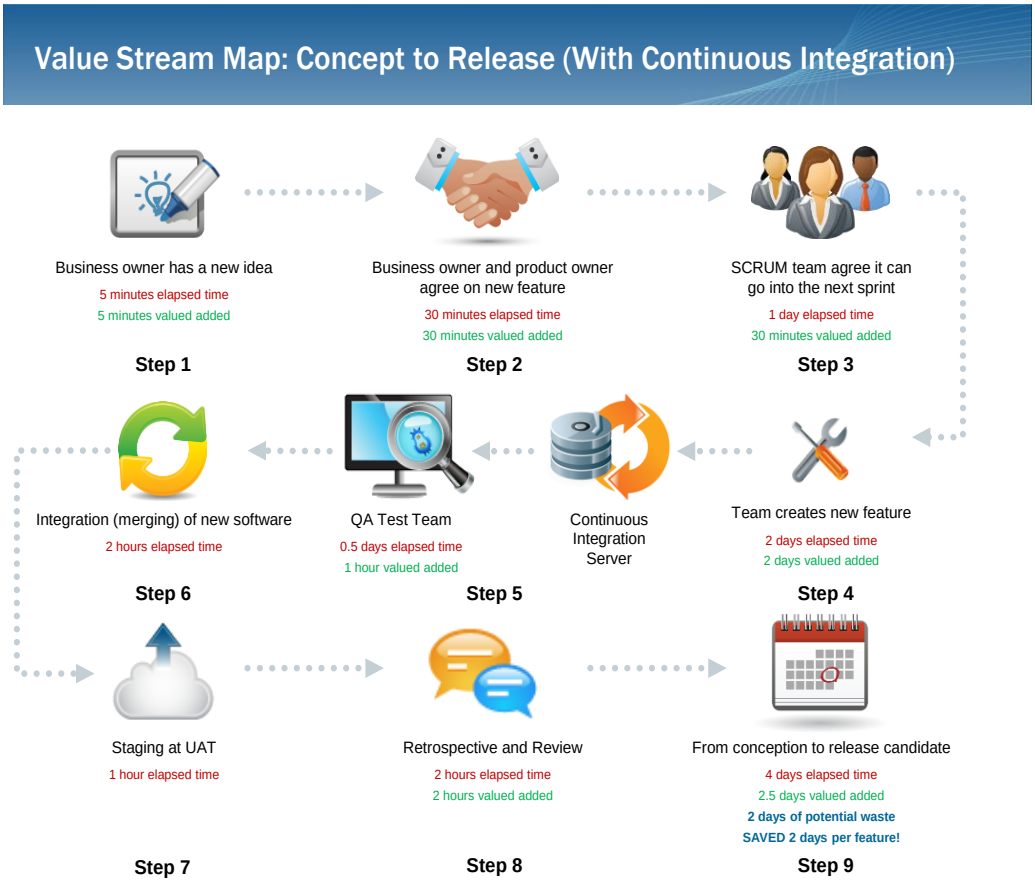
Continuous integration requires a central version control repository to house the software or working code. This repository typically should be integrated with the build server. In addition, the central store is required to house the test cases that are used for testing. When a developer commits the code to the central repository the build server polls the central repository for changes. The build server then starts assembling the relevant code, building that code and conducting unit testing. Functional testing is also conducted within the build process to determine if the application is functioning as it should be. The entire build-test process has a feedback mechanism built in so that at the end of the process, the developer knows the real-time state of the build and whether it has broken or not. In order to do this well, the team must incorporate build scripts for handling the source code compilation, integrated database run tests and inspections.

One important factor often overlooked is the need to establish cohesive teams that can collaborate across functions. It's also critical to build accountability within teams so that as issues arise, there is a clear sense of ownership to ensure issues will be addressed and resolved in a timely manner.



Value Stream Map: Concept to Release
Scenario 2: With Continuous Integration

Once continuous integration has been incorporated into the software delivery process, to determine the amount of waste that has been eliminated from the process, a new Value Stream Map can be drawn to compare to the original Value Stream Map.



Scenario 2: Value Stream Map With Continuous Integration			
		Time Elapsed	Value Added
Step 1	Waste has not been eliminated in these 3 steps since it still takes a day to deliver thirty minutes of value in step 3.	0.05 hours	+ 0.05 hours
Step 2		0.5 hours	+ 0.5 hours
Step 3		8 hours	+ 0.5 hours
Step 4	The team then diligently works on the product. They deliver it within 2 days. They deliver 2 days of value within the 2 days of elapsed time.	16 hours	+ 16 hours
Step 5	In step 5 a considerable amount of waste is eliminated using a continuous integration server. The QA team has started automating repetitive unit tests.	4 hours	+ 1 hour
Step 6	The integration process has become more streamlined since the team is integrating more frequently. Now it only takes two hours of elapsed time, instead of one day, to integrate the product into the unified application.	2 hours	0[(1)]
Step 7	Since the team is now using continuous deployment to release directly to staging, time spent interacting with the infrastructure team for releasing the application onto the system is eliminated. Now only one hour of time elapses to stage the application.	1 hour	0[(1)]
Step 8	After staging, a sprint review meeting is and the team demos the product for the product owner. 2 hours of time elapses and 2 hours of value is delivered.	2 hours	2 hours
Total	Activities idea through release	32 hours (4 days)	20.05 hours (2.50 days)
		Equals 2 days of waste eliminated (or 2 days per feature)	

[(1)] *The definition of "value" in Lean is that the activity must add value to the business. Even though integration and staging must be done – they don't in themselves add business value.*

Example: The Financial Benefits of Continuous Integration

In total for the second scenario using continuous integration, the team has eliminated 2 days of waste. Saving two days of waste per feature is significant, particularly when this savings is scaled across an enterprise with multiple teams and projects. For example, if a mid-sized enterprise has five development teams and each of these teams does twenty sprints per year, this would total one hundred sprints per year. If each team works at a rate of five features per sprint, together the teams would produce 500 new features per year.

For an enterprise developing 500 new features per year, two days of time saved per feature would translate into 1000 days saved over the course of a year. This would translate into $325,000 saved or a productivity increase of 3 persons per year. For this calculation an average hourly wage of $40 of a developer was used, making this a conservative calculation.

Dealing with Antipatterns

Now that the basic benefits of using continuous integration have been explored, what are some of the specific ways teams can remove waste in their process using this technique? One effective way is to identify and address the antipatterns within the software delivery process. Antipatterns are ineffective or counterproductive practices that produce more bad consequences than beneficial results. Following are several antipatterns commonly seen in the build and test phase of software development.

Antipattern #1: Suboptimal Branching and Merging

One common antipattern is *Suboptimal Branching and Merging* practices. In large projects there is usually a great deal of parallel development activity which generally leads to lots of branching. Merging these branches later can be a painful process. In many cases, developers defer the process as long as possible and as a result, the process becomes even more difficult and complex as branches become more disparate. One of the key ways to alleviate this problem is to practice testing and integrating more frequently, and to use a continuous integration server to build the software automatically once that integration has taken place.

Antipattern #2: Long Test Cycles

Long Test Cycles are usually due to a lack of automation, once the development team passes the work to the quality team. The quality team may conduct manual testing and use spreadsheets to document the outcomes. This often results in long periods of quality assurance and manual testing, which ultimately impacts the delivery of the release. Or even worse, testing can become almost an afterthought, which without doubt will negatively impact the quality of the delivered product.

To avoid these problems, a testing framework should be incorporated at the outset so that the build process includes a variety of testing: unit testing, integration testing, functional and nonfunctional testing, and acceptance testing – with a feedback mechanism built in. This testing framework should also include thresholds for testing. For example, for unit testing there should be a minimum of 80 percent coverage. If it drops below that percentage, then the build should be marked bad and the coverage should be improved.

Antipattern #3: The Dependency Maze

A *Dependency Maze* can manifest itself in different ways. For example, especially common in smaller organizations, the software delivery process can be less disciplined. Developers often build on their own workstations instead of using a common, shared server that helps to ensure consistency of production. When code is built on different 'bare metal' machines most likely there will be differences in libraries, which can cause the release to break when it goes into production. Another problem teams have when they are not using a central code repository is that they have difficulty tracing the source code to the binary running in production. With continuous integration, traceability is shown within the build. It allows the team to see what revision of the source code was used in the build to produce a particular release of the binary.

When a commit or change set that is pushed out into code fails in a particular function, the problem usually shows up in regression testing. A central continuous integration server helps to address this issue and minimizes the impact of dependencies. Final commits to the repository are tested in the central build server and once completed, leave an audit trail. This lets the team see who committed that particular revision of the source code and how it relates to the binary releases. It will also show all of the work from binary to the source code and who made each of the changes.

CollabNet Teamforge for Continuous Integration

By now the reader should have developed an appreciation for the benefits of continuous integration, and the critical elements that automation repeatability and traceability play.

From a technical perspective, compiling an infrastructure for continuous integration requires close integration of build- and test servers, bug and change issue tracker, file release management software, collaboration and reporting tools, and so forth. While manual integration is possible, that approach is typically error-prone, time-consuming and (over time) prohibitively expensive.

Integrated Applications Lifecycle Management (ALM) platforms can help, by providing a management and orchestration framework for Agile development and continuous integration.

CollabNet Teamforge in particular provides *application lifecycle management with full traceability, governance and process compliance* from planning through release. It contains all the tools developers need, integrated into *one shared framework*. TeamForge lets teams embed and synchronize their favorite point tools, whether they are proprietary or open-source. Teams can snap-in tools installed on a separate server or hosted in a

Scaling Continuous Integration

CollabNet was asked by a Fortune 500 bank to create a build management service that provided build tools and services to a community of 20,000 developers. Today there are about 80 to 100 groups that are using this build service. The outlay for the bank was minimal, because the service was built using best of breed open source tools. These open source tools were incorporated into the TeamForge framework so they look, feel and perform like one integrated application. CollabNet Lab Management enabled the service to be scaled and elastically provisioned across the community.

private or public cloud. With this flexibility, TeamForge still guarantees that central orchestration and traceability across the entire application delivery process is maintained.

TeamForge simplifies the continuous integration process, because by default users get a central repository for source code and all associated artifacts, using Subversion or Git. The repository assures that code is centrally managed, secure and is consistently and automatically backed up according to corporate specifications. Also, it can be securely accessed by globally distributed teams.

With TeamForge, developers have clear visibility into requested code changes, directly from within their preferred IDE or desktop. They can see the history of a work item, including all code changes, builds and tests. They can also review change requests, update code and commit changes to the code repository. (Screenshot 1)

When a build fail, it's critical for developers to be notified right away. TeamForge lets developers receive automated notifications via email or alerts directly within their IDE. In addition, the system assists in the root cause analysis by pinpointing log files and responsible servers. If issues are discovered in production, IT Operations can select earlier, stable builds from Lab Management's project build library.

CollabNet TeamForge integrates with Hudson and Jenkins build servers, so teams and enterprises can use Hudson and Jenkins within the framework of application lifecycle management. (Screenshot 2) This integration enables teams to be up and running quickly and easily access project information. Role based access control (RBAC) allows project administrators to securely delegate tasks within a project, including tasks within the build process. Workflows can be automated and created to assign broken builds to specific people within a development team.

TeamForge orchestrates the build and test processes and automates traceability and governance rules. By integrating build tools like Hudson and Jenkins into TeamForge's common governance framework, project members are ensured transparency and process compliance. Unit testing and builds can be run automatically. Once they are complete, team members can trigger additional tests like functional tests, integration or performance tests.

For enterprises with multiple distributed sites and code servers maintaining a local backup and disaster recovery can be costly and error-prone. CollabNet provides hosting services, including cloud storage and backup that can help drive efficiencies while improving data security. To improve cycle-times and ensure conformity with IT standards, teams can securely integrate public and private clouds for build and test.

Hudson and Jenkins Build Servers

The most popular build servers in the market today for continuous integration are Hudson and Jenkins.

Open source products: Hudson and Jenkins are free to download and easy to install; teams can have them installed and running within thirty minutes. They have widespread developer adoption and a strong community base.

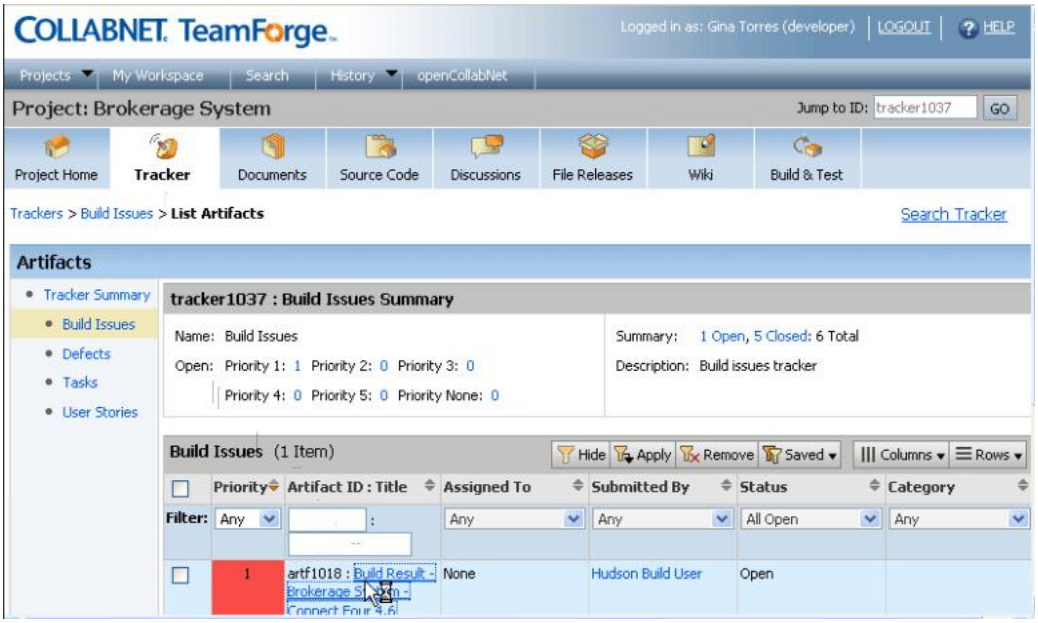
Highly Extensible: Hudson and Jenkins each have over 400 plug-ins and integrations that include the majority of test frameworks and build tools in the industry. Subversion is a default plug-in for both servers.

Highly Scalable: Hudson and Jenkins are highly scalable in distributed environments. Based on a master / slave architecture, they can scale across build servers with different operating systems and build stacks. All of the builds can be centralized on the master, and slaves can be continually added to the build architecture.

TeamForge Screenshot 1:

Automate Build and Delivery Traceability

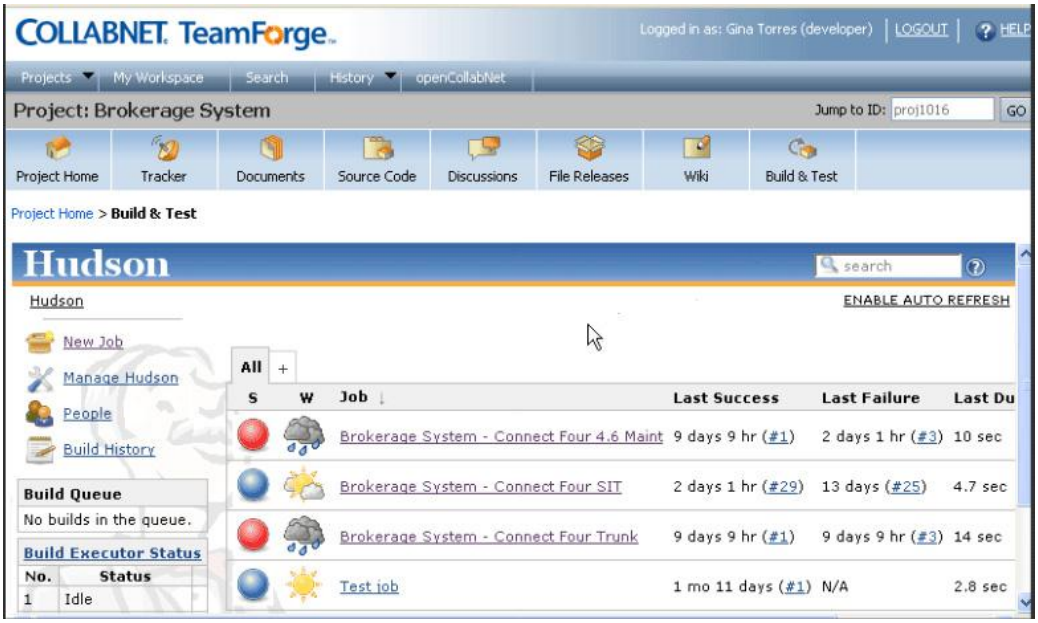
Improve build through release automation and traceability by automatically updating tracker status, or generating tracker defects, based on build and test results. Track delivered file releases back to builds, code changes and requirements.



TeamForge Screenshot 2:

Integrate Hudson or Jenkins

Easily integrate Hudson or Jenkins using certified plugins. Or integrate any other build tool to implement your own continuous integration and delivery capability using TeamForge's open API's with CollabNet Connect.



CONTACT US
Corporate Headquarters
8000 Marina Blvd, Suite 600
Brisbane, CA 94005
United States
Phone: +1 (650) 228-2500
Toll Free: +1 (888) 778-9793

Topics trending now

Many of the latest technology announcements have implications for PaaS and cloud development that will serve agile businesses everywhere.

- Enterprise Cloud Development, www.collab.net/ecd
- Continuous Integration, www.collab.net/getci
- 5 Things your Development Team need to be doing now, www.collab.net/5things

Summary

Continuous integration is a development practice tied to the concepts of "eliminating waste" and ensuring "rapid feedback" Teams practicing continuous integration in the build and test process integrate their work frequently, often multiple times per day. Each integration is verified by an automated build. Rapid feedback is built into the process, so that developers are automatically notified when a build fails so that they can resolve it quickly. Many teams find that this approach minimizes integration problems, builds value more quickly by eliminating waste and accelerating the overall release cycles. Teams implementing continuous integration will find that those advantages are well worth the investment, as product quality and return on investment will remain at the forefront of the software delivery process.

About the Authors

Darryl Bowler, CollabNet Technologist and Architect

Darryl leads the CollabNet consulting services business specializing in SDLC build automation, Continuous Delivery and DevOps. His career spans over 15 years in IT, working extensively with Fortune 500 customers designing complex systems customized to their business and technical needs. Darryl's accomplishments include large enterprise software build and deployment architectures in financial services and the federal government and delivering high-value virtualization / cloud and test lab automation consulting services to major enterprises.

His focus throughout his career has been mainly on emerging technologies, and he has extensive experience in network infrastructure, IT security, system management, engineering and software product development. His expertise includes visualization technologies, cloud computing and agile best practices. He has an MBA from Edinburgh Business School (Heriot-Watt University)

Shari Gardner, CollabNet Product Marketing Manager

Shari has over 20 years product marketing experience in the enterprise software industry working with high-profile technology companies including CollabNet, Sun Microsystems, Lotus Development and Banyan Systems. Shari holds an MBA from Babson College, Wellesley, Massachusetts and an MFA from the University of California at Berkeley.

About CollabNet

CollabNet is a leading provider of Enterprise Cloud Development and Agile ALM products and services for software-driven organizations. With more than 10,000 global customers, the company provides a suite of platforms and services to address three major trends disrupting the software industry: Agile, DevOps and hybrid cloud development. Its CloudForge™ development-Platform-as-a-Service (dPaaS) enables cloud development through a flexible platform that is team friendly, enterprise ready and integrated to support leading third party tools. The CollabNet TeamForge® ALM, ScrumWorks® Pro project management and SubversionEdge source code management platforms can be deployed separately or together, in the cloud or on-premise. CollabNet complements its technical offerings with industry leading consulting and training services for Agile and cloud development transformations. Many CollabNet customers improve productivity by as much as 70 percent, while reducing costs by 80 percent.

For more information, please visit (www.collab.net).

CollabNet, Inc.
8000 Marina Blvd.,
Suite 600
CA 94005

Tel +1 650 228 2500
Fax +1 650 228 2501
www.collab.net
info@collab.net

Blog blogs.collab.net
Twitter twitter.com/collabnet
Facebook www.facebook.com/collabnet
LinkedIn www.linkedin.com/company/collabnet-inc