Continuous Integration
Northwestern University

Evanston, Illinois
November 2012

Top 6 reasons students say they join the Pariveda team

1. **Clearly defined career path** shows you where you will be in 5 years
2. **Limited travel** allows you to better connection with your local market
3. **Dedicated training and mentorship programs** provide a strong support system during career progression
4. Committed to the **core value of servant leadership** and dedicated to making a difference in the local community
5. Wealth of ***opportunity*** in a growing technology business
6. Operations located in **major cities across the US**

While I was going through the interview process with Pariveda, I quickly realized that it was a place I could make an immediate impact at. After my first year, I can look back and say that this is absolutely correct. I not only feel like an asset to the company, but I have also learned a ton thanks to my mentors and peers.

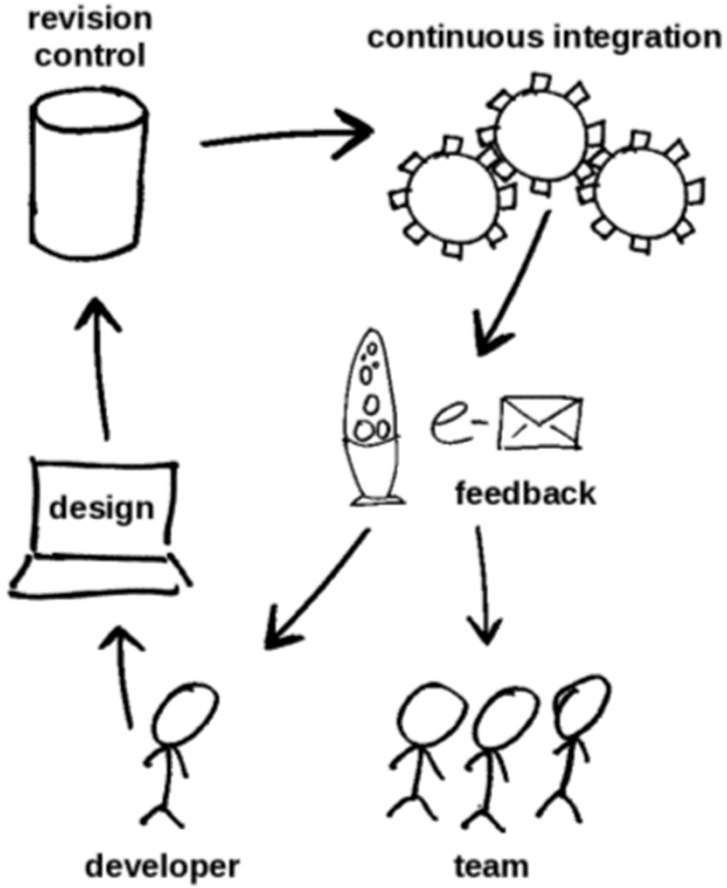
Brett Hlavinka, Texas A&M Class of 2011

Table of Contents

Continuous Integration is the practice of frequently integrating one's new code with the existing code repository

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) 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]

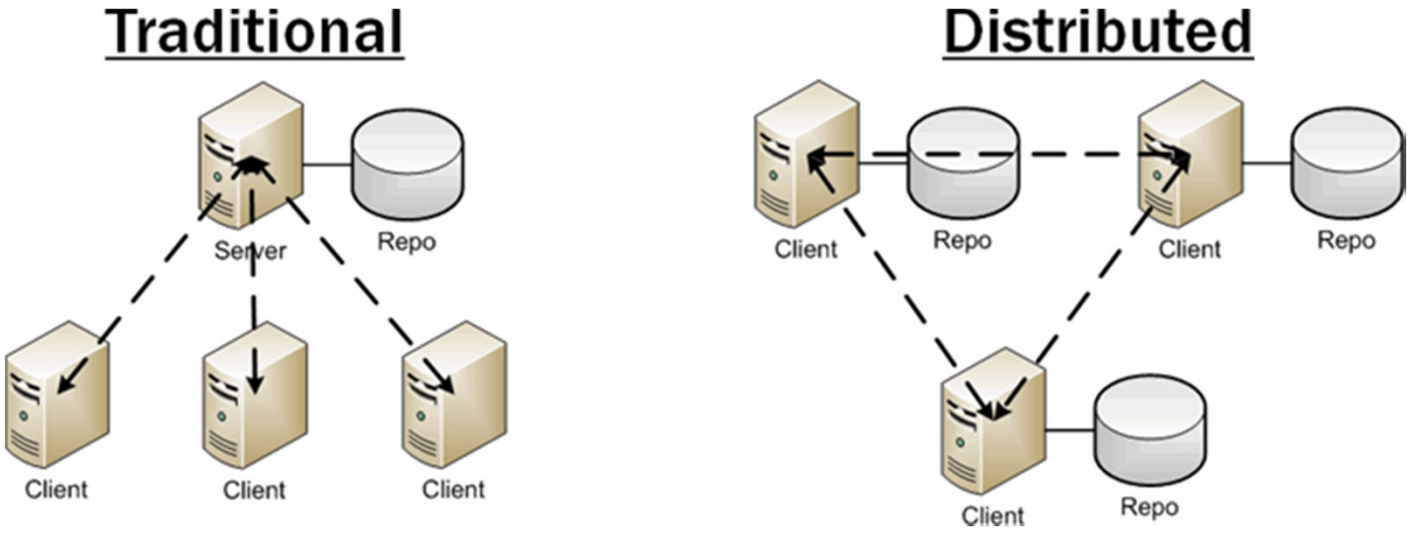
Martin Fowler



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

Source Control 101

- Most software projects require a *team* of developers
- These developers all contribute to the effort by checking code in and out of a source control repository
- Many different tools can be used for source control (such as: SVN, Git, TFS, Perforce, ~~Visual Source Safe~~, etc.)
- When a developer cannot build the latest revision of code checked into the source code repository, the build is said to be “broken”



Why use Continuous Integration?

- On each check-in, CI
  - Attempts to compile code
  - Automatically runs unit tests
  - Checks test coverage of the application
  - Can perform any number of other checks: cyclomatic complexity, check naming conventions, looks for copied and pasted code, etc.
- Continuous Integration gives you an extra team member
  - Prevents "it works on my machine" situations
  - Allows you to continuously make builds without a build manager
- Continuous Integration enforces build consistency
  - Ensures everyone is building the application the same way
  - Ensures the application is built correctly every time
- Continuous Integration immediately identifies when a build is broken
  - What is broken (won't compile or failing test)
  - Who broke it (enables you to quickly ask questions and help the team fix the build)
- Continuous integration makes your application **alive** instead of just static code sitting in a repository

Continuous Integration is part of a bigger landscape, and can often imply other tools and processes are at play

- Automated Tests
  - Although a CI tool could be set up to just compile on check-in, this does not make full use of the CI concept
  - Any code added to the repository should have unit tests written against it to make sure it is functioning as expected
  - Any existing code that has been modified should have its unit tests updated before it is checked in
  - Having up-to-date unit tests helps developers ensure that they are not breaking existing code while making updates
  - Tools: NUnit, MSTest, Test Driven.net, Rhino Mocks (Unit testing and mocking frameworks are available for just about any language)

- Code Coverage Reporting
  - Analyze and enforce how well source code is covered by unit tests
  - Tools: TFS, NCover, PartCover
- Code Style Enforcement
  - Description: Enforce coding standards (placement of braces, comment formatting, etc.)
  - Tools: TFS, FxCop
- Code Metrics
  - Analyze Lines of Code, Cyclomatic Complexity, Coupling, Nesting Depth, etc.
  - Similarity analysis to detect when a developer copies and pastes code
  - Code metrics tools include: Vil, NDepend, Similian
- Issue or Feature Tracking System
  - Any sizable project needs a formal feature and issue tracker
  - Feature tracking systems include: Redmine, Rally, JIRA, BugZilla, FogBugz
  - Many feature trackers can be integrated with your Continuous Integration tool

There are many Continuous Integration tools available; decide which features your project requires and choose the one that suits you best

- Hudson / Jenkins
  - Open source (Oracle owns Hudson and Jenkins is a fork of Hudson)
  - Massive amount of plugins available
  - Supports most platforms
  - Runs on both Windows and ‘Nix platforms

- Buildbot
  - Open Source (MIT License)
  - Used by Google
  - Written in Python
  - Runs on both Windows and ‘Nix platforms

- Go
  - Per-User License
  - Manages your release process with workflows

- Team City
  - Build-Agent-Based License
  - Runs on both Windows and ‘Nix platforms

Martin Fowler, Chief Scientist at ThoughtWorks, has written extensively about the benefits of Continuous Integration

- Martin Fowler is an author and international speaker on software development, specializing in object-oriented analysis and design, UML, patterns, and agile software development methodologies, including extreme programming

- For a long time, Fowler has been a champion of Continuous Integration
  - “We’ve found CI to be a core technique at ThoughtWorks and use it almost all the time”

According to Martin Fowler, CI is comprised of 10 key practices

1. **Maintain a Single Source Repository**
   - *store all code required for a build in source control*
2. **Automate the Build**
   - *have an IDE-independent build server that can build and launch a system from a single command*
3. **Make Your Build Self-Testing**
   - *write a suite of automated tests that can check a large part of the code base for bugs (not necessarily TDD)*
4. **Everyone Commits to the Mainline Every Day**
   - *integration is about communication, by committing frequently developers can identify conflicts quickly*
5. **Every Commit Should Build the Mainline on an Integration Machine**
   - *maintain a continuous integration server that monitors the repository for changes to trigger builds and send notifications*
6. **Keep the Build Fast**
   - *keep builds slim, if necessary create a two-stage build pipeline for isolated compilation and testing*
7. **Test in a Clone of the Production Environment**
   - *have at least one environment that duplicates the production environment*
8. **Make it Easy for Anyone to Get the Latest Executable**
   - *anyone involved with a software project should be able to get the latest binary*
9. **Everyone can see what's happening**
   - *extremely important to convey state of the mainline build; can be done with website, desktop alerts, lava lamps*
10. **Automate Deployment**
    - *not necessarily to Production, have scripts to deploy applications quickly. also consider tools for rollback approach.*

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

In order to help you feel comfortable, here are some major companies who use Continuous Integration

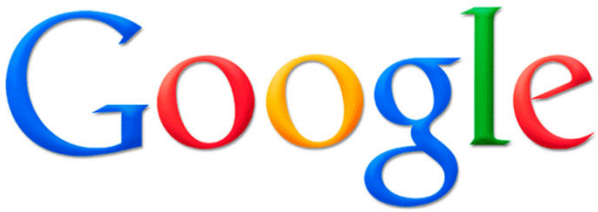


Table of Contents

The most wide ranging benefit of Continuous Integration is reduced risk

- The trouble with deferring integration is that it's very hard:
  - To predict how long it will take to complete the integration
  - To see how far you are through the process

- You are putting yourself into a complete blind spot right at one of the most tense parts of a project - even if you're one of the rare cases where you aren't already behind schedule

- Continuous Integration completely handles this problem
  - There's no long integration, so you completely eliminate the blind spot
  - At all times you know
    - What state the system is in
    - What works and what doesn't work
    - The outstanding bugs you have in your system

- Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove
  - If you introduce a bug and detect it quickly it's significantly easier to get rid of
  - Since you've only changed a small part of the system, you don't have far to look
  - Since that part of the system is the part you just worked with, it's fresh in your memory

- Projects with Continuous Integration tend to have dramatically fewer bugs, both in Production and in process
  - The degree of this benefit is directly tied to how good your test suite is
  - Getting there means constantly working on and improving your tests

Continuous Integration removes one of the biggest barriers to frequent deployment

- Frequent deployment is valuable because it allows your users to
  - Get new features more rapidly
  - Give more rapid feedback on those features
  - Become more collaborative in the development cycle

- This helps break down the barriers between customers and development

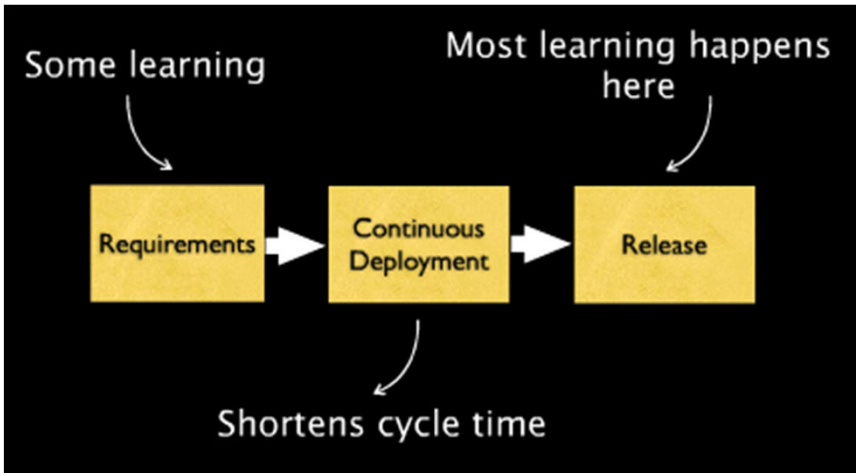


Table of Contents

The full set of practices outlined above give you the full benefits, but you don't need to start with all of them

- Perhaps the best way to establish a sustainable CI environment is to build it incrementally

- One of the first steps is to get the build automated
  - Get everything you need into source control
  - Make it so that you can build the whole system with a single command
  - For many projects this is not a minor undertaking, yet it's essential for any of the other things to work

- Introduce some automated testing into your build
  - Identify the major areas of your system where things go wrong and get automated tests to expose those failures
  - On an existing project it's hard to get a really good suite of tests going rapidly
  - It takes time and diligence to build tests up

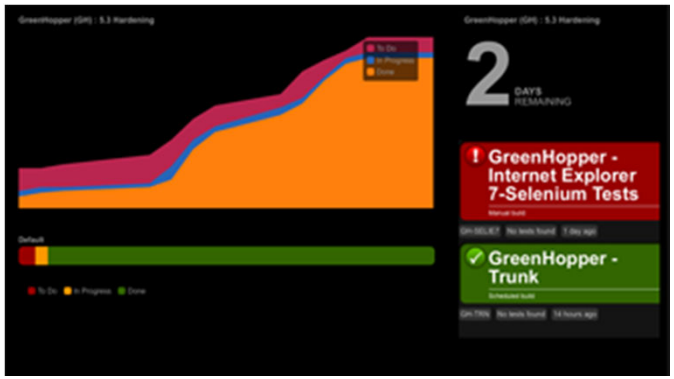
- Try to speed up the commit build
  - Continuous Integration on a build of a few hours is better than nothing, but getting down to ten minutes is much better
  - This sometimes requires serious surgery on your code base as you break dependencies on slow parts of the system

- If you are starting a new project, begin with Continuous Integration from the beginning
  - Keep an eye on build times and take action as soon as it starts going longer than ten minutes
  - By acting quickly you'll make the necessary restructurings before the code base gets so big that it becomes a major pain

- Above all, get some help
  - Find someone who has done Continuous Integration before to help you
  - Like any new technique, it can be hard to introduce it when you don't know what the final result looks like

Establishing the right team culture is more important than the Continuous Integration toolset you choose

- Continuous Integration is much more than just a collection of tools and scripts
  - It's a practice, a way of doing something, and it has to be part of your working culture to be truly effective
  - It's all too easy to have a wonderful CI system, and then have it ignored unless there is the right level of buy-in from the people who this system is meant to cater to – Developers, Testers, and Management



- Team members must take **shared ownership** of their development and build process
  - Reliance on CI tools and other mechanisms isn't a substitute for discipline

- One way to encourage shared ownership is to openly display various source code quality metrics and trends, such as:
  - Code coverage
  - Cyclomatic complexity
  - Coding convention violations
  - Version control activity
  - Bug counts

- Influence the right behavior in the team by displaying an **Information Radiator** in the team area
  - Provide clear visual feedback about the build status

Get the attitude right, and the rest will follow

- Continuous Integration is people collaborating to achieve a shared goal
  - Continuous Integration is primarily a human practice
  - If you can't do it **without** tools, you're not really doing it!

- Broken builds must be **rapidly fixed** by developers
  - Fixing the build should take priority over most everything else - meetings, lunch, work on the next task

- Don't punish the person who checks in the most and therefore has the most broken builds, even if s/he breaks that build at the same rate as other people who check in less often
  - Rather, shame the person who breaks the build and goes home for the night

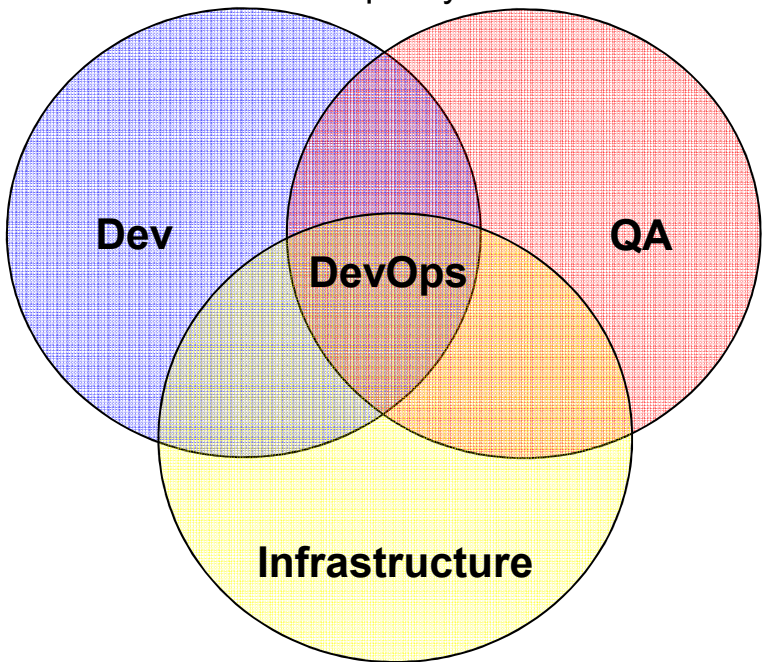
DevOps is a marriage between Development and Infrastructure to make sure both sides have a say in tools and best practices

- There must always exist a clear separation between the development and infrastructure teams
  - The teams should always work closely, and have good communication between them
  - Both Infrastructure and Development personnel may serve multiple project teams
  - One person cannot be on both teams

- Automation of deployment, testing, and other routine activities have brought about new tools that are often shared by development and infrastructure teams

- Because distinctly different sets of people use these tools, they should all be represented during tool selection

- A DevOps council consists of one member of each development team, representatives from the infrastructure team, and representatives from the quality assurance team



Links for setting up a Git repository and Jenkins

GitHub SSH Keys
http://help.github.com/win-set-up-git/

Git Ignore
http://help.github.com/ignore-files/
http://juststuffreally.blogspot.com/2008/10/if-you-have-file-that-lives-in-git.html

Unit Testing (NUnit)
http://yakiloo.com/jenkins-tfs-and-msbuild/
http://www.nunit.org/index.php?p=consoleCommandLine&r=2.2.4

Remote Notifications (CCTray)
https://wiki.jenkins-ci.org/display/JENKINS/Monitoring+Jenkins

Code Coverage (Part Cover)
http://yakiloo.com/jenkins-tfs-and-msbuild/

Feature and Issue Trackers
http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems

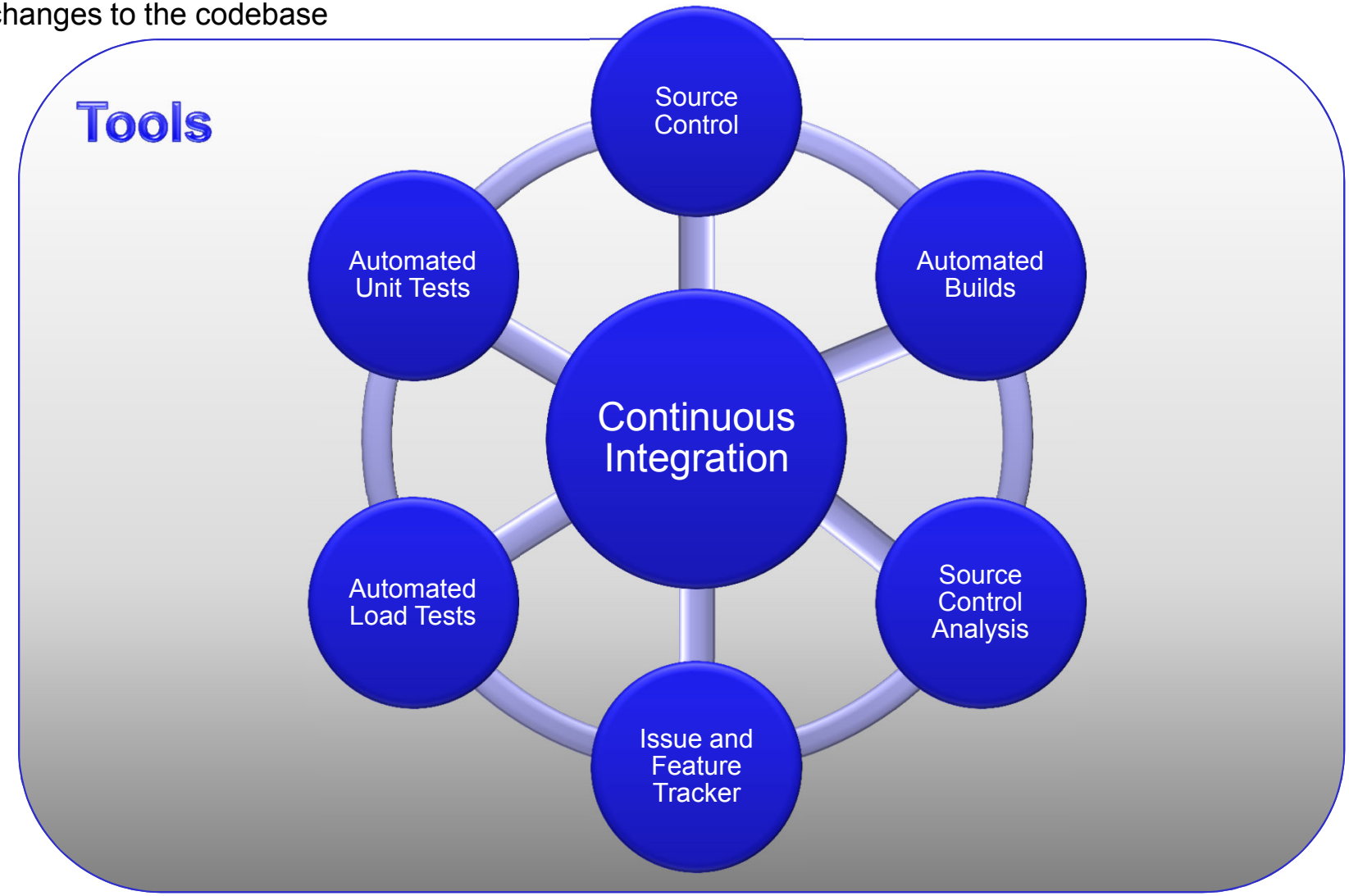
Contact Info

Andrew Bose
andrew.bose@parivedasolutions.com
Akhil Patel
akhil.patel@parivedasolutions.com
Daniel Posada
daniel.posada@parivedasolutions.com

Appendix

Continuous Integration (CI) serves as a hub and ties the most critical developer enablement tools together

- Continuous integration tools give us the ability to tie together many critical developer tools and display them in an easy-to-use dashboard
- Via the CI dashboard, a development team can view the health of the build, monitor code coverage, and changes to the codebase



Equipping your team with developer enablement tools can prevent the enterprise from wasting millions of dollars annually

- Technology research firm, IDC conducted a survey of 139 companies in North America
  - Ranged in sizes from 250 to 10,000 employees
  - Over 50% had over 1000 employees

- IDC reported that these companies had annual debugging costs of
  - **$5.2M** (midpoint-mean 100 IT employee organization)
  - **$22M** (midpoint-mean 416 IT employee organization)

- Dr. Laurie Williams of North Carolina State is one of the leading authorities on quantifying benefit from modern developer enablement tools
  - Dr. Williams concentrates specifically on Test Driven Development and Automated Unit Testing
  - Dr. Williams has conducted multiple case studies in various languages with Microsoft, IBM, and other smaller development groups
  - Research from IBM and Microsoft case studies shows that by implementing automated unit tests, the number of defects can be reduced by between **62% and 91%**

- This means that dollars wasted annually on debugging could be reduced
  - From **$5.2M** to between **$0.5M and $2.0M** (for a ~100 IT employee organization)
  - From **$22M** to between **$2.0M** and **$8.4M** (for a ~416 IT employee organization)

- If this core tenet of DDA will reduce defect density by 50%, for an enterprise smaller than the those represented by the survey (a company that spends $1.0M on defect mitigation) that enterprise would save $500K that could be used to add value elsewhere

One way to motivate developers to keep the build healthy is the Wombat of Shame Game

- Replace “Wombat” with your stuffed animal of choice

- The rules are as follows:
  - When the build breaks, the person responsible is handed the stuffed wombat
  - The person responsible should stop any activities, even urgent, and start to fix the build
  - Nobody can commit anything to the repository until the build is fixed
  - All team members have to wait until the build is fixed or incorrect changes are reverted
  - The next time the build breaks, the current wombat-holder must deliver the wombat to the new breaker

- What does this do?
  - It’s always clear who should be fixing the build – whoever has the wombat at his desk
  - Not wanting a stuffed wombat on your desk makes you extra diligent
  - It adds some fun and humor to the CI process

- Treat each broken build as an opportunity for the whole team to learn, so there's less chance of anyone making the same mistake again

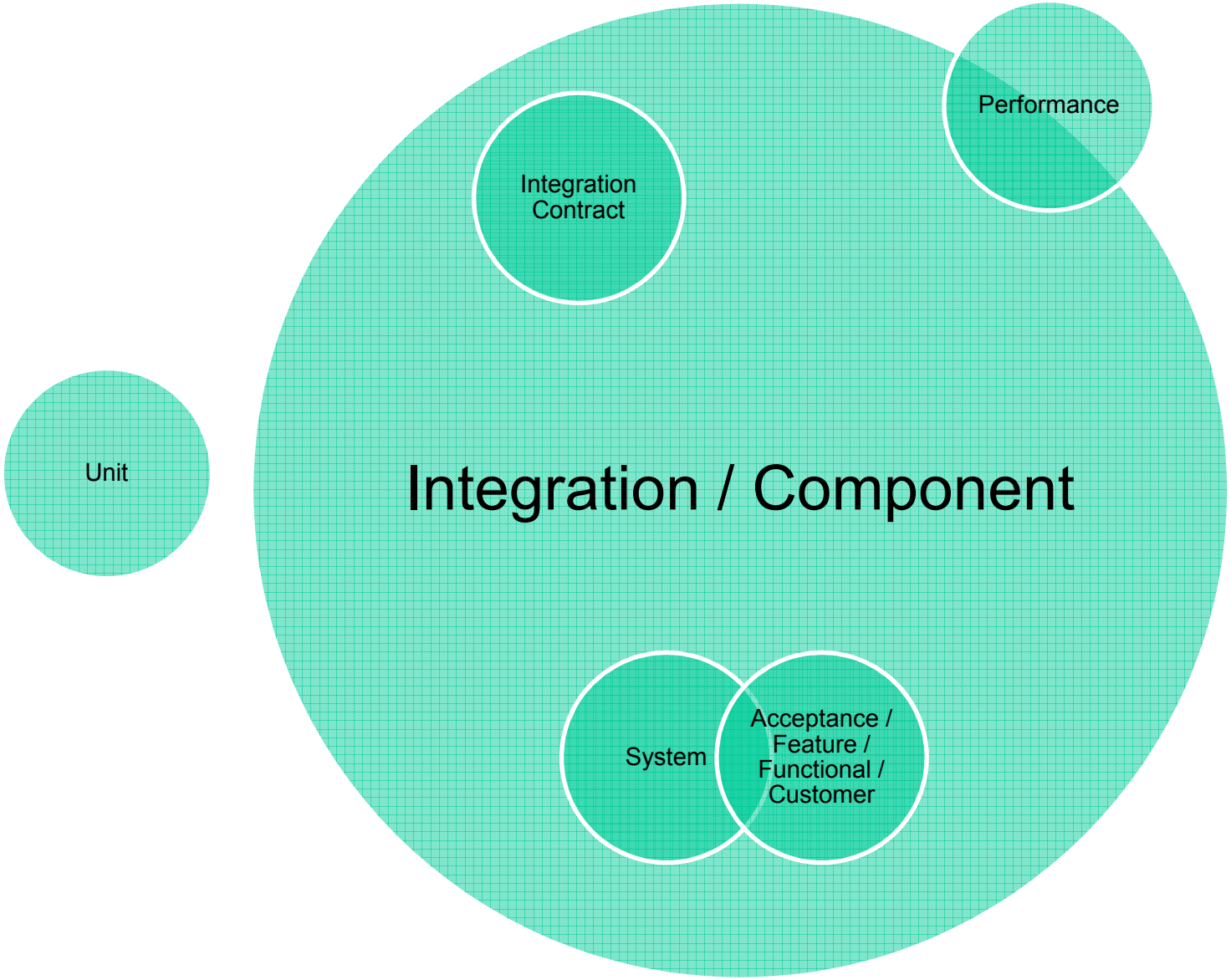
Many problems late in a project boil down to a lack of up-front integration testing

- Post-facto, manual, GUI-based functional tests (despite their enormous cost) do a lousy job of exercising all the paths through a system's business logic
- It's easy for such testing to miss parts of the system, so the team got poor feedback about whether all the features were being fully delivered as required
- The team did not begin running integration tests early enough, and throughout the project, the team ran integration tests too infrequently (or not at all)
- Integration tests can give us feedback **very early in** and **very frequently throughout** the project

We will define an integration test as a test that checks the interactions between at least two major components of a software system

Term	Definition
Integration / Component	Checks the interactions between at least two major components of a software system
Unit	• A "unit" is the smallest piece of code software that can be tested in isolation • "In isolation" means separate and apart from the application, and all other units
Acceptance / Feature / Functional / Customer	A formal test conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system
Integration Contract	• Checks that all the calls against your test mocks return the same results as a call to the external service would • A failure in any of these integration contract tests implies you need to update your test doubles, and probably your code
System	Conducted on a complete, integrated system
Performance	Determines how a system performs in terms of responsiveness and stability under a particular workload

Some tests fall into more than one of the aforementioned categories



Integration tests have advantages over many kinds of black-box ad-hoc functional tests

- Integration tests can give us feature feedback very early in the project - in fact, the tests can be written first, so programmers can code to the tests

- Integration tests can give us feature feedback very frequently
  - They can be run manually or automatically by anyone with access to the tests, as frequently as required
  - Every week, every day, every hour in a crunch

- They are deterministic: they either run **green (pass)** or **red (fail)**
  - If they run green for a given requirement, either the requirement is done and we move on to others, or the set of tests is not yet exactly right, in which case we refine them
  - Either way, we are successively refining the system in an orderly way
  - With each new test running green, we can all see the system getting better, more valuable, closer to what we need

- Being based on example data, integration tests exercise more paths through the business logic - when you run pre-defined integration tests, you run less risk of missing important features or functional behavior

There are many powerful languages available for encoding integration tests

Tool	Description	Use When You Want…
FitNesse	Open-source framework used to: • Collaboratively define web pages containing simple tables of inputs and expected outputs • Run those tests and see the results	• Collaboration between developers, testers, and business analysts • Deterministic comparisons of customers' expectations to actual results
Gherkin (Cucumber / SpecFlow)	Business Readable, Domain Specific Language that lets you describe software's behaviour without detailing how that behaviour is implemented	• Combined documentation of business requirements and automated tests • A natural language encoding of the test scenarios
xUnit	Allow testing of different elements (units) of software, such as functions and classes	• Testing of different elements (units) of software, such as functions and classes
Concordion	Lets you write tests in plain English using paragraphs, tables and proper punctuation	• Highly readable tests • Clear separation of tests from implementation
Twist	Designed from the bottom up to make test authoring and maintenance clear and efficient	• A bridge from manual to automated testing • Easy reuse and refactoring of tests to produce new or altered versions
Homegrown Domain Specific Language	Programming language or specification language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique	• To express a particular type of problem or solution more clearly than an existing language or tool would allow

Note: This is not an exhaustive list.