Jenkins

Presented by Daniel Hinojosa on behalf of Develop Intelligence

dhinojosa@evolutionnext.com

Continuous Integration

- Taking code and integrating and testing with every check in!
- Employs Continuous Integration (CI) Server
  - Jenkins
  - Bamboo
  - Cruise Control
  - Travis

Continuous Integration

- Tool that monitors version control of changes
- When a change is detected, the tool will automatically
  - compile
  - test
  - report
- Overtime reports on the overall code quality of your project.

Purpose of CI Server

- Configured to watch your version control system
- Check out or update your source code every time a change is made
- Run the automated build process
- Store the binaries where they are accessible to the whole team
- Reducing risk by providing faster feedback
- Identify and fix integration and regression issues faster

How does CI improve software quality?

- Everyone has information on the build at every single moment
- Developers are aware of the constant change in their project
- Notifications to all Developers when the build fails

Provides Value to the End User Faster

- Ensures that software is build, tested, and maintained regularly
- There is always a deployment of some kind
- Deployments are no longer magical mystery events that happen every month

Empowering Teams

- Continuous Integration can deploy onto the servers themselves.
- Power to real time view results
- There will be no longer the uber-operations-specialist
- Expand the role of the developer include operations.

Reducing Errors

- Ensures the correct version, configuration, database schema, etc. are applied the same way every time through automation
- Staging areas are equivalent to production areas
- Less surprises.
- No more hidden configuration that we don’t know about at production time

Reducing Errors

- Ensures the correct version, configuration, database schema, etc. are applied the same way every time through automation
- Staging areas are equivalent to production areas
- Less surprises.
- No more hidden configuration that we don’t know about at production time

Lowers Stress

- A Release becomes commonplace without the typical stress
- No weekend battles
- Less embarrassing releases
- Rollback versions easily to a well known build

Deployment Flexibility

- Instantiate a new environment or configuration by making a few changes to the automated delivery system.
- Create multiple version of environments

Practice Makes Perfect

- Final deployment into production is being rehearsed every single time the software is deployed to any target environments.
- Overtime deployment becomes standard practice
- Check in frequently
- Don’t check in broken code
- Don’t check in untested code
- Don’t check in when the build is broken
- Don’t go home after checking in until the system builds
- Every Check in is a potential release!

Will it happen overnight

- It will take time for all those invested
- Testing will have to be commonplace
- Commitments in code will have to be small
- Behaviors and rituals will need to change

How does CI help release on time?

- Automation, Automation, Automation, Automation
- Humans are slow, why not let computers do the work?
- Problems no longer are debt: If anything fails it must be fixed immediately
- Bugs, along with testing, are squashed oftentimes permanently!
- Lessens the fear of "integration hell"
- Lessens the fear of production deployment!

Levels of Testing

Unit

- Isolated and Repeatable Tests
- Best when employed through TDD
- Small and Fast (It is isolated)
- First line of defense against bugs in CI environment
- No I/O, No reaching out to external resources
- Use of Stubs, Mocking, Functions to develop tests since we do not want to employ real world objects at this particular stage

Integration

- Occurs after unit testing
- Aggregates the components that have been developed using unit testing
- Verify functional aspects of the code
- I/O testing capability
- Integrate Components of Focus, the rest either stub or mock away

Acceptance/Functional

- Black box testing
- Consider your system as a whole that takes input and returns an output
- External Acceptance Testing - Not employees of the organization that develop the software.
- Internal Acceptance Testing - Member or employees that are not entirely familiar with the project
- Use or employee testing structures like FitNesse, Cucumber
- Are you delivering what was promised?

Performance Testing

- Load Testing - How well does the product fare under heavy load
- Stress Testing - Included with Load Testing, how well will your product survive under extreme conditions
- Soak Testing - How does the system perform over an extended period of time
- Spike Testing - How well does it perform with a spike in a short time span
- Configuration Testing - How well does your project perform with different configurations?

Which Tests Are More Important?

- Unit by far is the most important, it is the first line of defense.
- Other can be implemented on an as need or important to the customer basis!
- You may want to accept some failure in some tests (not unit)

Performance Testing within CI

- Don’t run every hour. Performance Testing is slow and methodical
- Distribute to an agent to handle the work
- Some will need a dedicated remote agent if it is vital
- Assertions are driven through acceptable thresholds.

Working With These Results

Crafting Ideal Plans Based on Tests

- What tests will be included at the onset?
- When do you want the test results?
- Which test results are most important?
- Smoke Tests - Favorite or important functional integration tests should run early on to gauge the health of the system

Test Driven Development

- Making assertions about your code before writing the code
- Use of stubs, mocks, or functions to isolate the test
- Not Easy
- Instant Coverage
- Provides faster feedback when integrated into CI because naturally there are more tests!
- Not QA's job for knowing what you are trying to do

Testing during Sprints

- Test during your code constantly
- Check in to the master/trunk branch daily, preferably multiple times a day. (Oh oh)
- Make small incremental changes through out the day
- Add value to commit messages
- Commit an atomic unit

Prefer Toggle Switching to Feature Branching

http://martinfowler.com/bliki/FeatureToggle.html

Continuous Deployment vs. Delivery

- Continuous Deployment
  - Take every build that passes testing into production
- Continuous Delivery
  - Any successful build be deployed into production via a fully automated one-click process
  - Stakeholders decide when the release is made <u>not</u> the engineers

Jenkins

About Jenkins

- Originally called Hudson
- Written in Java
- Open Source
- Supports multiple languages:
  - Java, Ruby, C#, Groovy, PHP, etc.
- Easy to Use
- Appealing
- Large Community
- Multiple Plugin Support
- Has LTS Support

The History of Jenkins

- Authored by Kohsuke Kawaguchi
- Started as Hudson
- Began in 2004
- Oracle acquired Sun in 2009
- In 2010, Oracle and the Developers argued on ownership
- In January 2011, the Hudson developer community decisively voted to rename the project to Jenkins

Thanks