Agile Methods

Module 4: Quality Development Practices

- Collective Code Ownership
- Continuous Integration and Testing
- Developer Best Practices in the Context of Agile
Collective Code Ownership

- The code for the project is owned by the entire project team. No one developer "owns" any of the code.
- While building new functionality, developers may modify any code in the system.
- Developers are responsible for maintaining all unit tests and writing new ones for new functionality.
- All the normal integrity preservation duties apply to changes to any code.
Continuous Integration & Testing

• What is Continuous Integration?
  – Integrate & build the system several times a day
  – Integrate every time a task is completed

• Continuous integration & relentless testing go hand-in-hand.

• By keeping the system integrated at all times, you increase the chance of catching defects early and improving the quality and timeliness of your product.

• Continuous integration helps everyone see what is going on in the system at all times.

If testing is good, why not do it all the time? (continuous testing)
If integration is good, why not do it several times a day? (continuous integration)
If customer involvement is good, why not show the business value and quality we are creating as we create it (continuous reporting)
Fowler’s 10 Best Practices for CI

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

1. Maintain a Single Source Repository
2. Automate the Build
3. Make your Build Self-testing
4. Everyone Commits Everyday
5. Every Commit should Build the Mainline on an Integration Machine
6. Keep the Build Fast
7. Test in a Clone of the Production Environment
8. Make it easy for Anyone to get the Latest Executable
9. Everyone can see what’s Happening
10. Automate Deployment
CI & Test

- Unit, System, & Integration tests run continuously!
  - Requires test automation and reporting framework
  - Post results to a dashboard for all to see
    - Daily standup starts by checking if the dashboard is “green”
- Report on your static analysis / metrics too!

Together with burndown charts, these show business value being built, with an attention to quality, at a sustainable pace.
Agile Dev Best Practices

• These you may already know:
  – Source Code Control & Change Management
  – Unit testing (and Test-Driven Development)
  – Static Analysis
  – Measures and Metrics
  – Refactoring

• How do these work together in an Agile world?
Agile Dev Best Practices

- **SCC** – DVC systems like Git support flexible workflows
- **Unit Test/TDD** works well with **Refactoring**
  - Red-Green Refactor
- **Static Analysis** has in some ways supplanted the traditional code review
- But there is still **Pair** and **Mob Programming**
  - Again, if it is good, “do it all the time”
- **Metrics** help identify systemic “smells”

The ability to define a quality policy and visualize/enforce it via your CI&Test platform creates Agile nirvana
Exercise

• Clone the code in your team’s repo
  – Run “ant junit” to see results
  – Review Junit reports in your browser

• Login to Jenkins https://swent0linux.asu.edu
  – Execute a build for your team
  – Review the unit test reports

• Make a change to pass a test
  – Push to Github
  – Re-build on Jenkins

• For the Brave: Red-Green-Refactor!
  – Add a new feature by first writing a new failing unit test
  – Make the test pass!