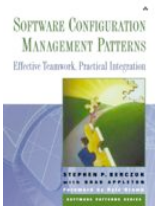
Getting Agile with Legacy Code

Steve Berczuk

About Me



Agenda

- Agenda
  - Agile and Legacy Code
  - Approach
  - Summary
- Goals:
  - Concepts
  - Strategy

Agile

- Principles
- Practices
- Feedback
- Pragmatic Approach
- Why?

The Agile Manifesto

- *Individuals and interactions* over processes and tools
- *Working software* over comprehensive documentation
- *Customer collaboration* over contract negotiation
- *Responding to change* over following a plan

What is Legacy Code?

- Hard to understand
- Hard to change
- Slow to build
- "Big Ball of Mud"
- No Feedback
- Age doesn't matter!

…Legacy Code

- Source Code
- DDL/SQL
- Build Scripts
- Installation Scripts

Common Legacy Code Issues

- Fear of change
- Difficult to write tests
- Difficult to verify
- Dependencies
- "Broken Windows"
- Challenge: In production

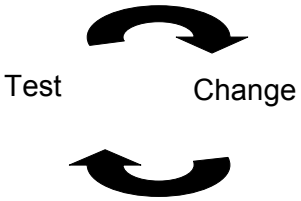
Agile Code Qualities

- Unit Testable
- Low-coupling
- Easy to Refactor
- Quick to build & test
- “Safe” to Change
- Feedback Mechanisms

Goal

- Transform Legacy Code to make it more robust to change
  - Build and test in a repeatable fashion.
  - Enable validation
  - Enable incremental change
- How?
  - Enable Feedback
  - Testing

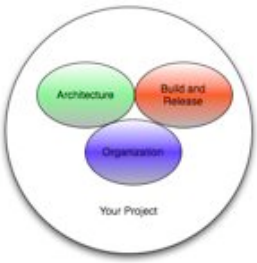
The Legacy Code Dilemma



The Art of the Possible

- Teamwork
- Common Goal
- Small Steps
- Basic Principles
- Discipline

Project Ecosystem & Success



Organization Issues

- Organization
  - Agile Means Feedback
  - Agile Exposes Problems
  - Organization influences Architecture
    - Conway's Law
  - *Customer Collaboration…*

Conway's Law

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

-Mel Conway
-"How Do Committees Invent?"
-*Datamation*, April 1968

Ecosystem: The Team

- Pair Programming / Commit Buddy
- Continuous Integration
- *Individuals and Interactions…*

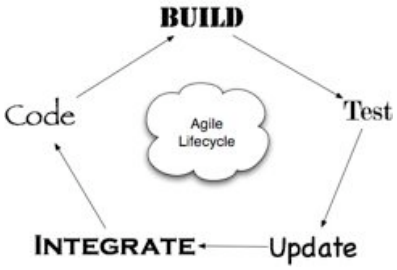
Architectural Issues

- Coupling / Modularity / Dependencies
- Testability enables stability
- Testability encourages modularity
- Influences Build and release strategy

Build, Release, Test

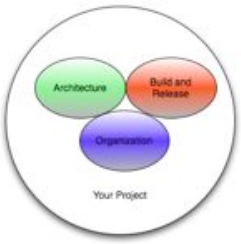
- Testing enables single code line
  - Establish a policy requiring testing
- Build enables automated integration
  - Quick builds enable rapid feedback
  - *Responding to change…*
- Release enables feedback
- *Working software…*

Agile Development Lifecycle



Ecosystem

- Not Just Code Changes
- Code "lives" in a context
- Agile practices reflect this



Enable Change: Feedback

- Working Software
  - Build-able (repeatable)
  - Tests define “done”
- Iterations
  - (Small steps, incremental change)
  - Feedback at each step
- Levels of Scale
  - Workspace
  - Integration
- Empower Team

Feedback

- Feedback enables change
- Tests provide feedback
- Build enables testing
- Confident change enables agility

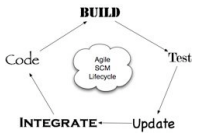
Repeatable Build

- Build = Component Architecture
- Repeatable = Testable
- Dependencies = Difficult to Test
- The Build Scripts *are* Code

Common Build Problems

- Build too slow
- Can’t create a workspace without manual steps
- Don’t know your dependencies
- Don’t have Continuous Integration
- Hard coded paths

Build Speed

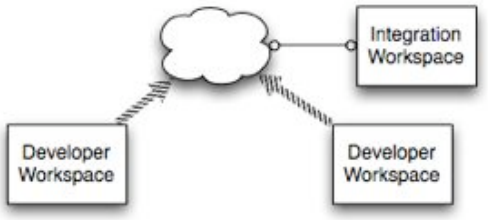


- Slow builds slow cycle & disrupt flow
- Module architecture

Build Speed

- Modules
  - Build and test at component level
- Hierarchy of tests
  - Balance completeness and time
- Fix Tests (& Build)
  - Tests are code too

Repeatable Process



Enable Agile Build

- Understand Dependencies
  - Explicit, not implicit
- Repeatable Structure
  - Create workspaces
- Metrics
  - Add tooling to build to measure coupling
- Testing (with each build)
  - Smoke Test

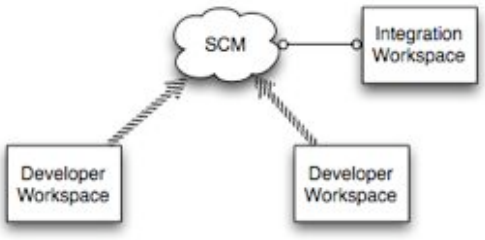
Dependency Management

```
<property name="lib.java" path="WEB-INF/lib"/>
<path id="build.classpath">
      <fileset dir="${lib.java}" includes="*.jar"/>
</path>
```


Dependency Management

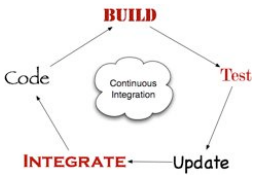
```
<artifact:dependencies pathId="build.classpath">
    <dependency groupId="junit"
        artifactId="junit" version="3.8.2"
        scope="test"/>
    <dependency groupId="commons-collections"
        artifactId="commons-collections"
        version="3.2" />
</artifact:dependencies>
```


Repeatable Workspaces

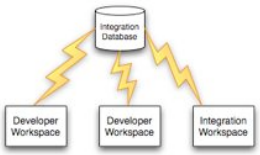


Continuous Integration

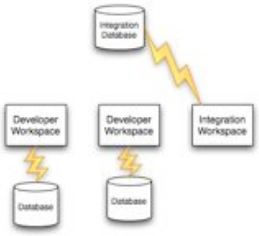
- Immediate
- Continuous
- Code Compiles
- Tests Pass
- Essential



(In) Flexible Configuration

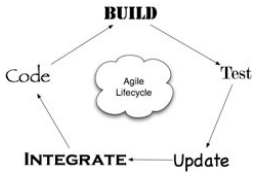


Flexible Configuration

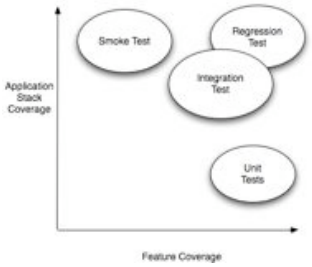


Progress

- Can build and deploy from a known state
- Clear module dependencies
- Identified incorrect & inflexible Configurations



Types of Tests



Enable Test Execution

- Run existing tests
  - Integration Tests
- Build
- Private Workspaces
  - Database Test Environments
  - App Servers
  - Will find config dependencies

Testing Impediments

- Global Dependencies
  - Singletons
- Coupling
  - Creating rather than parameters
  - Not Using Interfaces

Tools to Break Dependencies

- Dependency Injection
- Configuration
- Refactoring
- Fake Objects (For test)

Testing Risk Factors

- Multiple Tests for same thing
  - Increase cost of change
- Overuse of Mocks to hide complexity
  - Remove the complexity
- Excessive code changes enable tests

Tools To Enable Change

- Define what you want to change
- Verify that your change did not break anything
- Write new code to isolate
- Test Locally
- Test Globally
- Test Driven Development

Test Driven Development

- Write a Test
- Make it Compile
- Make it Pass
- Refactor
- (Repeat)
- Legacy Code Dilemma

Tests and Code Quality

- Less coupling
- Verifiable
- Less Fragile
- But…
  - TDD Not a silver bullet!
  - Beware of "micro-optimizations"

Where to Start?

- Test when making changes
- Evaluate existing tests
- Run end to end tests
- Unit Test
- Repeat

Build Time

- Slow Builds v Test Coverage
- Slow Builds v Modules
- Integration v Unit Tests

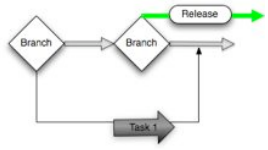
Progress

- Verify that system works as expected
- Infrastructure for new tests

Release Line

- Support existing customers
  - Initial agile experiments are risky.

Isolate Less Agile Code



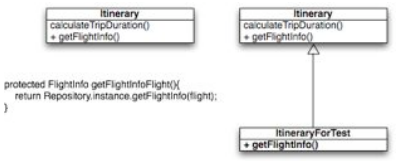
Refactoring Techniques/Patterns

- (From Michael Feathers *Working Effectively with Legacy Code*)
- Replace Global Access with Getter
- Problem: Want to test a method that references a Singleton.

Replace Global with Getter

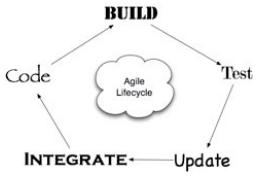
```
Class Itinerary {
  public Duration calculateTripDuration(Flight flight, Date day) {
    FlightInfo info=   Repository.getInstance().getFlightInfo(flight);
    if(isFriday(today)){
       return info.getDuration()  + asHours(3);
    } else {
       return info.getDuration() + asHours(1);
    }
  }
}
```


Replace Global with Getter



Summary

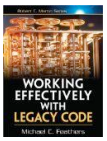
- Enable Feedback
  - Build
  - Code
    - Refactor
  - Test
  - Update
  - Integrate

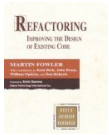


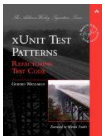
Agile Code

- Unit Testable
- Low-coupling
- Easy to Refactor
- Quick to build & test
- “Safe” to Change
- Feedback Mechanisms

Resources







Questions?