

Agile Firestarter

Continuous Integration

Concepts, Values, and Tools to kick start
your own CI process

Erik Wynne Stepp
Lead Consultant, ThoughtWorks

The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the **right**, we value the items on the **left** more.

The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it.

Working software over comprehensive documentation

Responding to change over following a plan

The Agile Manifesto

Individuals and interactions over processes and tools

there is value in the items on the **right**

WHAT IS CONTINUOUS INTEGRATION?

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.

--Martin Fowler

*Automated way to enable us to
rapidly respond to change
while
continuing to have working software.*

- Erik Wynne Stepp

Benefits of Continuous Integration

Shorten
feedback loop

- Broken build
- Failed tests
- Can't deploy

Reduce risks

- Exercise whole build and deployment process

Provide
confidence of
quality

- Unit, integration, functional testing
- Code metrics
- Code conformance

HOW DOES IT WORK?

CI Servers

Atlassian's
Bamboo

ThoughtWorks'
Cruise

CruiseControl family
(CC, CC.NET, CC.rb)

Hudson

JetBrains'
TeamCity

Microsoft's Team
Foundation
Server

Components

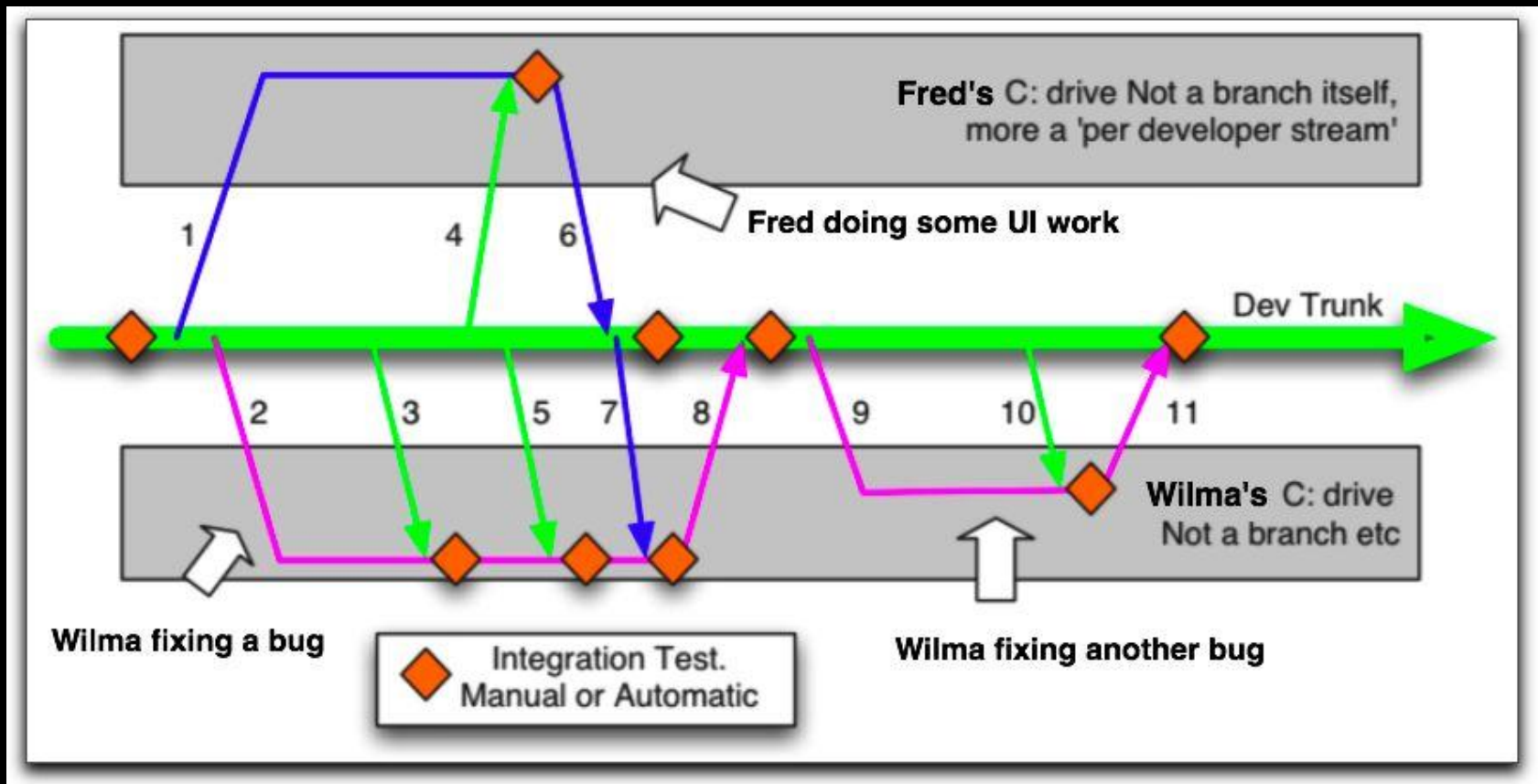
CI Server

- Background process
- Web site
- Notification (email, system tray, RSS, etc.)

Build Scripts

- Compile
- Deploy
- Run tests

Continuous Integration



GETTING STARTED

Get it building locally

- Write an automated build script.
- MSBuild is a good place to start
- Consider other build tools when you grow
 - [MSBuild Extension Pack](#)
 - [NAnt](#) + [NAntContrib](#)
 - [PSake](#)
- Make it simple and fast

Build Scripts – MSBuild

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <OutputPath>.\bin</OutputPath>
  </PropertyGroup>
  <Target Name="FooCompilation">
    <MakeDir Directories= "$(OutputPath)"/>
    <Copy SourceFiles="Image.jpg"
      DestinationFiles="$(OutputPath)\Image.jpg"/>
    <Csc Sources="Foo2.cs" TargetType="module"
      OutputAssembly="$(OutputPath)\Foo2.netmodule" />
    <Csc Sources="Foo1.cs" TargetType="exe"
      AddModules="$(OutputPath)\Foo2.netmodule"
      LinkResources="Image.jpg"
      OutputAssembly="$(OutputPath)\Foo1.exe" />
  </Target>
</Project>
```

Build Scripts – NAnt

```
<?xml version="1.0"?>
<project name="Hello World" default="build" basedir=".">
  <description>The Hello World of build files.</description>
  <property name="debug" value="true" overwrite="false" />
  <target name="clean" description="remove all generated files">
    <delete file="HelloWorld.exe" failonerror="false" />
    <delete file="HelloWorld.pdb" failonerror="false" />
  </target>
  <target name="build" description="compiles the source code">
    <csc target="exe" output="HelloWorld.exe" debug="${debug}">
      <sources>
        <includes name="HelloWorld.cs" />
      </sources>
    </csc>
  </target>
</project>
```

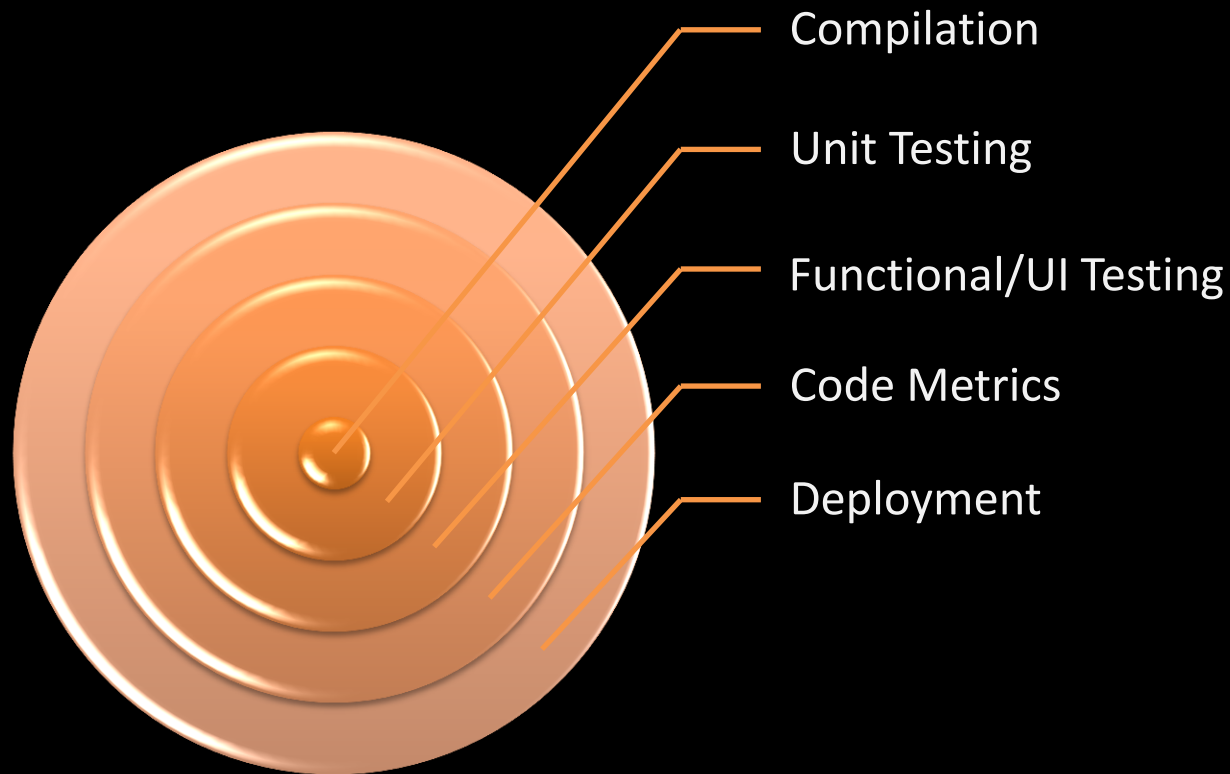
Build Scripts – psake

```
properties {  
  $testMessage = 'Executed Test!'  
  $compileMessage = 'Executed Compile!'  
  $cleanMessage = 'Executed Clean!'  
}  
task default -depends Test  
task Test -depends Compile, Clean {  
  Write-Host $testMessage  
}  
task Compile -depends Clean {  
  Write-Host $compileMessage  
}  
task Clean {  
  Write-Host $cleanMessage  
}
```

Manage Dependencies

- Consider adding 3rd party tools to SCC
- Add 3rd party libraries to SCC
 - Commit binaries so that you don't need to build these every time.
 - Prefer a public release. Avoid custom changes.
 - Document clearly which version you use.

Quality in Depth



Oops, I did it again

- Build status must be visible
- Use the systray app for your CI server
- Consider adding the Build Dashboard to an “Information Radiator”
- Some have used ambient lights, lava lamps, or even songs to let the whole team know when a build breaks.

Notifications

Emails

System Tray utilities (CCTray, CCMenu, etc.)

RSS feed

Text Message

Notifications – Lava Lamps



<http://agileworks.blogspot.com/2009/02/lava-lamp-with-cruisecontrol.html>

Notifications – iPhone app



Introducing CI

- Compile-Only builds can work as “training wheels” for a team that is new to CI.
- Add metrics early so that you can measure your progress as you add automated testing.
- Add automated tests ASAP.
- Make a clear distinction between automated Unit Tests and Functional Tests.
- Automate deployment.

Philadelphia Winter 2011

Agile
Firestarter 

EDUCATE DEVELOPERS

Check-In Process

1. Wait for the CI server to be **green**.
2. Sync your local working folder.
3. Verify the code works using the same build script as the build server.
4. If the CI server is still **green**, commit your changes.
5. Verify that the CI server is **green** before you leave.

Best Practices

- Only do atomic commits. Check-in all or nothing.
- Don't wait until you want to leave for the day. Give yourself time to resolve failures.
- Check-in often.
- Learn the build process. Don't treat it as a "black box".
- Ignore generated binaries in source control.

Broken Builds

- If someone breaks the build, own it and let everyone else know that you're fixing it.
- Don't take a broken build too lightly.
- Don't take a broken build too seriously, either.
- Resolve the broken build quickly. If it will take a while to figure it out, revert the commit that broke it.

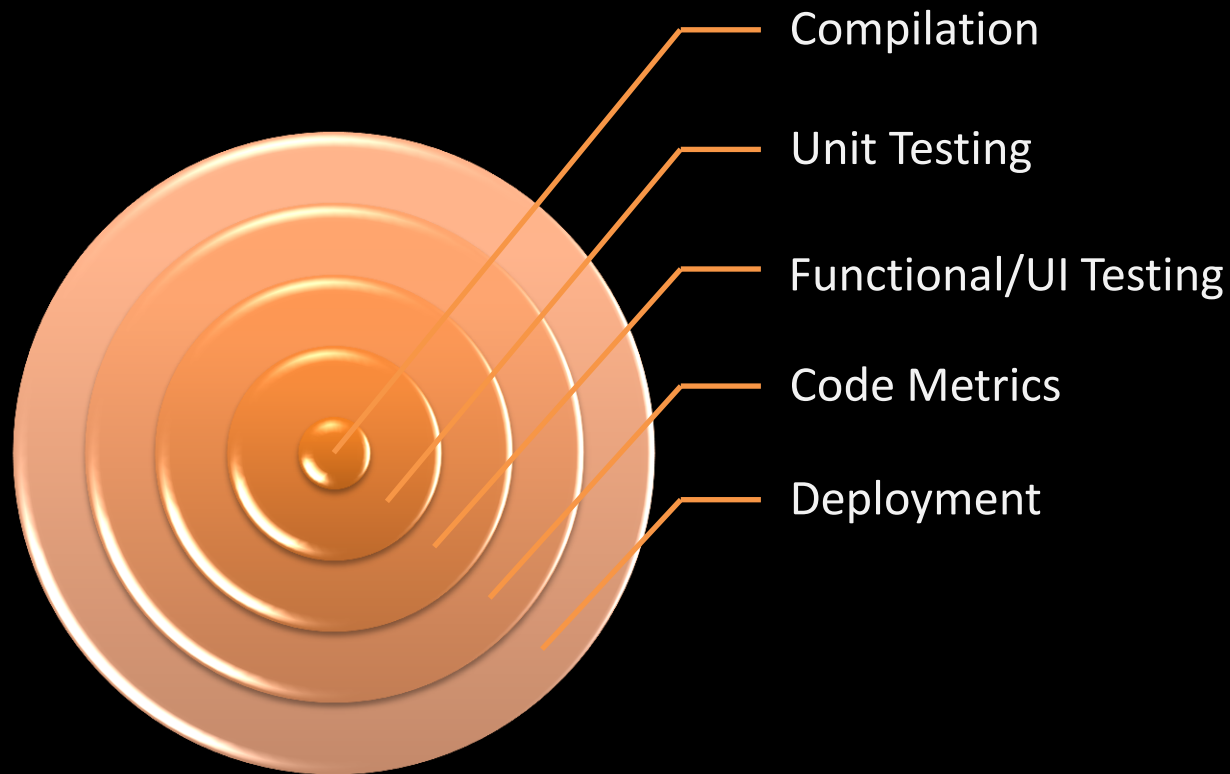
AUTOMATED TESTING

I see CI as primarily giving birth to a release candidate at each commit. The job of the CI system and deployment process is to disprove the production-readiness of a release candidate.

This model relies on the need to have some mainline that represents the current shared, most up to date picture of complete.

--Dave Farley

Quality in Depth

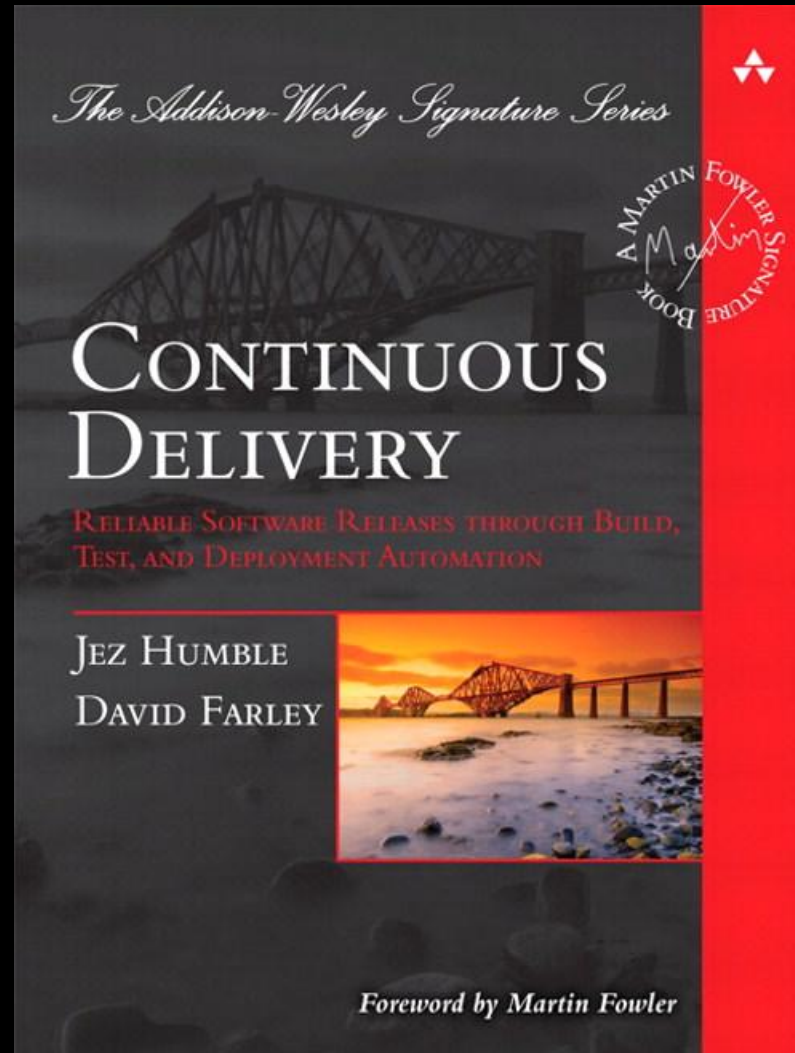


Automated Testing Tools

- NUnit
- MbUnit
- xUnit
- MSTest
- PartCover
- NCover

Of course you can't count on tests to find everything. As it's often been said: tests don't prove the absence of bugs. However perfection isn't the only point at which you get payback for a self-testing build. Imperfect tests, run frequently, are much better than perfect tests that are never written at all.

- Martin Fowler



CONTACT INFORMATION

Erik Wynne Stepp

erik.stepp@gmail.com

@erikwynne

<http://blog.erikstepp.com>

<http://meetup.com/silverlight>