



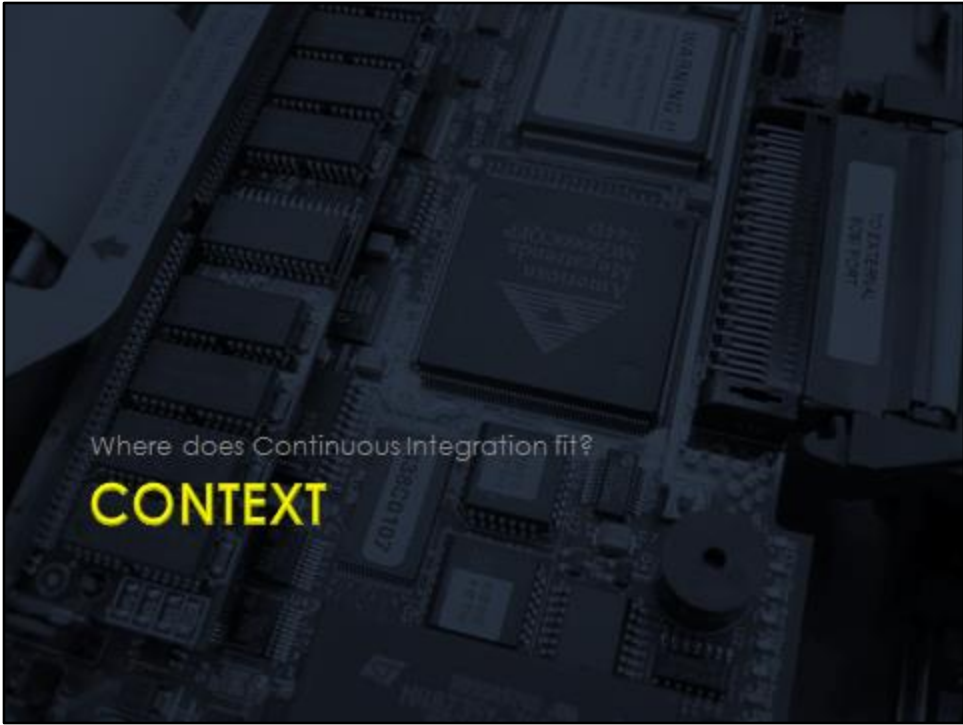
continuous integration



# Continuous Integration

Bevan Arps  
bevan@nichesoftware.co.nz  
@unrepentantgeek





Where does Continuous Integration fit?

# CONTEXT

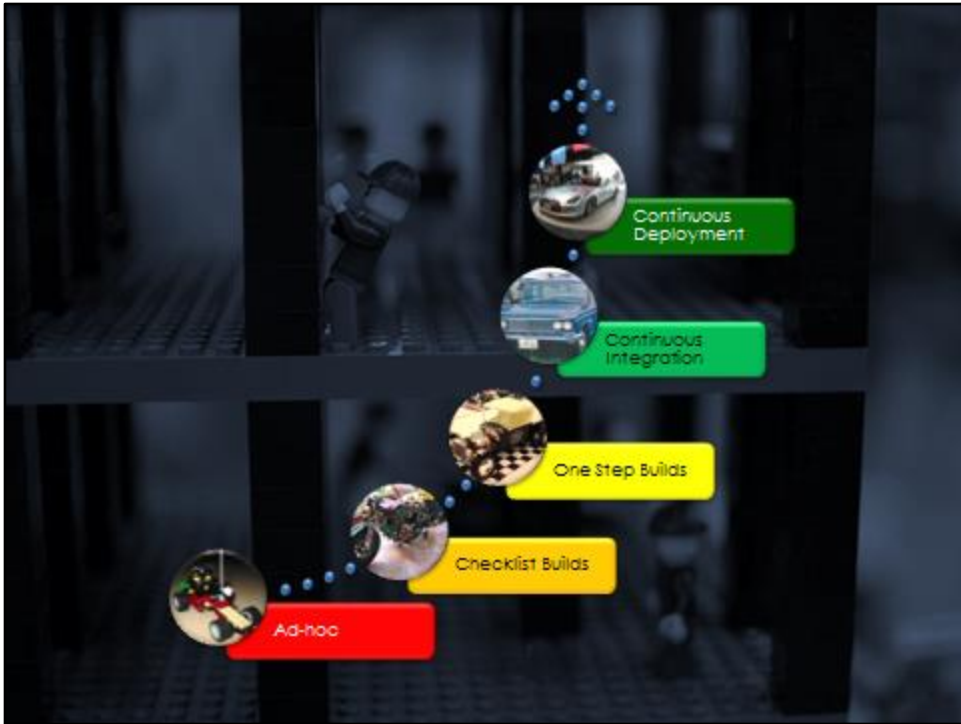


Image Credit: <http://www.flickr.com/photos/warquel/3397541204/>



Fully manual – hit build in your IDE of choice and then manually copy files from the developers machine

Easy and simple  
But never done the same way twice

Image Credit: <http://www.flickr.com/photos/gokrzy/478929790/>

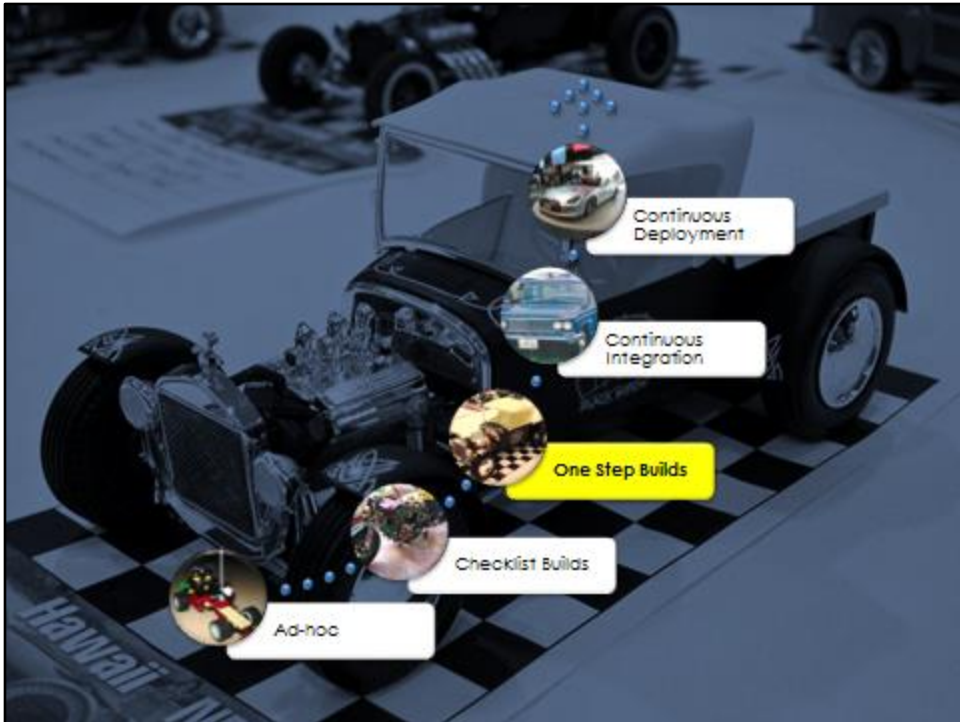


Introduce a checklist to try and ensure the build is done the same way every time  
BUT: Humans are not good at doing this kind of thing

AND: Differences between developer machines can give results that are different  
from one dev to the next

Image Credit: <http://www.flickr.com/photos/meccanohig/3277421633/>

Works on My Machine: Scott Hanselman,  
<http://www.hanselman.com/blog/IntroducingRockScroll.aspx>



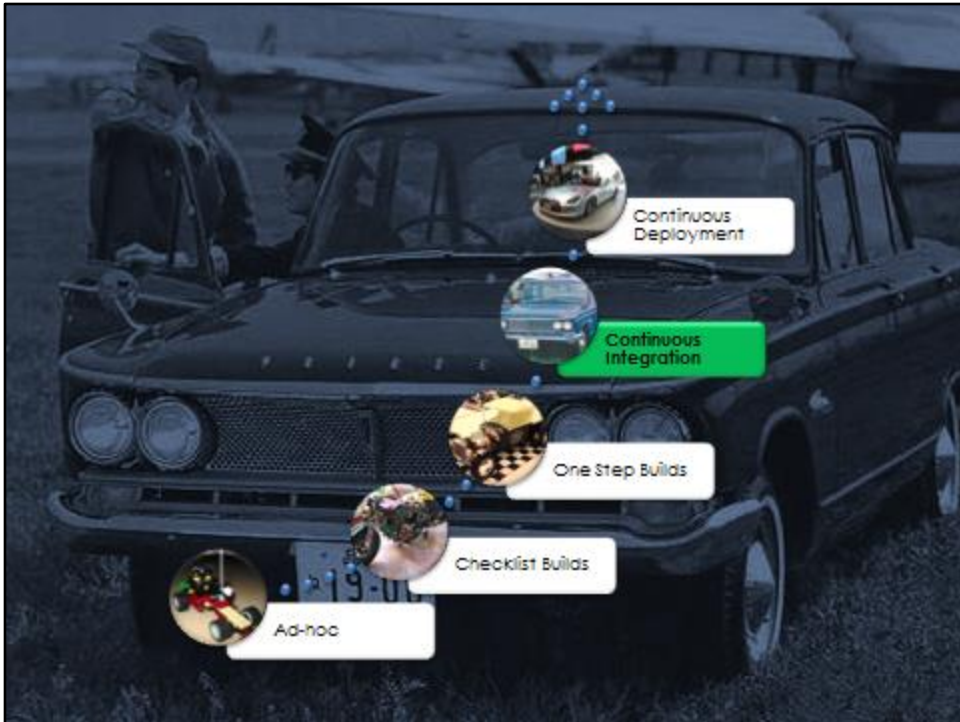
The entire process is automated, with no human intervention

Reliability and consistency are improved

BUT: Still subject to the differences between developer machines

Image Credit: <http://www.flickr.com/photos/puukibeach/5528930610/>





More than just one step builds

Scheduling, triggering, a dedicated build machine

All builds done by the same process on the same machine  
Very reliable, predictable, quick turnaround on changes without compromising,  
without skipping steps

Image Credit: <http://www.flickr.com/photos/hugo90/4195673863/in/photostream/>



The ultimate goal – from commit to go live in minutes

Image Credit:

<http://www.flickr.com/photos/terryansimon/2324103766/in/photostream/>



# Welcome to code.flickr

Your one-stop shop for information, gossip and discussion with the Flickr developer community

**A QUICK TOUR**  
You can read the [DevBlog](#), [dissect the Uploadr](#), or [hone your API wizardry](#) in the forums.

**DEVBLOG: RECENT POSTS**  
Flickr development team talks nerdy.

**Don't be so PuSHy**  
Posted by nls on June 30 2011  
You know three things that would be cool? the ability to subscribe to the output of a Flickr API call in a feed aggregator the ability to get the results of a Flickr API call as... Oh wait. That was a while ago. Wouldn't it be good if you didn't have to poll our API over and over just to get...  
[Read on](#)

**Flickr now Supports OAuth 1.0a** Posted by paulmison on June 20 2011

**Refreshing The API Explorer** Posted by paulmison on June 1 2011

**Inspiration Tuesday** Posted by standardpixel on March 1 2011

**A YUI3 Module for the Flickr API** Posted by standardpixel on February 22 2011

**The Joy of Popup Windows** Posted by stephen on February 1 2011

**THE API: RECENT POSTS**  
Discuss developing against the Flickr web services API.

**Reply to A simple "get images" request, and a numbers question!** Posted 56 minutes ago

**Reply to how to get a photo's list in photo stream** Posted 37 minutes ago

**how to get a photo's list in photo stream** Posted 2 hours ago

**Reply to A simple "get images" request, and a numbers question!** Posted 2 hours ago

**Reply to Can't add videos to a group whilst processing...** Posted 9 hours ago

[more...](#)

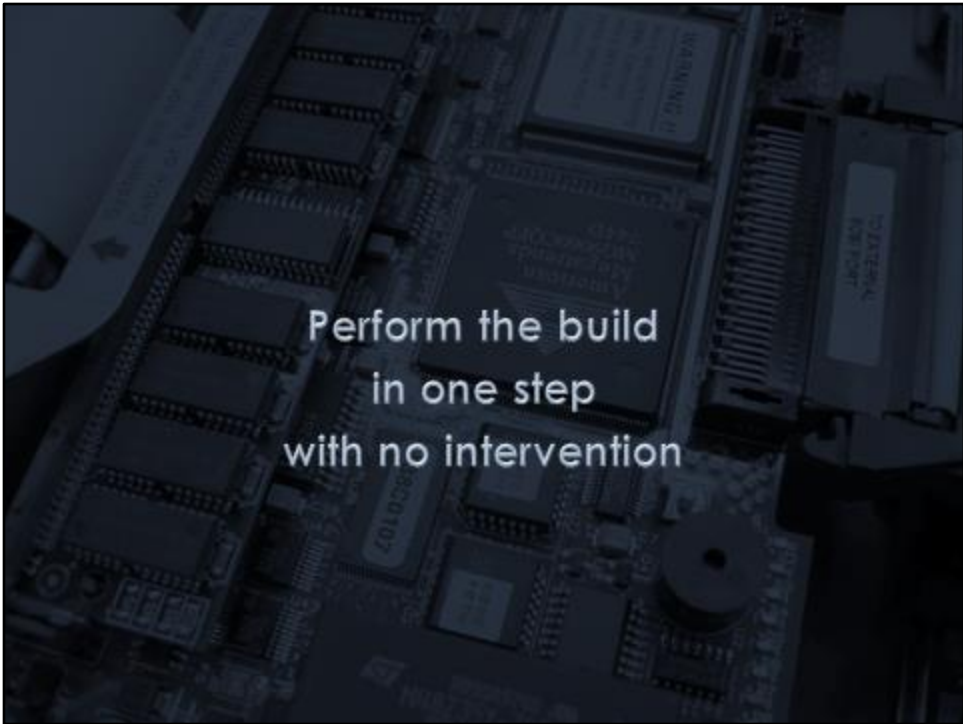
**HACKING UPLOADR: RECENT POSTS**  
A place to moan about all things XULRunner.





The first step is to be able to run a build in one step

And, no, Control-Shift-B in Visual Studio doesn't count



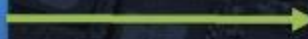
Perform the build  
in one step  
with no intervention

## Some Definitions

Target  
Dependency

Clean

Compile



# What to include?

Things to build

Things to do

**Executables**

**Unit Tests**

**Installers**

**Acceptance Tests**

**API Documentation**

**Static Analysis**



**Make:** The original build automation tool  
Great for C programs, but not really up to date with modern environments

**NAnt/Ant:** Mature tool with Xml based build files  
<http://nant.sourceforge.net/>

**MSBuild:** Microsoft's build tool.  
Visual Studio project files are MSBuild scripts.  
Some say inspired by NAnt  
<http://msdn.microsoft.com/en-us/library/0k6kkbsd.aspx>

**Rake:** Ruby Make  
<http://rake.rubyforge.org/>

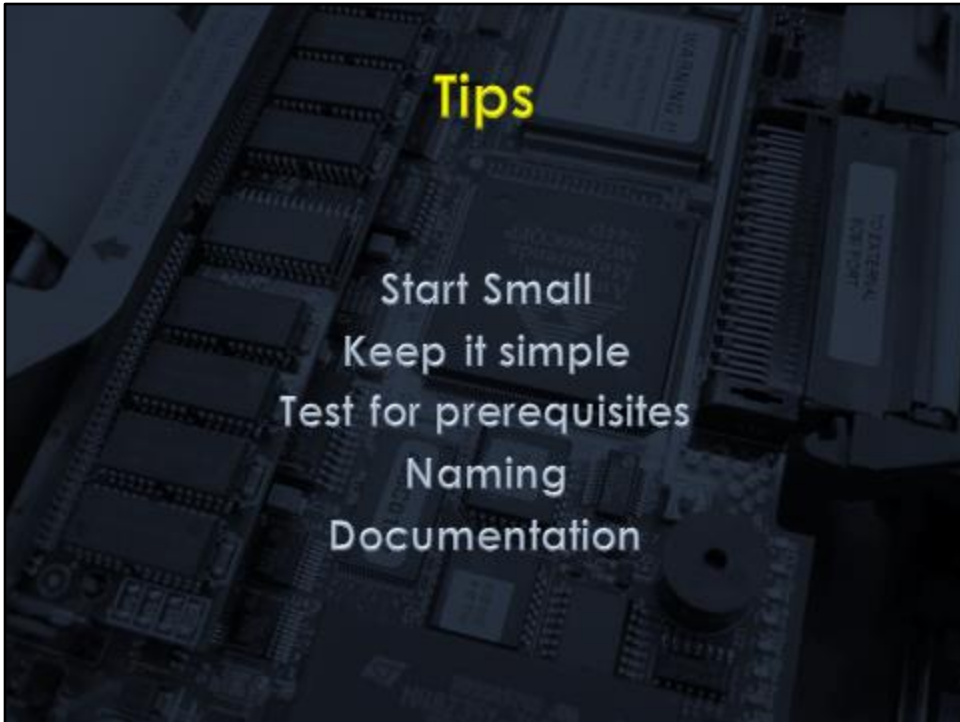
**Psake:** Powershell Make  
<http://codebetter.com/jameskovacs/2010/04/12/psake-v4-00/>

### Which one to choose?

- Technology** – stick with a tech platform familiar to your team  
Don't introduce Rake if you have no other Ruby code
- Support** – how much support is available for the product you choose?
- Documentation** – how much is there?



Need to find a level that is comfortable for your team, your organisation



### **Start Small**

Don't try to do everything at once  
Start small, build it incrementally; And always keep it tidy  
Your build scripts are code too  
My approach: Improve things one step with each build,  
each time I need to push a release through the system make one thing easier to do  
next time.

### **Keep it Simple**

Keep your targets simple and easy to understand  
Yes, you can get all meta and write reusable tasks controlled by metadata and  
variables you define in the script  
I've done it!  
Turned out to be needless complexity.  
Rely on the power of the tool you're using and keep things straightforward  
Refactor your targets if they start getting complex

### **Test for Prerequisites**

Some of your targets will have prerequisites  
Break these out into separate targets that test for them  
Fail the build if they're not found

## **Naming**

Name things well

Use standard names for standard things

clean • compile • build • test

## **Documentation**

Your build scripts are only going to have occasional maintenance

No one – not even you – will remember all the details

So put comments/documentation into the build script so that anyone can maintain it

# Naming

require.xyz  
compile.xyz  
test.xyz

```
<target name="compile assemblies" description="Build output from source">
  <!-- Define the directory where we expect to find msbuild -->
  <property name="msbuild dir" value="C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319" />
  <property name="msbuild exe" value="{ path combine( msbuild dir, 'msbuild.exe' ) }" />
  <fail message="Didn't find ${msbuild exe}" unless="{ file exists( msbuild exe ) }" />

  <!-- Update
  <version s
  <!-- Gener
  <asmInfo 1
  <imports
  <import
  <import
  </import
  <attribu
  <attrib
  </attrib
  </asmInfo>
  <!-- Comp
  <exec prog
  <arg fil
  <arg val
  <arg val
  </exec>
</target>
```

Keep your targets simple and easy to understand

Yes, you can get all meta and write reusable tasks controlled by metadata and variables you define in the script

I've done it!

Turned out to be needless complexity.

Rely on the power of the tool you're using and keep things straightforward

```
<target name="compile versionInfo"
  description="Generate a VersionInfo file for our build">

  <!-- Update our version number -->
  <version startdate="1 May 2010"
    path="${src.dir}\version.txt" prefix="build" />

  <!-- Generate our shared AssemblyInfo file -->
  <asminfo language="CSharp"
    output="src\VersionInfo.cs">
    <imports>
      <import namespace="System" />
      <import namespace="System.Reflection" />
    </imports>
    <attributes>
      <attribute type="AssemblyVersionAttribute"
        value="${build.version}" />
      <attribute type="AssemblyFileVersionAttribute"
        value="${build.version}" />
    </attributes>
  </asminfo>
</target>
```

Keep your targets simple and easy to understand

Yes, you can get all meta and write reusable tasks controlled by metadata and variables you define in the script

I've done it!

Turned out to be needless complexity.

Rely on the power of the tool you're using and keep things straightforward

```
<target name="compile.assemblies"
  description="Build output from source"
  depends="require.msbuild, compile.versionInfo">

  <!-- Compile everything -->
  <exec program="$(msbuild.exe)">
    <arg file="NicheDashboard.sln"/>
    <arg value="/t rebuild"/>
    <arg value="/verbosity quiet"/>
  </exec>

</target>
```

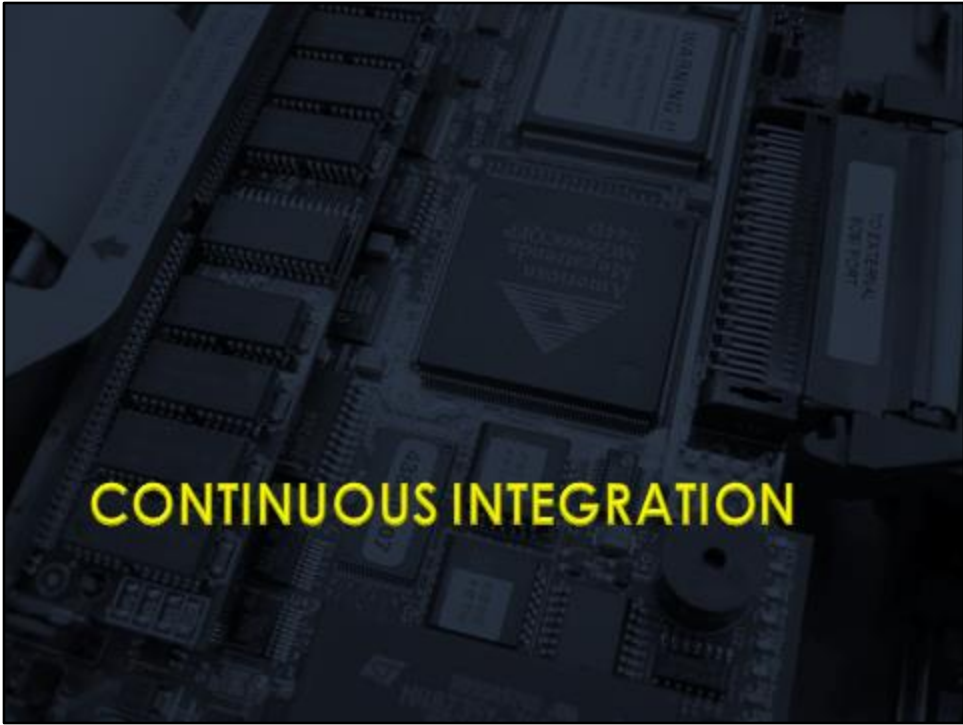
Keep your targets simple and easy to understand

Yes, you can get all meta and write reusable tasks controlled by metadata and variables you define in the script

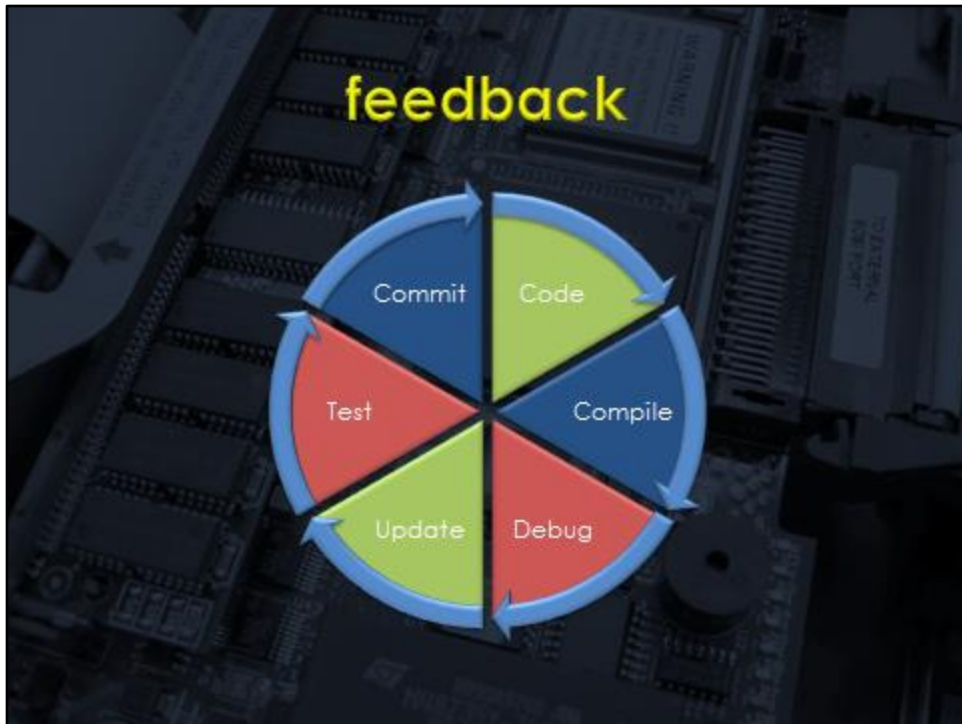
I've done it!

Turned out to be needless complexity.

Rely on the power of the tool you're using and keep things straightforward

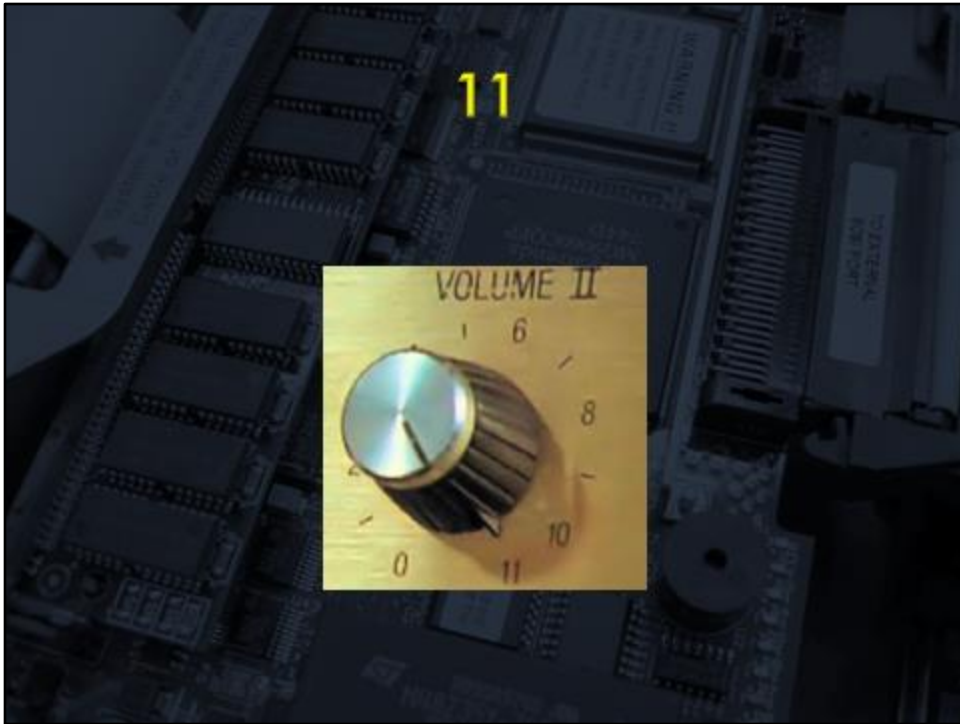




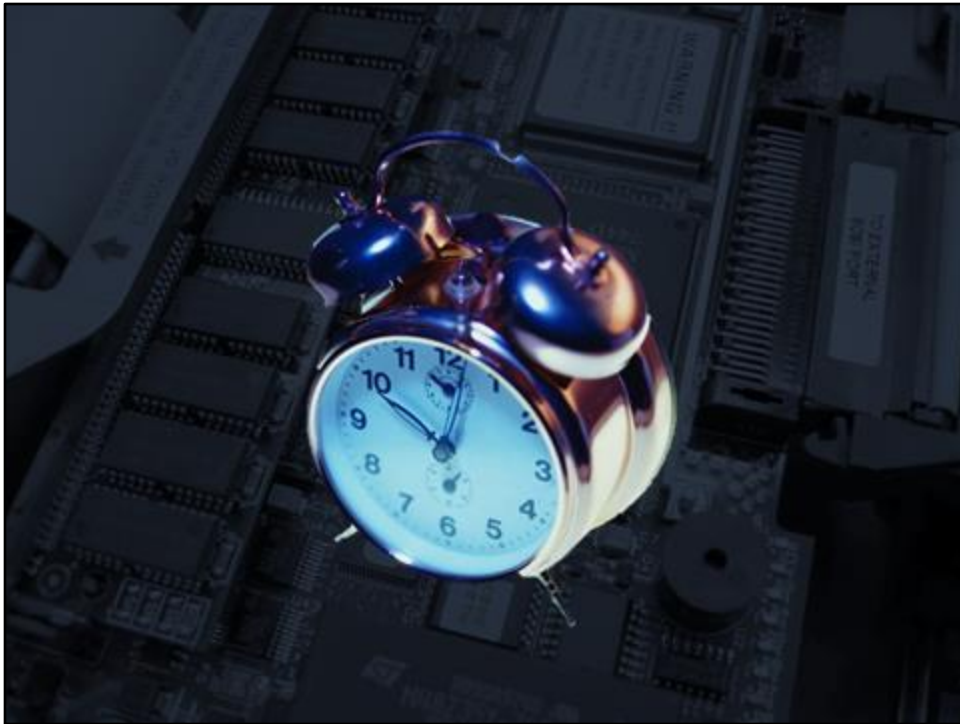


With one step builds you can automate some of this  
But you're still doing much of it manually  
And what happens when someone takes a shortcut ...

... by committing a trivial fix without compiling  
... or by not testing changes after updating



Continuous Integration is about taking that feedback loop and turning the volume up to 11



It's about being able to schedule builds to happen at particular times  
Or to be triggered by regular events



And about having the alarm raised when something goes wrong



CruiseControl.Net – ThoughtWorks

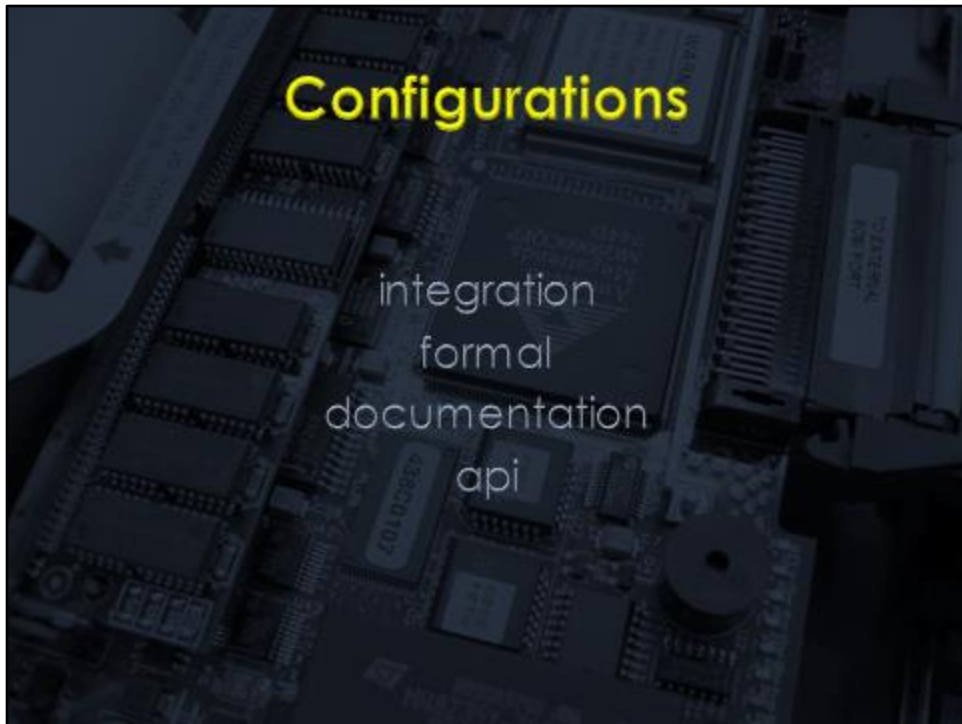
TeamCity – JetBrains – Not open source, but free

http://teamcity.codebetter.com

Projects - Agents (5) Build Queue (3) Welcome, Guest user

Collapse All | Expand All

Amazon API client	typica & maragotype	3 successful
Apache Ant		3 failing 3 successful
Apache Ivy		3 failing 1 successful
Artifactory Plugin for TeamCity		2 successful
Bundler	Bundler gem ( <a href="http://github.com/carlhuda/bundler.git">http://github.com/carlhuda/bundler.git</a> )	3 failing
Cucumber	Cucumber BDD Framework ( <a href="http://cukes.info">http://cukes.info</a> )	3 failing
Genome Query	<a href="http://confluence.jetbrains.net/display/GQRY/Genome+Query">http://confluence.jetbrains.net/display/GQRY/Genome+Query</a>	3 failing 2 successful
Google Web Toolkit	Google Web Toolkit - Trunk	2 failing 1 successful
Gradle		1 failing 2 successful
Groovy		3 successful
IdeaVIM	Vim emulation plugin for the IntelliJ platform products	1 successful



## Suggested Build Configurations

### **Integration – or Continuous Integration**

Triggered on check in  
Compilation + Unit tests  
FAST feedback

### **Formal**

Everything required for release of the software  
Triggered daily – at lunchtime  
Compilation + all testing + archiving artefacts

### **Documentation**

Generate documentation e.g. api – typically developer focussed

### **Analysis**


Additional stuff for consumption by the team  
Triggered when a formal build succeeds  
Static code analysis – Test coverage



Once you have your build in place,  
You can start advertising the state of the build

Automated emails, especially to the person who broke the build





thanks

twitter [unrepentantgeek](#)  
email [bevan@nichesoftware.co.nz](mailto:bevan@nichesoftware.co.nz)  
blog [www.nichesoftware.co.nz](http://www.nichesoftware.co.nz)