domineur inc.

(design, build, deploy)+

Toronto Progress User's Group

Continuous integration with Progress

24 October 2017

CoLABorate

About dave

I’m a Toronto based software developer with a small company that builds bespoke applications and software tools. I also provide consulting around software development practices.

Our primary language these days is Java and we have built some tools that IFDS is using in their CI pipeline but as with anyone who’s been programming for a while I’ve used a lot of different languages and databases starting with C/C++ and Clipper ‘till now with many things in between.

I’ve worked on telecom applications, APIs for building SGML and XML applications, integration software using RDF, OWL and SPARQL, software for the healthcare industry as well as less interesting things using Java/Spring/Hibernate/etc., In short we’re Fully Catchword Compliant (FCC). We’ve been using CI in our work since about 2001.

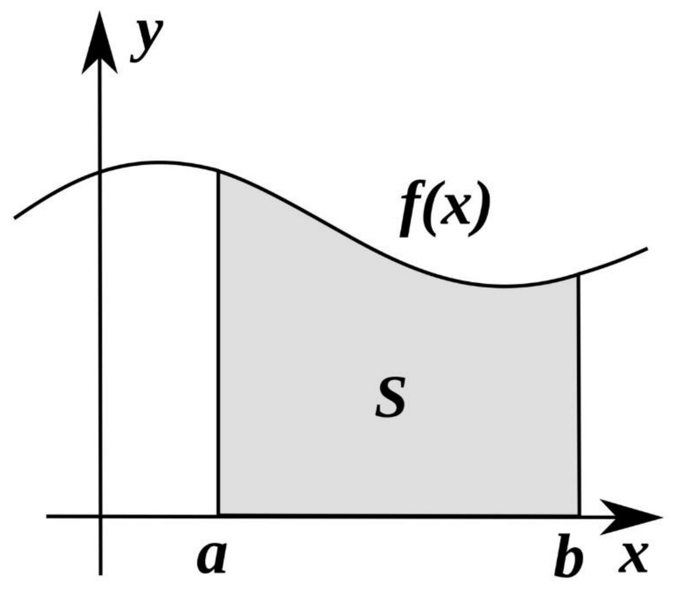
In my spare time I play the ‘cello. You can reach me at dave@domineur.com

The Issue of Integration

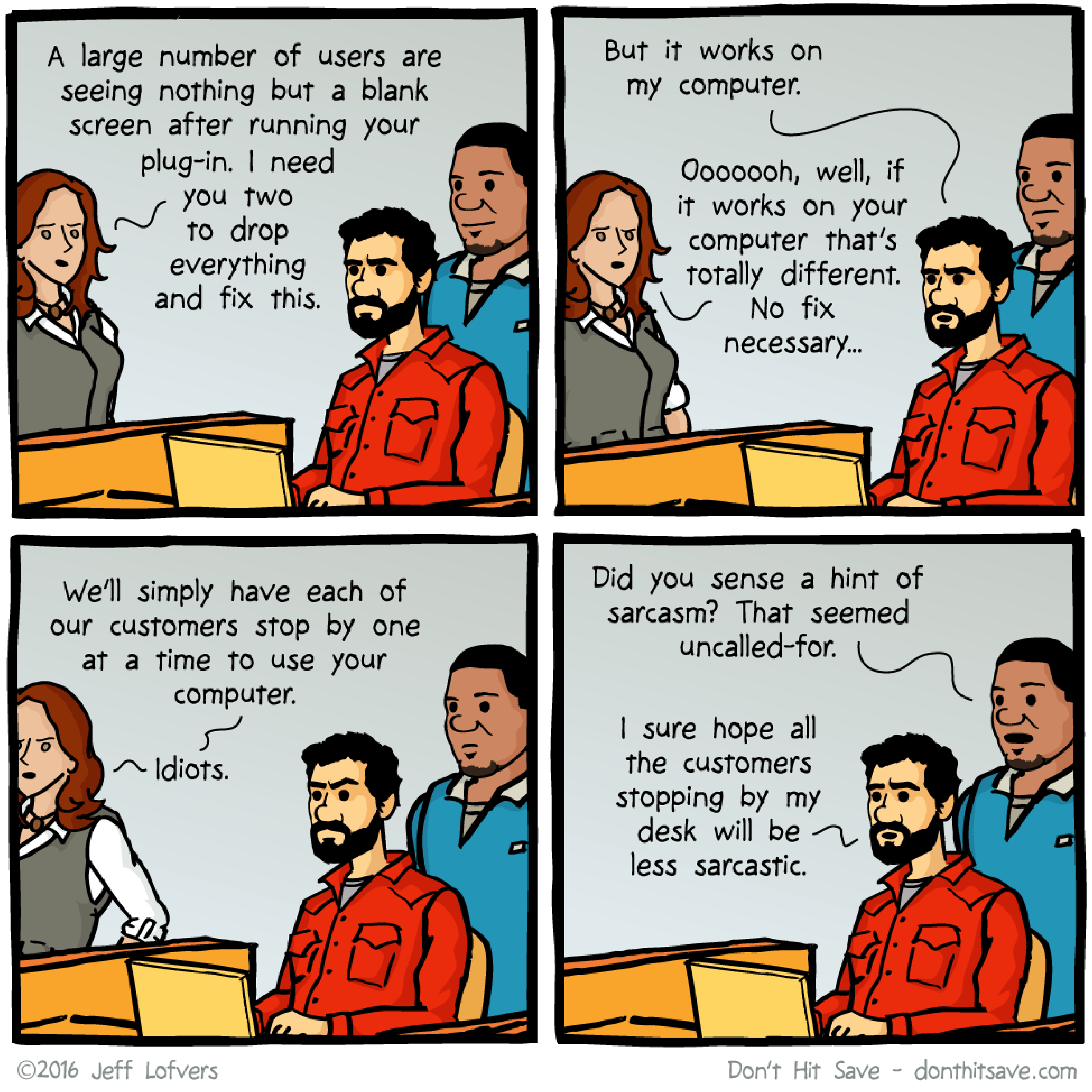
Your code or application is not an island!

When our code is “ready” it still has to be integrated – promoted to a special branch, deployed to environments other than our own machine, and just generally go out and play with the rest of the code nicely.

Lots of things can and do go wrong – can any of you give examples from your own experience?



It Works on my machine

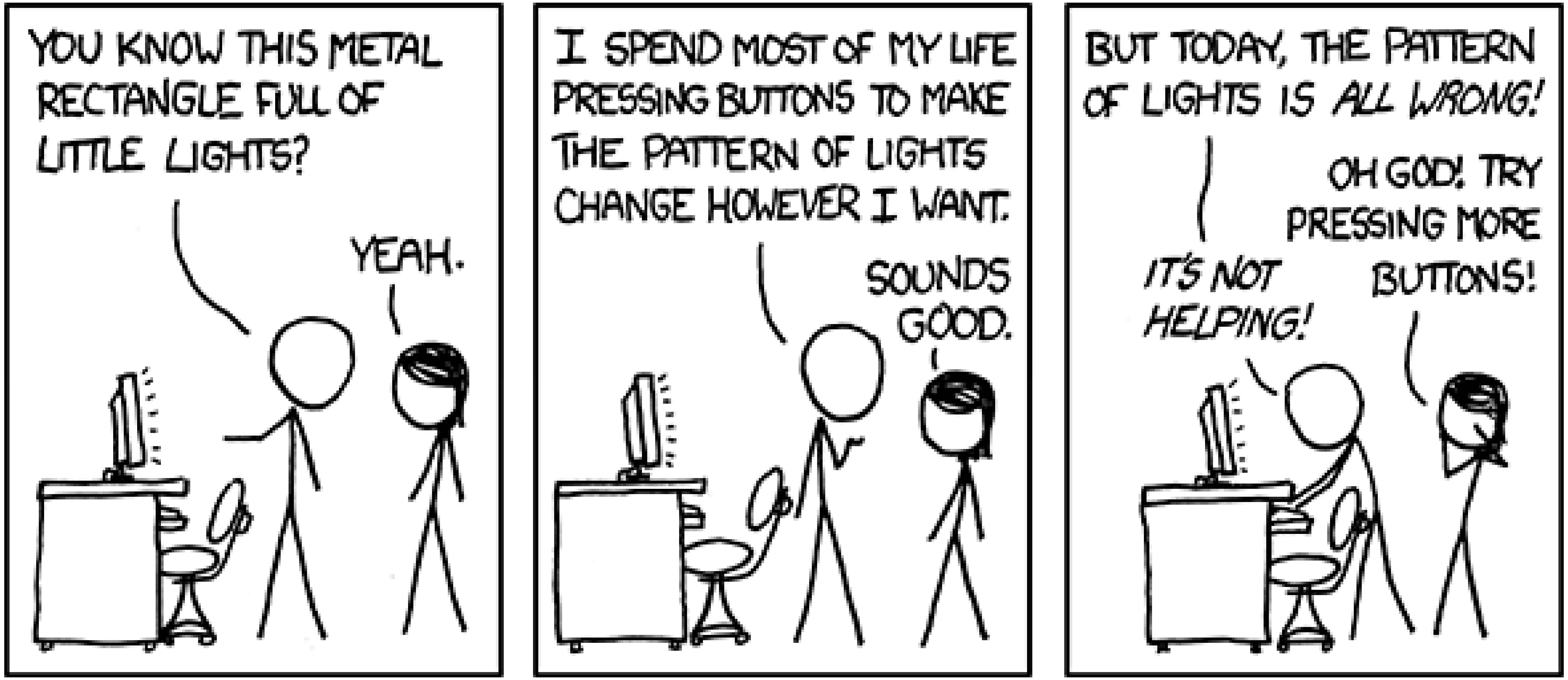

A large number of users are seeing nothing but a blank screen after running your plug-in. I need you two to drop everything and fix this.
But it works on my computer.
Ooooooh, well, if it works on your computer that's totally different. No fix necessary...
We'll simply have each of our customers stop by one at a time to use your computer.
Idiots.
Did you sense a hint of sarcasm? That seemed uncalled-for.
I sure hope all the customers stopping by my desk will be less sarcastic.
©2016 Jeff Lofvers
Don't Hit Save - donthitsave.com

http://donthitsave.com/comic/2016/07/15/it-works-on-my-computer

After merging it doesn’t work on my machine…

https://xkcd.com/722/

Sadly sometimes it doesn’t work on our machine because it’s working on someone **else’s** machine…


YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?
YEAH.
I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.
SOUNDS GOOD.
BUT TODAY, THE PATTERN OF LIGHTS IS *ALL WRONG!*
OH GOD! TRY PRESSING MORE BUTTONS!
IT'S NOT HELPING!

Public

How do we fix this?

This may seem like a strange idea, but the concept is essentially just that integration is hard so we must do it **more frequently** and ideally **as soon as possible.**

- It's important that whatever we are doing is automated – if lots of manual process is involved then it's not repeatable
- That doesn't mean that you need to first automate everything – do what you can and keep it simple to start
- To quote from the Martin Fowler blog post that we're about to run through: "Imperfect tests, run frequently, are much better than perfect tests that are never written at all."

Where does the concept of Continuous Integration come from?

It’s popularity from what I can tell comes from the technical practices paired with agile methodology in eXtreme Programming, and that’s where I was first exposed to it.

The first mention, according to Fowler was in Grady Booch’s excellent book Object-Oriented Analysis & Design with Applications circa 1990.

Grady Booch was one of the fellows providing leadership for the development the UML.

I don’t remember this line (which may only have been in the first edition from what I’ve heard) but that book is great and if you are interested in Domain Driven Design today you may still enjoy reading it for the applications part.

Practices of Continuous Integration

These practices are not technology specific

I'm referring to Martin Fowler's blog post:

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

This post is from 2006.

We'll run through this again step by step

You may know Martin Fowler from some of his books:

- 1996. *Analysis Patterns: Reusable Object Models*. Addison-Wesley. ISBN 0-201-89542-0.
- 1997. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley. ISBN 978-0-201-32563-8.
- 1999. *Refactoring: Improving the Design of Existing Code*, With Kent Beck, John Brant, William Opdyke, and Don Roberts (June 1999). Addison-Wesley. ISBN 0-201-48567-2.
- 2000. *Planning Extreme Programming*. With Kent Beck. Addison-Wesley. ISBN 0-201-71091-9.
- 2002. *Patterns of Enterprise Application Architecture*. With David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, and Randy Stafford. Addison-Wesley. ISBN 0-321-12742-0.
- 2010. *Domain-Specific Languages*. With Rebecca Parsons. Addison-Wesley. ISBN 978-0-321-71294-3.
- 2012. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. With Pramod Sadalage. Addison-Wesley. ISBN 978-0-321-82662-6.
- 2013. *Refactoring: Ruby Edition*. With Kent Beck, Shane Harvie, and Jay Fields. Addison-Wesley. ISBN 978-0-321-98413-5.

Maintain a single source repository

“Once you get a source code management system, make sure it is the well known place for everyone to go get source code. Nobody should ever ask "where is the foo-whiffle file?" Everything should be in the repository. Although many teams use repositories a common mistake I see is that they don't put everything in the repository. If people use one they'll put code in there, but everything you need to do a build should be in there including: test scripts, properties files, database schema, install scripts, and third party libraries. I've known projects that check their compilers into the repository (important in the early days of flaky C++ compilers). ”

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

Automate The Build

“Getting the sources turned into a running system can often be a complicated process involving compilation, moving files around, loading schemas into the databases, and so on. However like most tasks in this part of software development it can be automated - and as a result should be automated. Asking people to type in strange commands or clicking through dialog boxes is a waste of time and a breeding ground for mistakes.

Automated environments for builds are a common feature of systems. The Unix world has had make for decades, the Java community developed Ant, the .NET community has had Nant and now has MSBuild. Make sure you can build and launch your system using these scripts using a single command.

A common mistake is not to include everything in the automated build. The build should include getting the database schema out of the repository and firing it up in the execution environment. I'll elaborate my earlier rule of thumb: anyone should be able to bring in a virgin machine, check the sources out of the repository, issue a single command, and have a running system on their machine.”

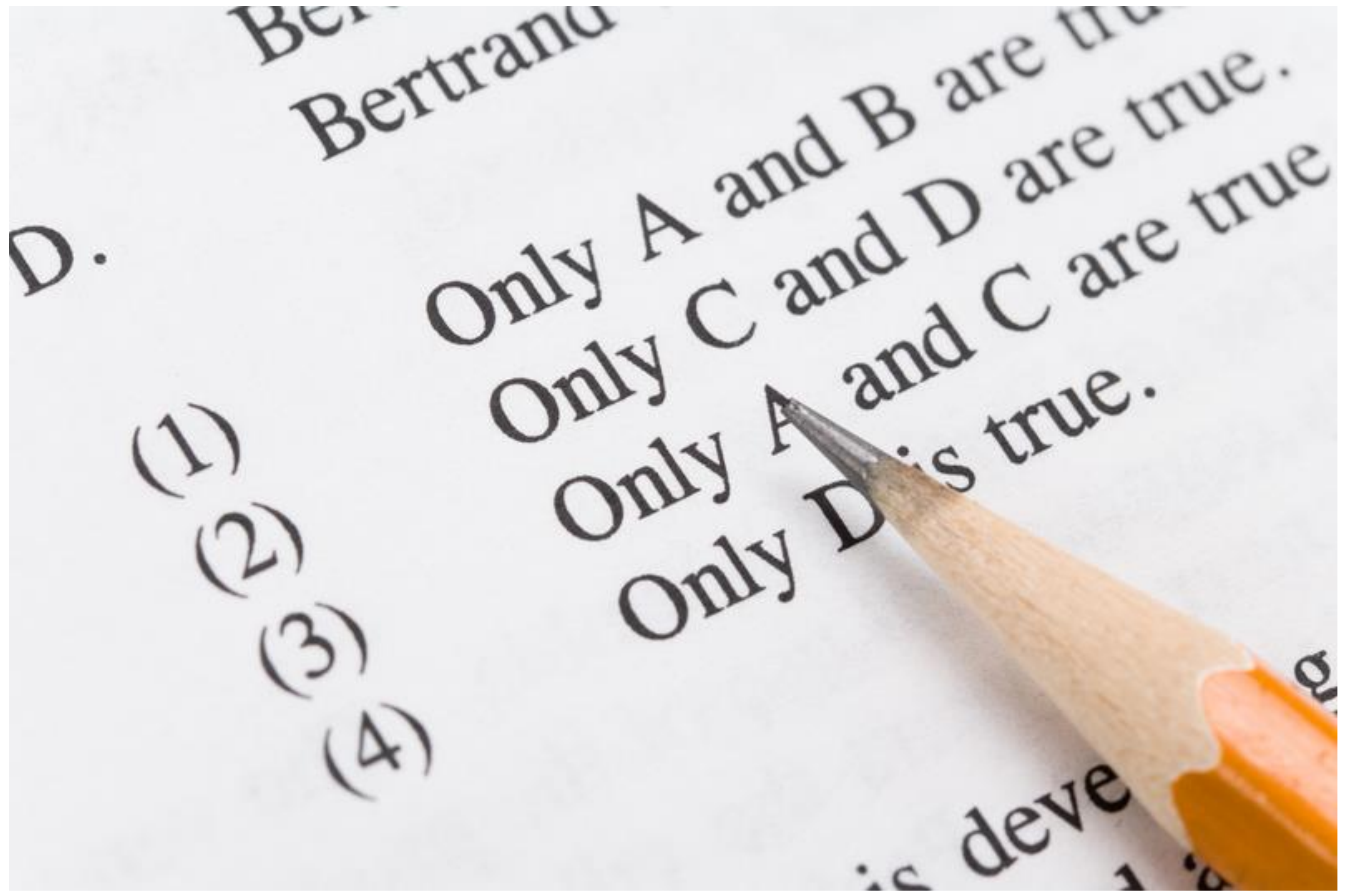
https://martinfowler.com/articles/continuousIntegration.html

Make Your Build Self-Testing

“Traditionally a build means compiling, linking, and all the additional stuff required to get a program to execute. A program may run, but that doesn't mean it does the right thing. Modern statically typed languages can catch many bugs, but far more slip through that net.

A good way to catch bugs more quickly and efficiently is to include automated tests in the build process. Testing isn't perfect, of course, but it can catch a lot of bugs - enough to be useful. In particular the rise of Extreme Programming (XP) and Test Driven Development (TDD) have done a great deal to popularize self-testing code and as a result many people have seen the value of the technique.”

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



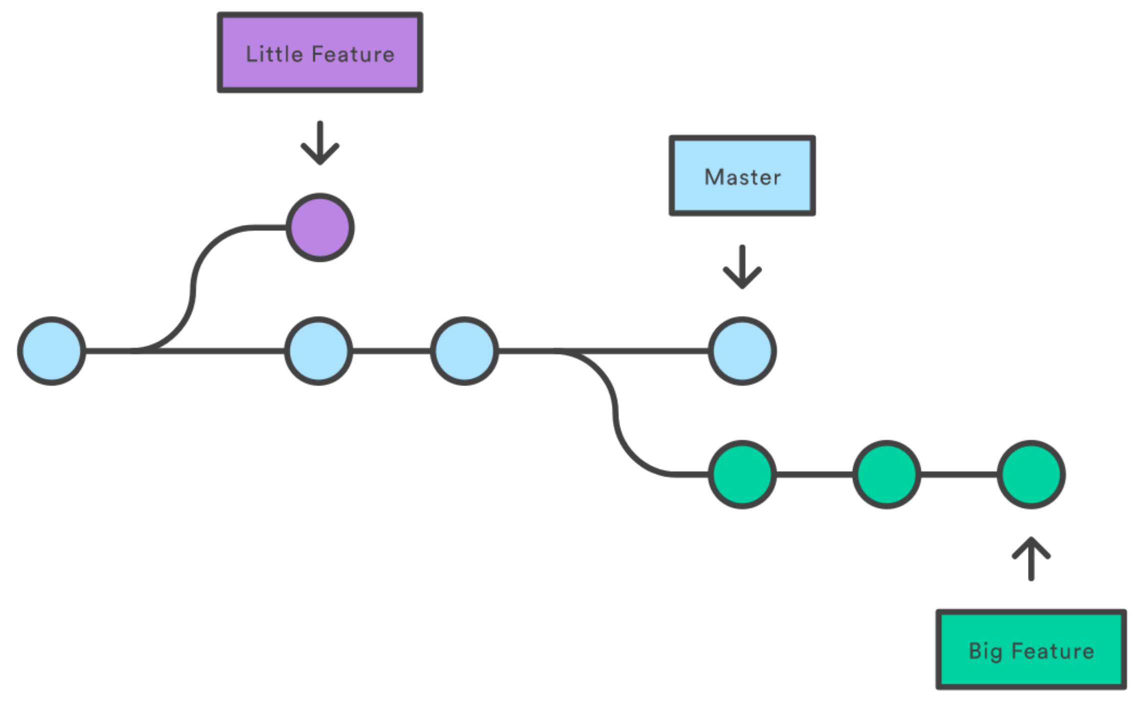
Everyone Commits to the Mainline Every Day

“Integration is primarily about communication. Integration allows developers to tell other developers about the changes they have made. Frequent communication allows people to know quickly as changes develop.

The one prerequisite for a developer committing to the mainline is that they can correctly build their code. This, of course, includes passing the build tests. As with any commit cycle the developer first updates their working copy to match the mainline, resolves any conflicts with the mainline, then builds on their local machine. If the build passes, then they are free to commit to the mainline.

By doing this frequently, developers quickly find out if there's a conflict between two developers. The key to fixing problems quickly is finding them quickly. With developers committing every few hours a conflict can be detected within a few hours of it occurring, at that point not much has happened and it's easy to resolve. Conflicts that stay undetected for weeks can be very hard to resolve.”

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



Every Commit Should Build the Mainline on an Integration Machine

“Using daily commits, a team gets frequent tested builds. This ought to mean that the mainline stays in a healthy state. In practice, however, things still do go wrong. One reason is discipline, people not doing an update and build before they commit. Another is environmental differences between developers' machines.

As a result you should ensure that regular builds happen on an integration machine and only if this integration build succeeds should the commit be considered to be done. Since the developer who commits is responsible for this, that developer needs to monitor the mainline build so they can fix it if it breaks. A corollary of this is that you shouldn't go home until the mainline build has passed with any commits you've added late in the day.”

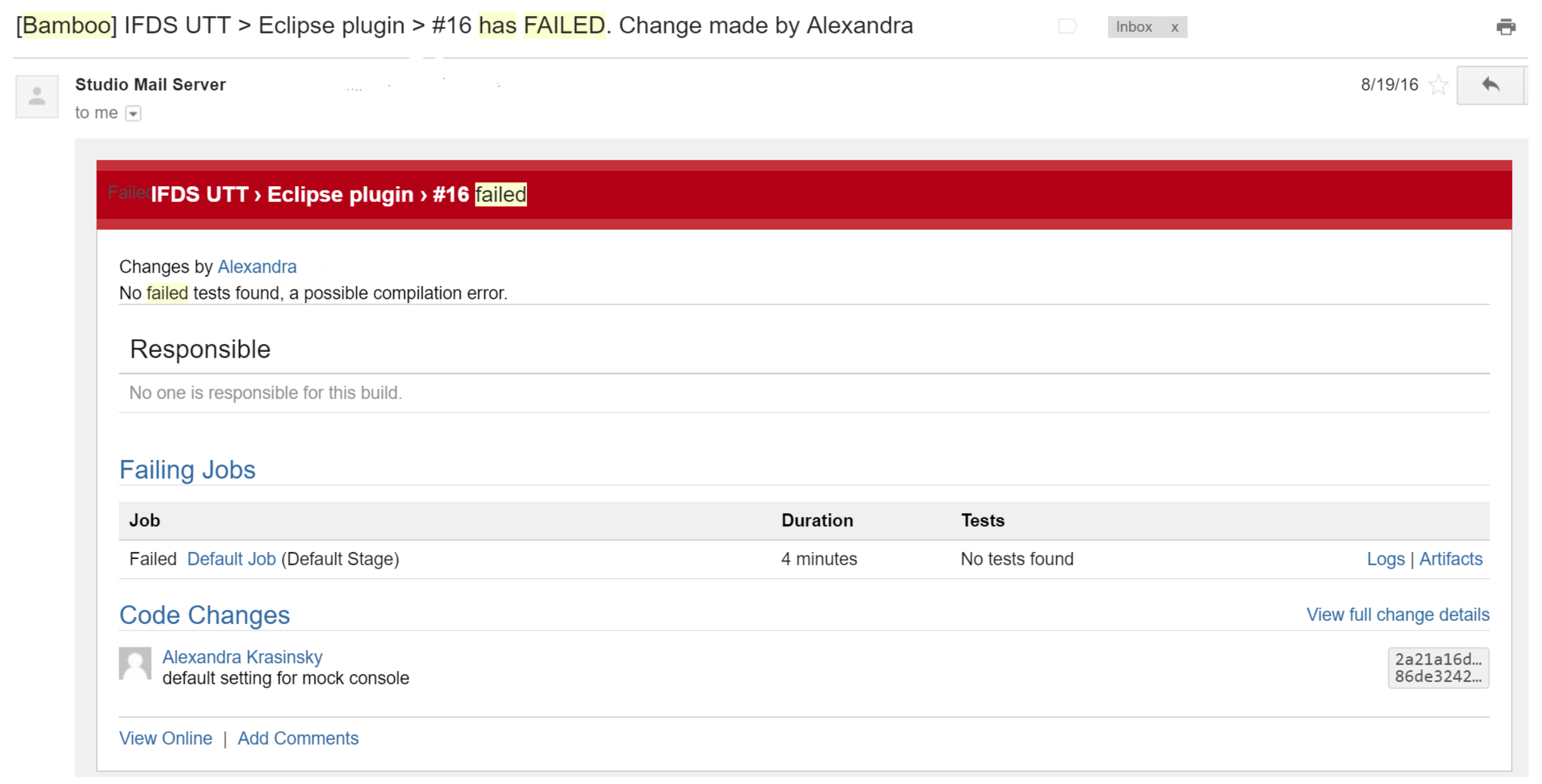
https://martinfowler.com/articles/continuousIntegration.html

Fix Broken Builds Immediately

Using daily commits, a team gets frequent tested builds. This ought to mean that the mainline stays in a healthy state. In practice, however, things still do go wrong. One reason is discipline, people not doing an update and build before they commit. Another is environmental differences between developers' machines.

As a result you should ensure that regular builds happen on an integration machine and only if this integration build succeeds should the commit be considered to be done. Since the developer who commits is responsible for this, that developer needs to monitor the mainline build so they can fix it if it breaks. A corollary of this is that you shouldn't go home until the mainline build has passed with any commits you've added late in the day.

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


[Bamboo] IFDS UTT > Eclipse plugin > #16 has FAILED. Change made by Alexandra

Studio Mail Server
to me

8/19/16

Failed IFDS UTT › Eclipse plugin › #16 failed

Changes by Alexandra
No failed tests found, a possible compilation error.

Responsible

No one is responsible for this build.

Failing Jobs

Job	Duration	Tests	
Failed Default Job (Default Stage)	4 minutes	No tests found	Logs | Artifacts

Code Changes — View full change details

Alexandra Krasinsky
default setting for mock console
2a21a16d...
86de3242...

View Online | Add Comments

Keep the Build Fast

The whole point of Continuous Integration is to provide rapid feedback. Nothing sucks the blood of a CI activity more than a build that takes a long time. Here I must admit a certain crotchety old guy amusement at what's considered to be a long build. Most of my colleagues consider a build that takes an hour to be totally unreasonable. I remember teams dreaming that they could get it so fast - and occasionally we still run into cases where it's very hard to get builds to that speed.

For most projects, however, the XP guideline of a ten minute build is perfectly within reason. Most of our modern projects achieve this. It's worth putting in concentrated effort to make it happen, because every minute you reduce off the build time is a minute saved for each developer every time they commit. Since CI demands frequent commits, this adds up to a lot of time.

…

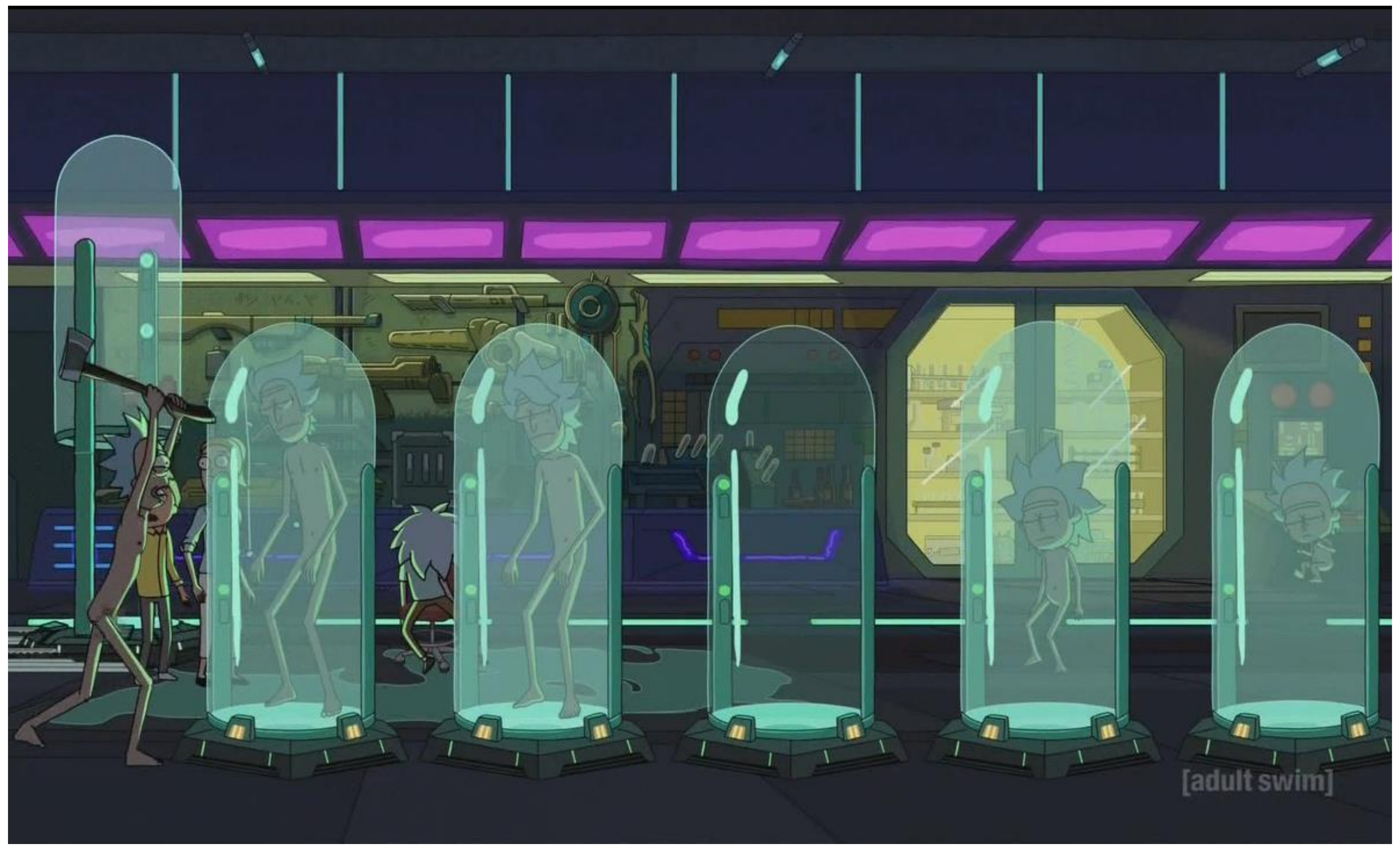
Once the commit build is good then other people can work on the code with confidence. However there are further, slower, tests that you can start to do. Additional machines can run further testing routines on the build that take longer to do.

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

Test in a Clone of the Production Environment

The point of testing is to flush out, under controlled conditions, any problem that the system will have in production. A significant part of this is the environment within which the production system will run. If you test in a different environment, every difference results in a risk that what happens under test won't happen in production.

As a result you want to set up your test environment to be as exact a mimic of your production environment as possible. Use the same database software, with the same versions, use the same version of operating system. Put all the appropriate libraries that are in the production environment into the test environment, even if the system doesn't actually use them.

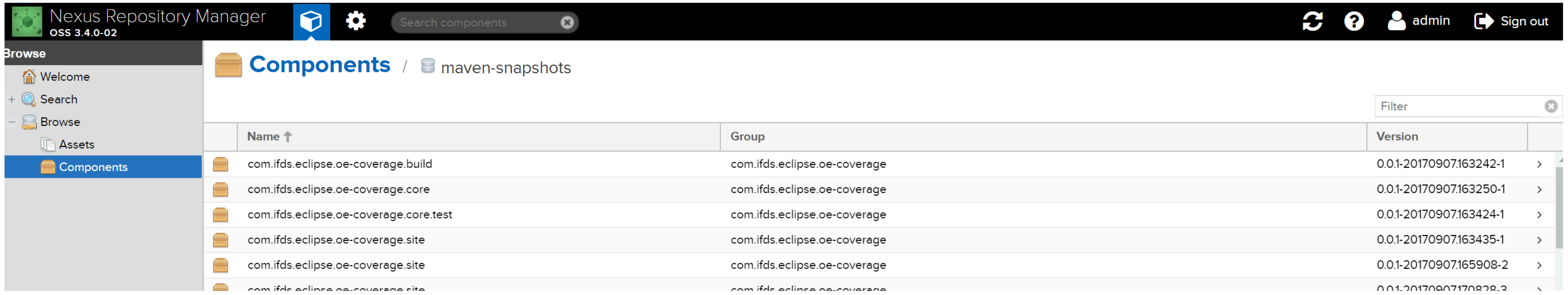


https://martinfowler.com/articles/continuousIntegration.html

Make it Easy for Anyone to Get the Latest Executable

One of the most difficult parts of software development is making sure that you build the right software. We've found that it's very hard to specify what you want in advance and be correct; people find it much easier to see something that's not quite right and say how it needs to be changed. Agile development processes explicitly expect and take advantage of this part of human behavior.

To help make this work, anyone involved with a software project should be able to get the latest executable and be able to run it: for demonstrations, exploratory testing, or just to see what changed this week.



Components / maven-snapshots

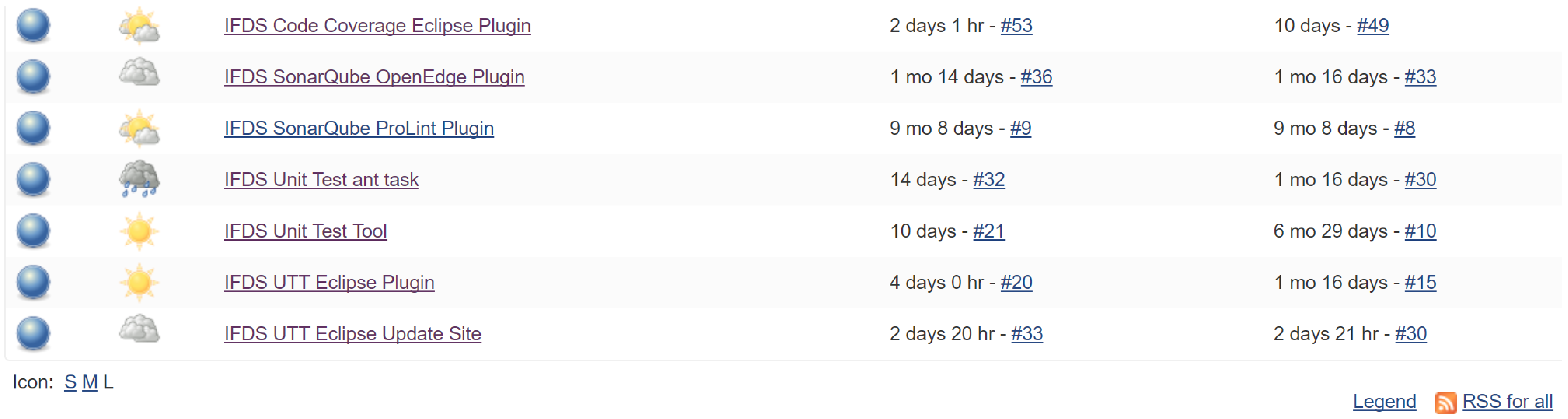
Name	Group	Version
com.ifds.eclipse.oe-coverage.build	com.ifds.eclipse.oe-coverage	0.0.1-20170907.163242-1
com.ifds.eclipse.oe-coverage.core	com.ifds.eclipse.oe-coverage	0.0.1-20170907.163250-1
com.ifds.eclipse.oe-coverage.core.test	com.ifds.eclipse.oe-coverage	0.0.1-20170907.163424-1
com.ifds.eclipse.oe-coverage.site	com.ifds.eclipse.oe-coverage	0.0.1-20170907.163435-1
com.ifds.eclipse.oe-coverage.site	com.ifds.eclipse.oe-coverage	0.0.1-20170907.165908-2
com.ifds.eclipse.oe-coverage.site	com.ifds.eclipse.oe-coverage	0.0.1-20170907.170828-3

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

Everyone Can See What's Happening

Continuous Integration is all about communication, so you want to ensure that everyone can easily see the state of the system and the changes that have been made to it.

One of the most important things to communicate is the state of the mainline build.



●	⛅	IFDS Code Coverage Eclipse Plugin	2 days 1 hr - #53	10 days - #49
●	☁	IFDS SonarQube OpenEdge Plugin	1 mo 14 days - #36	1 mo 16 days - #33
●	⛅	IFDS SonarQube ProLint Plugin	9 mo 8 days - #9	9 mo 8 days - #8
●	🌧	IFDS Unit Test ant task	14 days - #32	1 mo 16 days - #30
●	☀	IFDS Unit Test Tool	10 days - #21	6 mo 29 days - #10
●	☀	IFDS UTT Eclipse Plugin	4 days 0 hr - #20	1 mo 16 days - #15
●	☁	IFDS UTT Eclipse Update Site	2 days 20 hr - #33	2 days 21 hr - #30

Icon: S M L

Legend RSS for all

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

Automate Deployment

To do Continuous Integration you need multiple environments, one to run commit tests, one or more to run secondary tests. Since you are moving executables between these environments multiple times a day, you'll want to do this automatically. So it's important to have scripts that will allow you to deploy the application into any environment easily.

A natural consequence of this is that you should also have scripts that allow you to deploy into production with similar ease. You may not be deploying into production every day (although I've run into projects that do), but automatic deployment helps both speed up the process and reduce errors. It's also a cheap option since it just uses the same capabilities that you use to deploy into test environments.

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

Not a practice, but a good principle: Keep it Simple!

Start out with simple builds.

When something goes wrong that the build did not detect see whether you can add something that would have warned you beforehand. It can now become part of either a new version of the build or another parallel or subsequent build step.

As with code, refactor when it gets hairy into simpler, faster steps.

Once you have the build infrastructure in place there are a zillion things you will be able to do.

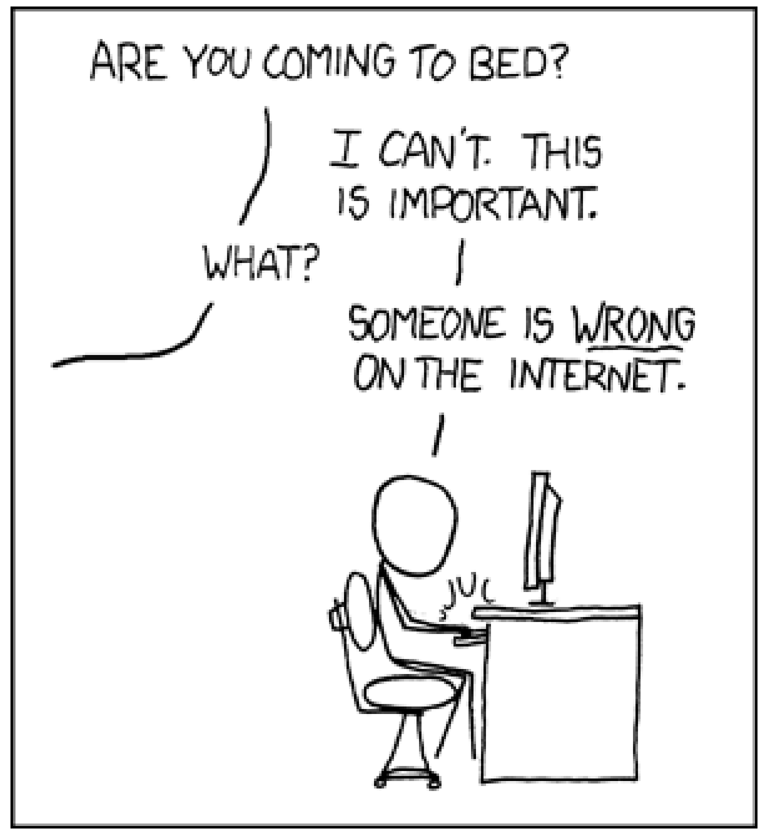
Common Obstacles

Common Obstacles

https://en.wikipedia.org/wiki/Perfect_is_the_enemy_of_good

A widely accepted interpretation of "The perfect is the enemy of the good" is that one might never complete a task if one has decided not to stop until it is perfect: completing the project well is made impossible by striving to complete it perfectly. Closely related is the Nirvana fallacy, in which people never even begin an important task because they feel reaching perfection is too hard.

https://xkcd.com/386/


ARE YOU COMING TO BED?
I CAN'T. THIS IS IMPORTANT.
WHAT?
SOMEONE IS WRONG ON THE INTERNET.

Public

Common Obstacles

https://en.wikipedia.org/wiki/Conway%27s_law

"organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."

*Conway, Melvin E. (April 1968), "How do Committees Invent?", Datamation, **14** (5): 28–31, retrieved 2015-04-10*

"If you have four groups working on a compiler, you'll get a 4-pass compiler.

Eric S. Raymond (October 1996). The New Hacker's Dictionary - 3rd Edition. ISBN 978-0-262-68092-9.

Common Obstacles

Brittle and complex tests

When we write tests it is intuitively simplest to consider an end-to-end test that fulfills the feature we are working on.
It’s easy to get going on these tests and it’s easy to understand them.
Unfortunately they do tend to be very brittle because they touch *everything*.
Depending how you do it, reordering fields on a screen (or adding a new one) may break your test.
And when it fails there can be many culprits.

Keep your tests simple – try to keep the proportion of unit tests high.

Common Obstacles

Prioritize Fixing the Build

When the build fails everyone needs to rally together to fix it.

Emails from your build server saying who broke the build usually will motivate sufficiently but sometimes we'll leave the build broken.

So long as the build is broken it completely stop helping you.
Developers check out and don't know if their local will work.
Developers who want to check in can't tell whether they have broken the build or if it was an existing issue.

Keep the build green!

Continuous Integration Tools (for Progress)

Bamboo

<APACHE ANT>

Joanju

SUBVERSION

RIVER SIDE software

Crontab

Progress

git

What is a Build?

Our starting point is typically running a build!

With languages that compile to an .exe or have a well-defined package format for some deployable unit this is well-defined but perhaps not always for languages that can be interpreted or which do not have such a package format.

But even if you do not create a .exe, the idea is that we duplicate a series of tests that can be run on the developers’ local machine on a server.

I strongly recommend that you make it so that the developer can themselves run these tests!

The starting point of a build is compiling all of your code, and this should succeed of course ☺

More sophisticated tests are possible and you can of course run tests that would take too long to be run by individual developers regularly, non-functional tests in particular are often set up this way.

What can we use to run the build?

I like to have something that will start it automagically.

This could just be a crontab entry!

Continuous Integration servers at their core are software that combine crontab with a VCS client and the command line tools that you use to build your software. That might just be a shell script or Progress program!

But **this is ALL YOU NEED:**

1. Some way to start a build when there are code changes
2. A place to run and test the system from #1
3. Some way to notify developers of the results of this

Your goal is to get these three steps working with the simplest possible variation of #2 initially.

Often #1 and #3 won't change once established…

Starting the build

A script run by cron that checks whether the revision in SVN has incremented and if so runs a build script is ok

A Continuous Integration server software provides essentially the same functionality, Jenkins actually asks you to specify this as a crontab entry ☺ (but it does provide help as pretty much everyone needs to look this up…)

There are a few CI servers, I personally started out using something called AnthillPro but these days Jenkins and Bamboo are probably the most popular CI servers, although there are lots of others.

Jenkins is open source, originally Hudson Jenkins is a fork as Hudson was a Sun open source project.

Bamboo is from Atlassian, the makers of JIRA. Although not open source Atlassian products are super cheap for small teams of up to ten they're $10.

What can we use to run the build?

The build can just be a Progress program that COMPILE's everything, this could be run by a script.

There are some Domain Specific Tools for build-ing, there is a long history to this -- probably everyone here has heard of Make?

Modern build tools have a lot more functionality and are more easily extensible, I won't go into that too much except to say that if you start with ant you give yourself a lot of useful options, they'll be easy to add to the build and others will often be able to read and understand your build scripts.

Ant = "ANother Tool" is a Java build tool that allows you to specify build targets in an XML file (it was the thing at that time.)

Gilles Querret from Riverside Software has written an excellent Open Source plugin for this called PCT = Progress Compile Tools that allows us to leverage this easily.

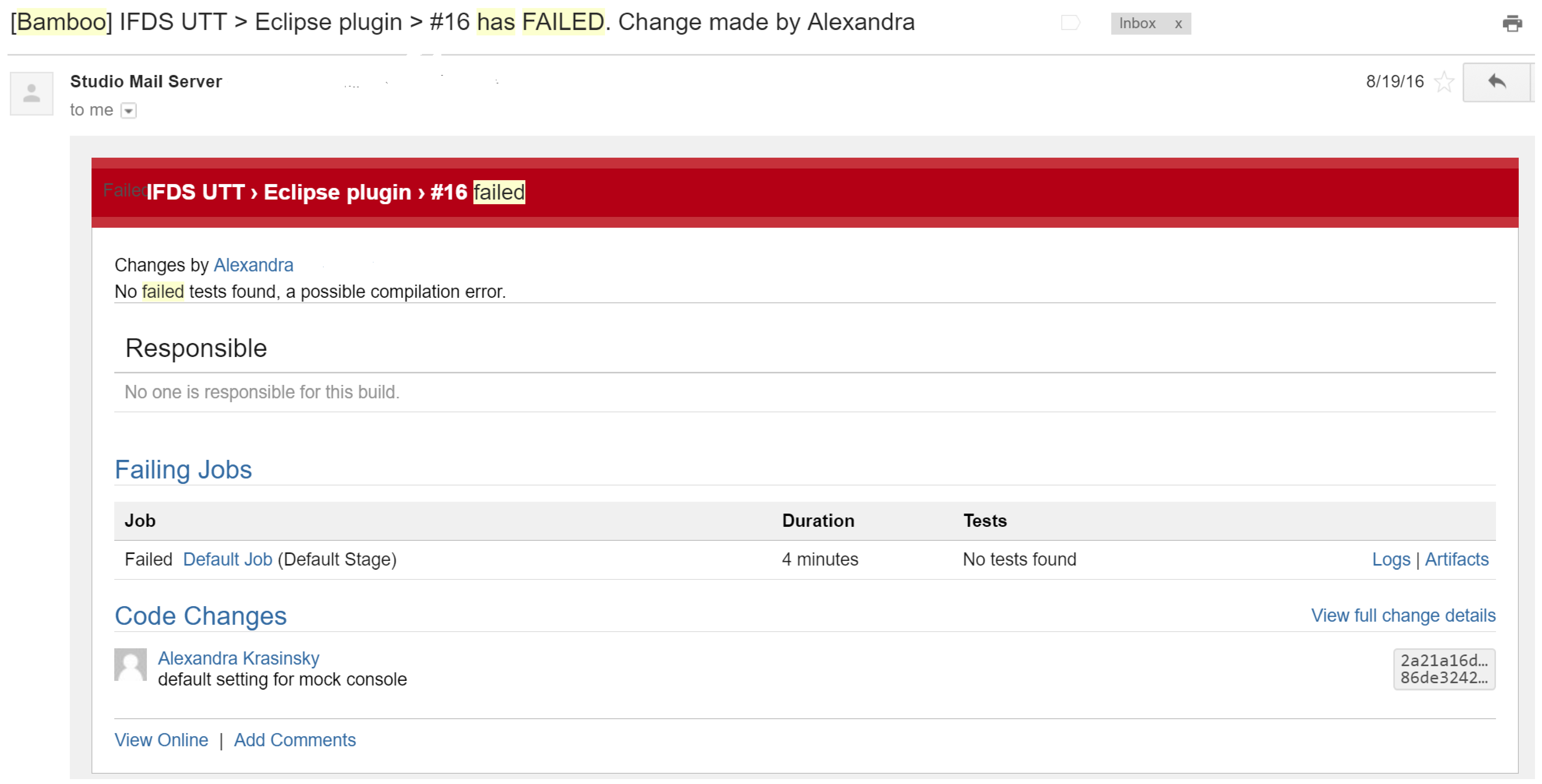
http://ant.apache.org

https://github.com/Riverside-Software/pct

Notify everyone!

If you’re using a CI server it will have a projects page that shows the status.

What is often a lot more useful is an email notifying everyone that the build failed.



[Bamboo] IFDS UTT > Eclipse plugin > #16 has FAILED. Change made by Alexandra

Inbox x

Studio Mail Server
to me

8/19/16

IFDS UTT › Eclipse plugin › #16 failed

Changes by Alexandra
No failed tests found, a possible compilation error.

Responsible

No one is responsible for this build.

Failing Jobs

Job	Duration	Tests	
Failed Default Job (Default Stage)	4 minutes	No tests found	Logs | Artifacts

Code Changes — View full change details

Alexandra Krasinsky
default setting for mock console

2a21a16d...
86de3242...

View Online | Add Comments

Unit Tests

Unit Testing is a development practice where small “units” of code are tested in isolation.

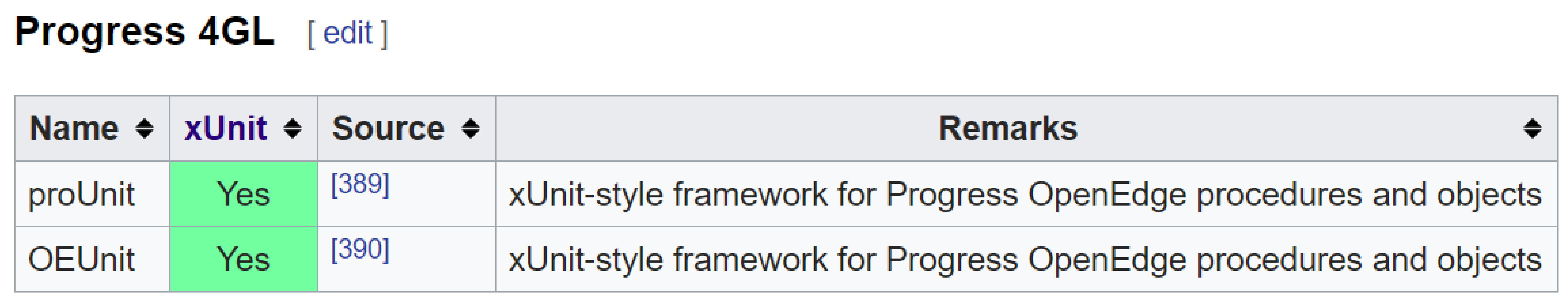
Even in large legacy systems this can be a big win, although breaking off isolated pieces may be more difficult when you have shared variables and database state to deal with.

xUnit frameworks to support such testing date back to the Sunit.

SUnit was originally described by Beck in "Simple Smalltalk Testing: With Patterns" (1989)

These days there is JUnit for Java, NUnit for .NET, and variations for Javascript and many other languages.

As of Progress 11.5 a simple implementation is included and integrated within PDSOE, there are also a couple of Open Source implementations in Progress – per https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#Progress_4GL



Progress 4GL [edit]

Name	xUnit	Source	Remarks
proUnit	Yes	[389]	xUnit-style framework for Progress OpenEdge procedures and objects
OEUnit	Yes	[390]	xUnit-style framework for Progress OpenEdge procedures and objects

What's in an xUnit?

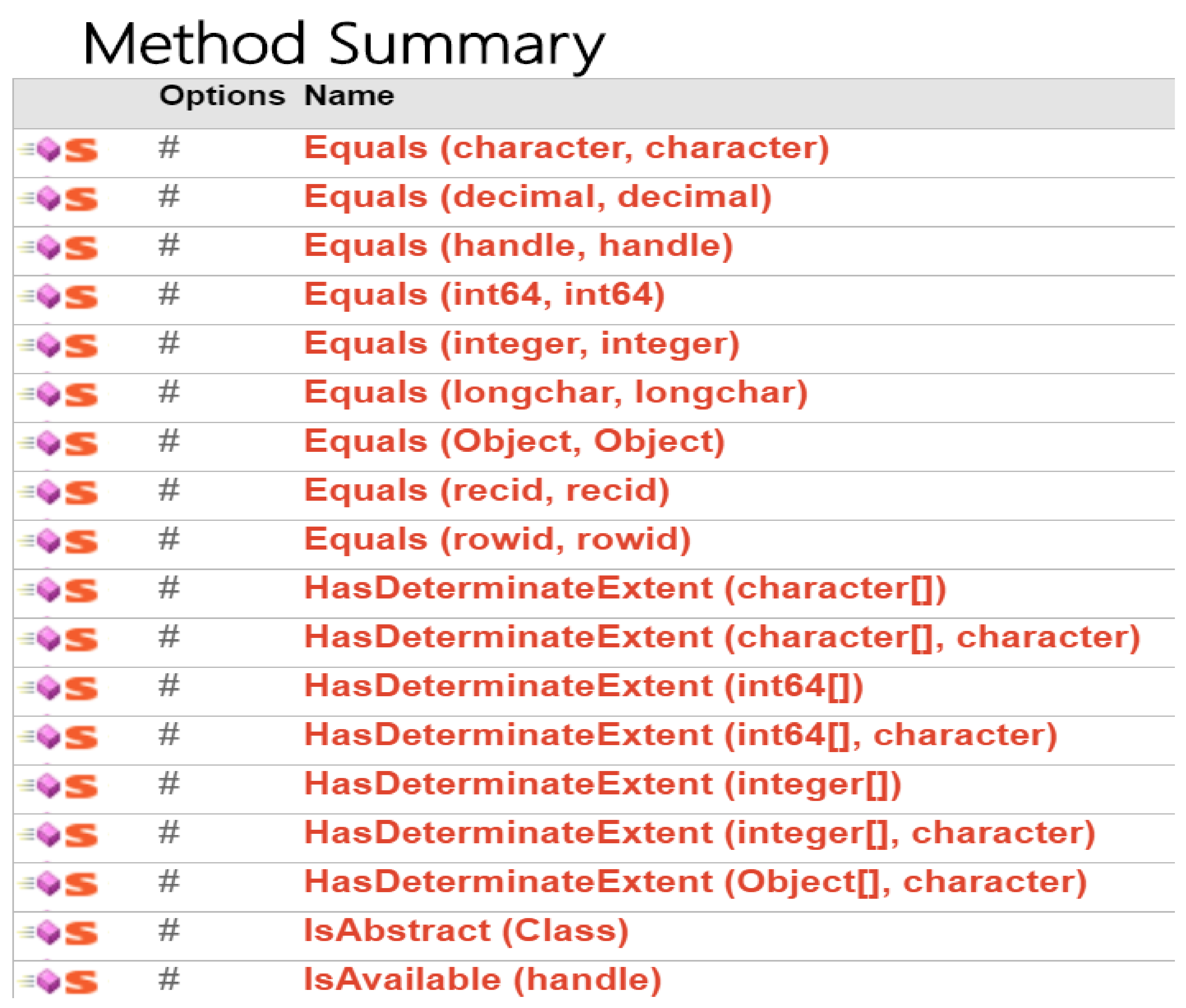
xUnit is a test runner but typically will also provide some way to make assertions.

What I mean by test runner is that some automated process will run your tests, which are small procedures, as part of the build. You can annotate your tests with lifecycle methods which are used to indicate the order of operations, as you usually want to prepare a test fixture before you run a test.

Marking Test Procedures

@Test.
@Setup.
@Before.
@Teardown.
@After.

Assertions



Method Summary

Options	Name
#	Equals (character, character)
#	Equals (decimal, decimal)
#	Equals (handle, handle)
#	Equals (int64, int64)
#	Equals (integer, integer)
#	Equals (longchar, longchar)
#	Equals (Object, Object)
#	Equals (recid, recid)
#	Equals (rowid, rowid)
#	HasDeterminateExtent (character[])
#	HasDeterminateExtent (character[], character)
#	HasDeterminateExtent (int64[])
#	HasDeterminateExtent (int64[], character)
#	HasDeterminateExtent (integer[])
#	HasDeterminateExtent (integer[], character)
#	HasDeterminateExtent (Object[], character)
#	IsAbstract (Class)
#	IsAvailable (handle)

Obstacles for Writing Unit Tests and Overcoming them

Legacy programs often are not written to be testable. This is not a lack of any older language itself *per se* it is simply without this additional ability to test sections of code in isolation easily we often fail to structure our programs to make it easy to do so. Unfortunately this tends to be a chicken and egg because legacy code is risky to change to restructure it.

This has further costs because we then must do end-to-end tests which have a high setup and execution cost. Additionally, debugging can be time consuming when a failure can occur in a long series of modules required to set up a test.

Older Progress code using shared variables may suffer further in that a large number of shared variables may be expected to instantiated in a parent program scope.

Even newer code often may call other procedures to carry out their work, this is a feature of structured programming but now if we make use of those procedures in our code they may call other procedures and the tables that they need must be established with the correct data and any expected shared variables set up…

Here at IFDS we have created a Unit Test Tool to mock code as is supported in object oriented languages such as Java. Your mileage may vary… trust to structured programming and make use of the preprocessor if you can to limit the scope of code under test.

Coding Time!

Coding Time!

A note about installations:

You will already have progress installed.
You can install Java from the Oracle site, but OpenJDK works too.

Ant: http://ant.apache.org/bindownload.cgi

PCT: https://github.com/Riverside-Software/pct/releases

Jenkins: https://jenkins.io/download/

Ant Basics

Ant Tasks

https://ant.apache.org/manual/tasksoverview.html

- Archive Tasks
- Audit/Coverage Tasks
- Compile Tasks
- Deployment Tasks
- Documentation Tasks
- EJB Tasks
- Execution Tasks
- File Tasks
- Java2 Extensions Tasks
- Logging Tasks
- Mail Tasks
- Miscellaneous Tasks
- Pre-process Tasks
- Property Tasks
- Remote Tasks
- SCM Tasks
- Testing Tasks

These are groups of tasks.. There are a LOT!

Not all are applicable to Progress.

Custom Ant Tasks

One of the things that led to the success of ant was that it is extensible.

Gilles' Progress Compile Tools can be used by ant so that you can natively perform Progress tasks from ant.
Under the hood a temporary .P is generated that is configured based on the declarative XML in your build file.

We import a task through a <taskdef> element

PCT

One of the things that led to the success of ant was that it is extensible.

Gilles' Progress Compile Tools can be used by ant so that you can natively perform Progress tasks from ant.
Under the hood a temporary .P is generated that is configured based on the declarative XML in your build file.

We import a task through a <taskdef> element

Let's do it!

The top level element is <project>

Build goals are <target>

We configure things with a mix of child elements and attribute values, the documentation has a lot of examples so when you want to use a task visit the Wiki or ant docs and shoomf to the bottom of the page.

Children of <target> are run generally top to bottom.

However a target may specify other targets are prerequisites in a depends= attribute.

Compile

PCTCompile is, naturally, used to compile.

<propath> is a list of <pathelement>’s
You do also need one or more <fileset>’s to specify which files to compile.

You also need a database connection, starting it as I’ve shown here means you don’t need a pre-existing db

Note that in this example I’m using an empty db

Running a Test

If you’re using ABLUnit then that is the element name to run the test.

You’ll need the database connection too generally, I also recommend starting from an empty db but your mileage may vary.

Debugging your ant script

Two useful things:

1. Run ant with –v(erbose)

2. For PCT scripts, PCTRun which is the base task for many commands has a ‘debugPCT’ option which will keep PCT temporary files on disk so you can go and look at them.

Now let’s run this from Jenkins

You’ll install Jenkins in a manner appropriate for your OS from:

https://jenkins.io/download/

The release I’m showing you isn’t the latest but the concept is the same.
Other CI servers are generally easy to install and can do what I’m showing you.

Creating a job

We want to poll our scm for changes so we can build when someone commits.

For this example I’ve created a local git repository in the file system but generally you will poll a remote SCM system across the network.
We then will check it out and run ant.

The build can also email notifications if you’ve got an SMTP server you can use.

I also recommend showing a web page somewhere visible to your dev team with at least the current build state.

If the build succeeded initially then here we’ll show a failure

But it may be vice-versa as this is a demo… we’ll roll with it ☺

This first build is just a gateway (usually)

Once you have that first build though you can launch more involved builds if the first build succeeds.

Then you can deploy that somewhere and run your QA smoketest against it using HP ALM or whatever tool(s) they may be using…

Thanks for coming! Questions?

APPENDIX

Useful things:

PCT https://github.com/Riverside-Software/pct

Ant http://ant.apache.org/

Jenkins https://jenkins.io/download/

Atlassian https://www.atlassian.com

Java (since ant, Jenkins and Atlassian tools are written in Java) https://java.com/en/download/

Gilles: http://riverside-software.fr/

ProLint: http://www.oehive.org/prolint

ProParse: http://www.joanju.com and http://www.oehive.org/proparse/

APPENDIX

Fun things:

xkcd: https://xkcd.com/

"A webcomic of romance, sarcasm, math, and language."

Don't Hit Save: http://donthitsave.com/

Rick and Morty: http://www.adultswim.com/videos/rick-and-morty/