COMP8005 – Applied Web Development

Continuous Integration

Introduction

- We will look at the important role Continuous Integration (CI) has in agile development
- We will look at CI in action
- We will examine how CI is important to facilitate automated delivery

Continuous Integration

- An agile development practice
- 2 of the 12 principles from the agile manifesto
  - Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
  - Working software is the primary measure of progress.
- Continuous Integration facilitates both

Continuous Integration

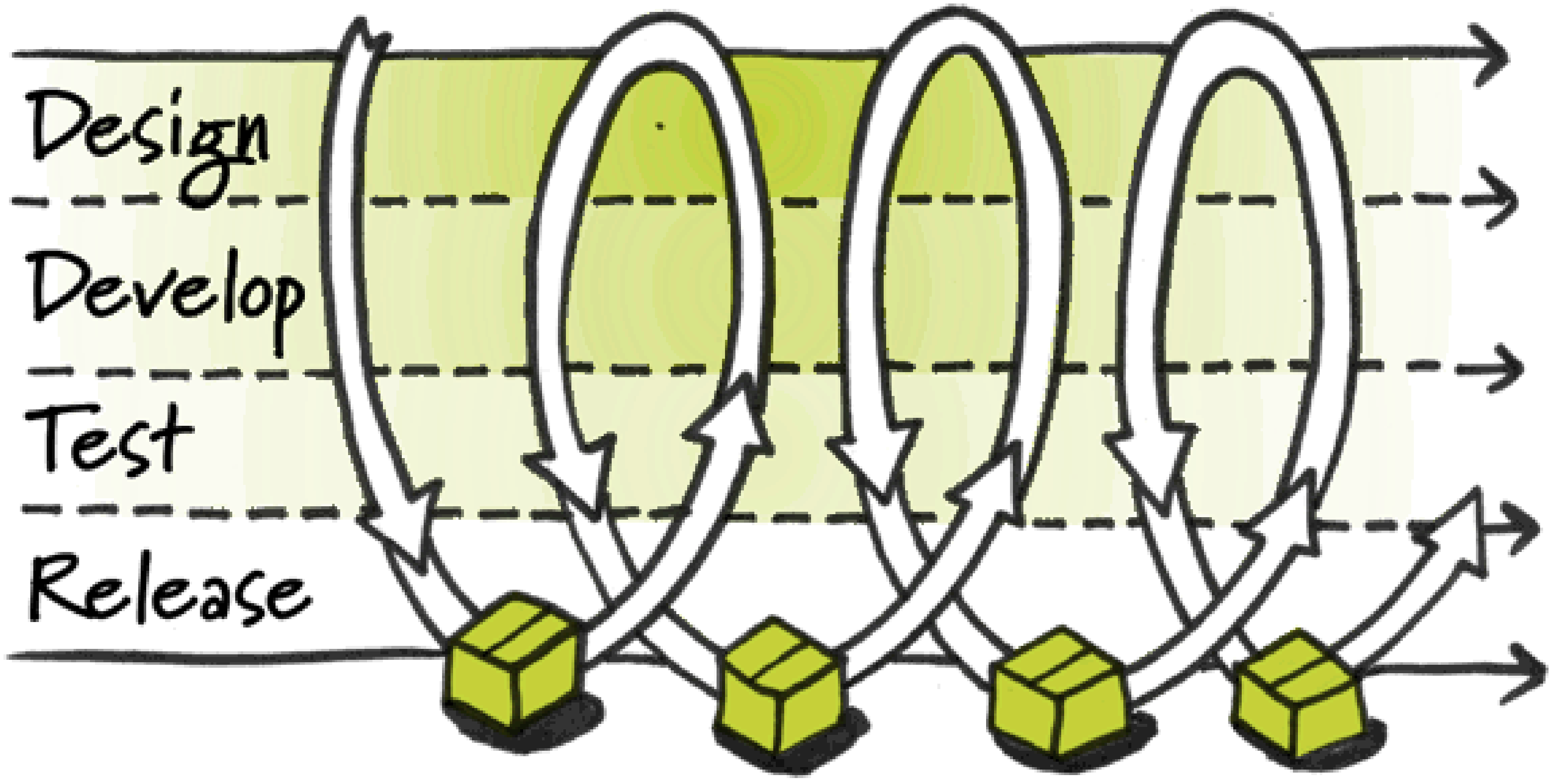
- *"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software"*
- Customer needs to see the software frequently, and from an early stage
- From day one, need to think about integrating the work of all developers
- This enables early prototypes to be visible to the customer
- But perhaps more importantly…

Continuous Integration

- *"Working software is the primary measure of progress"*
- Knowing that all of our components are integrating and that the system is working (or even partially working, since with agile practices you are continuously prototyping)
- Being able to see at a glance that everyone's work isn't in conflict – i.e. new component is checked in and doesn't affect any other checked in components

Continuous Integration

- CI, of course, is a continuous iterative process



Continuous Integration

- You begin with a shared code repository
- Examples include Subversion, Git and Team Foundation Server
- These allow developers to "check out" code (or develop a new component from scratch), develop/modify the component, and check it (back) into the shared repository
- This is done at least daily

Continuous Integration

- Therefore, you should never be too far away from working code
- This encourages you to design and build components that are cohesive (i.e. concentrate on one "concern") and loosely coupled
    - You don't want to have to change component A and then discover that there is a fundamental knock on effect B, C, D and E
    - Refactoring, aside – you can make changes to interfaces, for example, if it improves the design

Continuous Integration

- "Check in at least daily"
- Ideally more than once a day
- By doing this, you encounter issues earlier when it is easier to fix them
  - Issues are caused by smaller units of work that you have checked in, so easier to trace back to source
  - It is immediate and fresh in your mind, so you'll often have that light bulb light up immediately as you remember where the source of the problem is likely to be

Continuous Integration

- The longer the integration is left (more than a day, e.g. weekly) the more difficult the integration becomes
- If you decide to only integrate prior to important milestones, e.g. just before that QA release to the customer, integration becomes a stressful event leading possibly to one or more late nights getting things integrating properly

Continuous Integration

- In that respect, CI is the anti-procrastination practice, like the student who does a little each day rather than cramming at the end of the semester
- Cramming can be counterproductive just like last-minute big integrations
- So there is a psychological component to CI
  - Confidence at all times that you know where you are and that you are on track with few shocks ahead

Continuous Integration

- Benefits of CI as proposed by ThoughtWorks (for whom Martin Fowler is an employee):
- http://www.thoughtworks.com/continuous-integration
  - Say goodbye to long and tense integrations
  - Increase visibility which enables greater communication
  - Catch issues fast and nip them in the bud
  - Spend less time debugging and more time adding features

Continuous Integration

- CI benefits continued…
  - Proceed in the confidence you're building on a solid foundation
  - Stop waiting to find out if your code's going to work
  - Reduce integration problems allowing you to deliver software more rapidly

10 Principles of CI

- There are 10 core principles of CI
  - 1. Maintain a code repository
    - Everything required to build the system should be in the repository and ideally should include all dependencies.
    - Version control should be supported, allowing changes to be rolled back if necessary

10 Principles of CI

- Avoid branching if possible. Branching is where the repository splits and development on each branch happens in isolation. Some changes can be applied from one branch to another, but others remain isolated. Example would include development towards a new platform on one branch and legacy support on another.

10 Principles of CI

- 2. Automate the build
  - A single command or script should be able to build the entire system
  - Traditionally, *make* was used to build software and link up dependencies
    - Still fairly common to download software for unix/linux in source code format (e.g. C code) and do a *make* followed by *install*
    - More common now, though, to use RPM or DEB packages, or a shell script that packages binary within in

10 Principles of CI

- Many build tools have emerged since *make*, such as Ant and then Maven
- See a comprehensive list at:
- http://en.wikipedia.org/wiki/List_of_build_automation_software

10 Principles of CI

- 3. Make the build self-testing
  - We discussed last week using examples how to practice Test-Driven Development
  - I mentioned that this helps to facilitate CI
  - After the build, it should be possible for all tests to be run and the results of the tests known

10 Principles of CI

- 4. Everyone commits to the baseline every day
  - Find conflicts early when they are small and easily identifiable
  - Also, commit once a feature has been built – if it is complete and tested, why wait until the end of the day to find out whether it integrates?
  - A nightly automated build is useful to have a report ready the next morning for developers

10 Principles of CI

- 5. Every commit to the baseline repository should trigger a new build
  - Ideally, a CI server will be in place – we will discuss examples shortly
  - This allows for automation of builds and can be configured to build the system after each commit (assumed the CI server is constantly monitoring the version control system for changes)

10 Principles of CI

- 6. The build should be quick
  - If the system takes too long to build, then CI's value decreases
  - Quick builds help identify issues faster

10 Principles of CI

- 7. Test in a clone of the production environment
  - No good if the test environment is slightly different than the production environment
  - This is usually a pragmatic decision based on cost
  - Ideally test environment should be a scaled down replica of production – e.g. smaller disks, fewer nodes, less memory. But the technology stack (OS, server software, etc) should be the same

10 Principles of CI

- 8. Make it easy to get the latest deliverables
  - Stakeholders should have access to builds so that feedback can be received early
  - Examples include milestone builds and nightly builds on a website
    - E.g. http://docs.spring.io/downloads/nightly/snapshot-download.php?project=ROO

      has the last five nights' snapshots for Spring ROO

10 Principles of CI

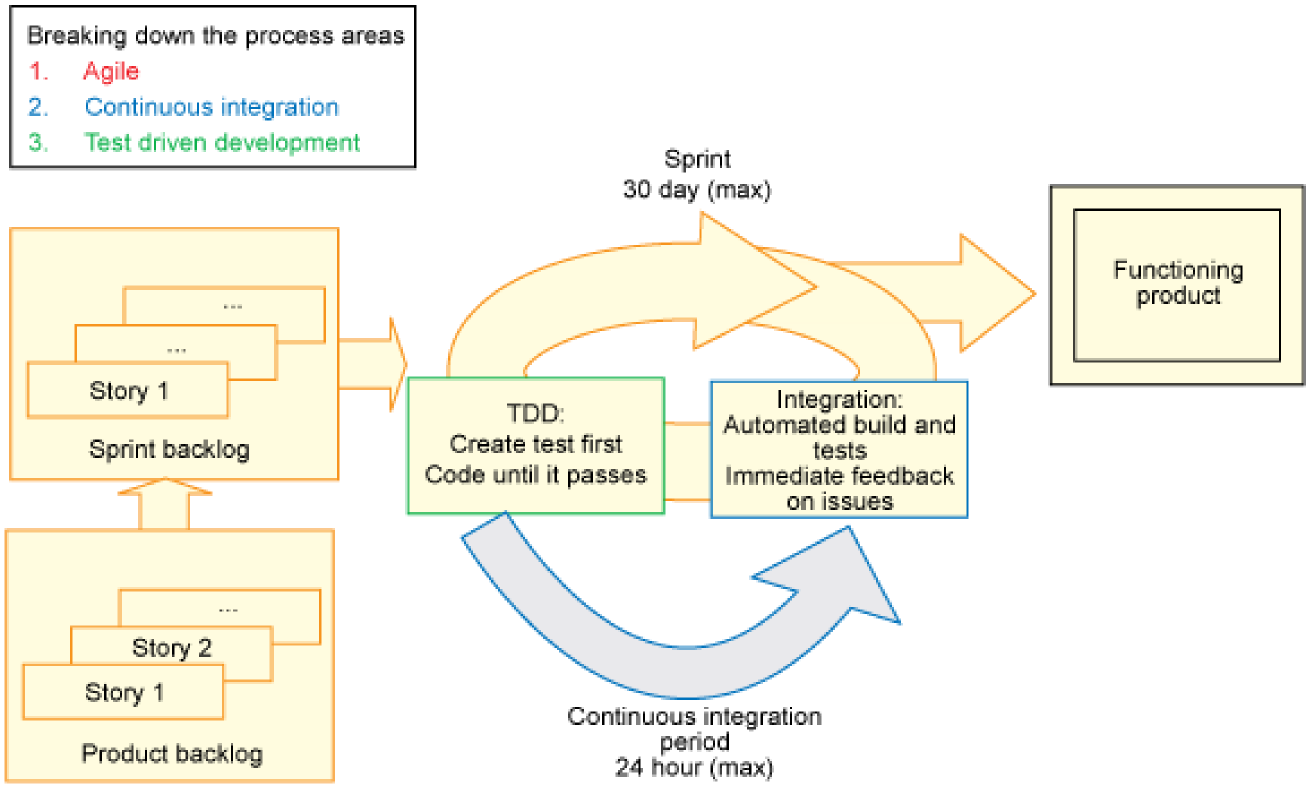
- 9. Everyone can see the results of the latest build
  - Did the last build break and if so who caused it
  - Should never be used as a tool to victimise developers
  - Finding issues is a good thing because when it is early, it is cheap to resolve

10 Principles of CI

- 10. Automate deployment
  - Should be possible to follow up builds automatically with a deployment
  - Build, deploy, run or browse the end result
  - Usually deployed to a test environment
  - Deploying straight to production would be a real no-no, wouldn't it???? (we'll come back to this)

CI in the bigger picture

- TDD and CI in an Agile project



CI servers

- CI server runs independently of the developer and the source code repository
- CruiseControl was an early example and became popular, particularly with Java developers
- But it has not been updated since 2010

CI Servers

- Jenkins – originally started as Hudson in Sun Microsystems
- However, Sun was taken over by Oracle and there was a dispute about its infrastructure and ultimately Oracle's ownership of it
- It was open source, so there was a fork of Hudson that was named Jenkins in 2011
- There are extensive plugins, making Jenkins popular with many

Jenkins

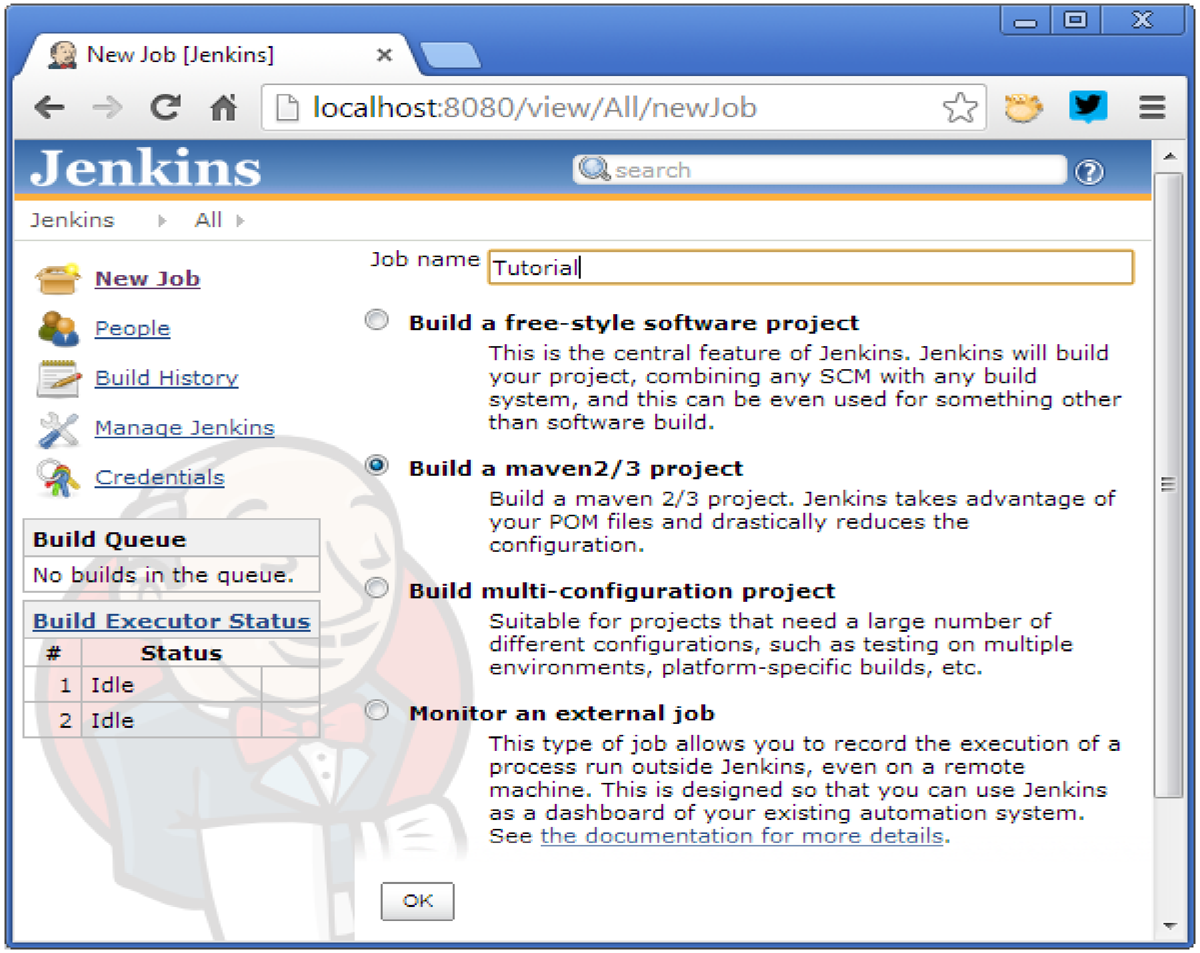
- Native support for CVS and Subversion version control systems
- But plugins available for many more, such as mercurial, Git and TFS
- Various other plugins support Android, PHP, .NET, Ruby, iOS, Maven and lots more

Jenkins

- Easy to install Jenkins using an installer (e.g. MSI on Windows)
- By default is accessible via web browser on port 8080

Jenkins and Maven

- Here, we can select Maven out of the box



New Job [Jenkins]

localhost:8080/view/All/newJob

Jenkins

search

Jenkins ▸ All ▸

- New Job
- People
- Build History
- Manage Jenkins
- Credentials

Build Queue

No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

Job name: Tutorial

( ) **Build a free-style software project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

[✓ **Build a maven2/3 project**]
Build a maven 2/3 project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

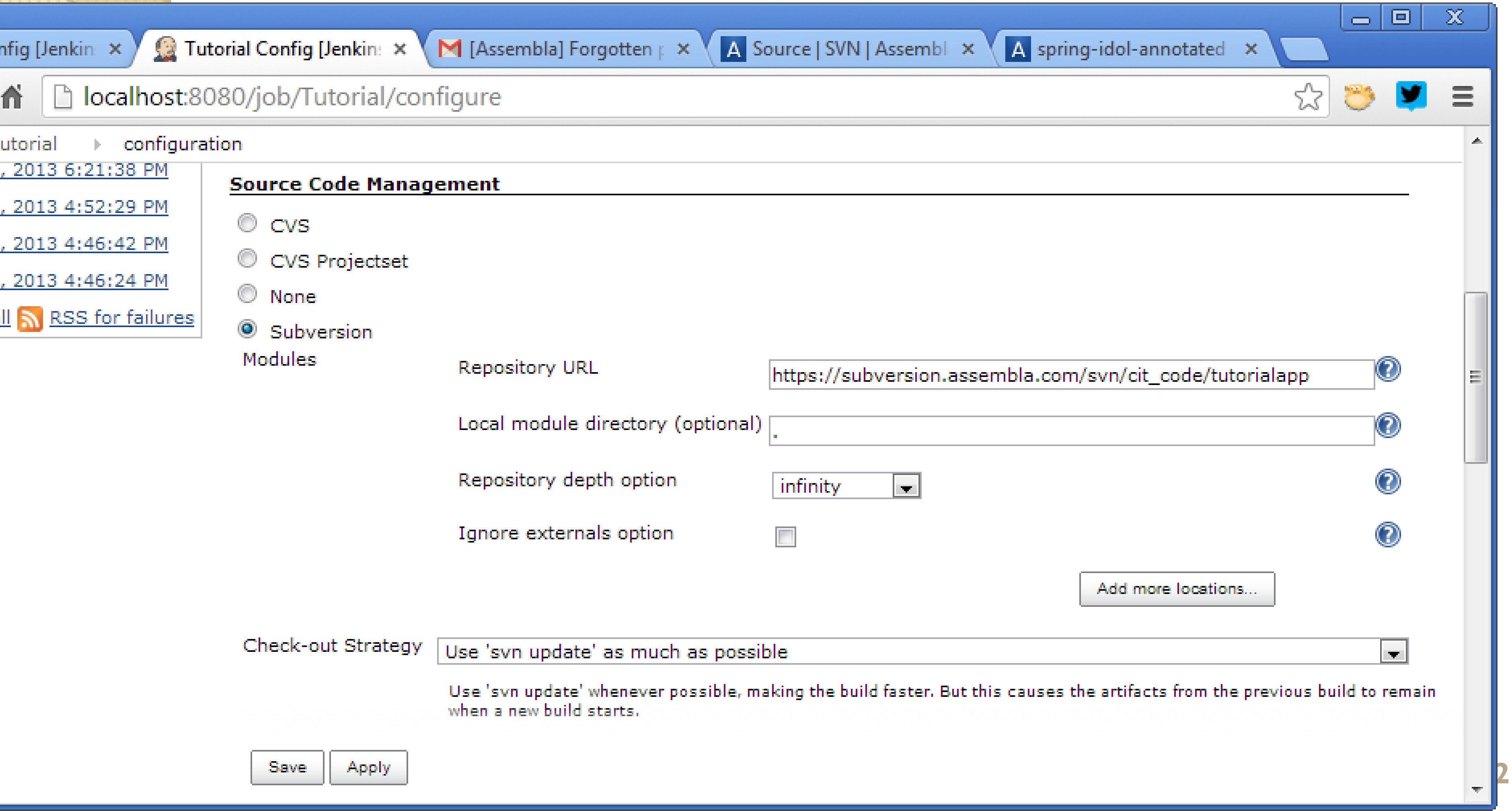
( ) **Build multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

( ) **Monitor an external job**
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See the documentation for more details.

OK

Jenkins and SVN

- Need to setup a project from a repository



Source Code Management

- CVS
- CVS Projectset
- None
- [✓ Subversion]

Modules

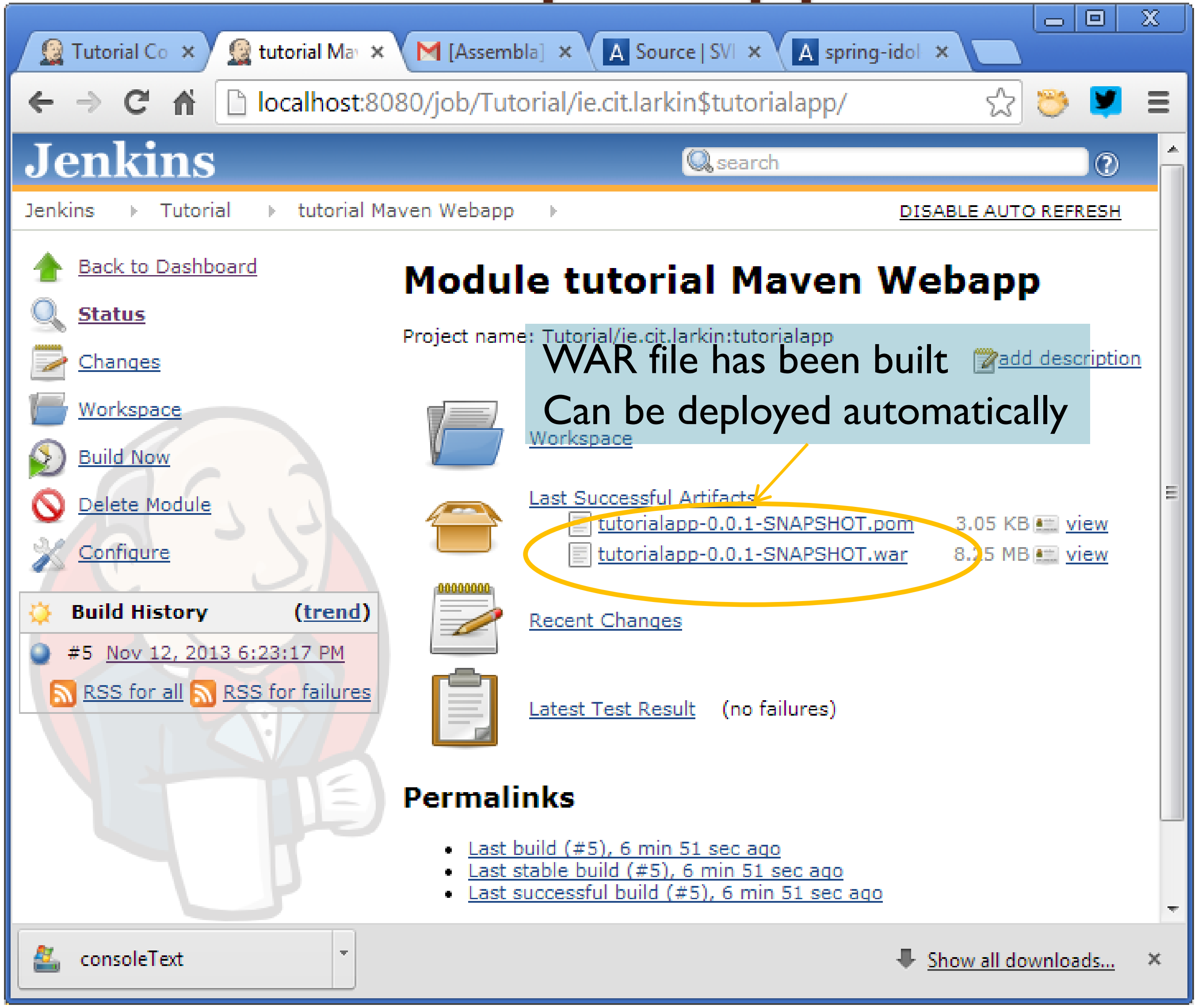
Repository URL	https://subversion.assembla.com/svn/cit_code/tutorialapp
Local module directory (optional)	.
Repository depth option	infinity
Ignore externals option	☐

Check-out Strategy: Use 'svn update' as much as possible

Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.

Jenkins and a Sample App

Project Dash-board



localhost:8080/job/Tutorial/ie.cit.larkin$tutorialapp/

Jenkins » Tutorial » tutorial Maven Webapp

DISABLE AUTO REFRESH

- Back to Dashboard
- Status
- Changes
- Workspace
- Build Now
- Delete Module
- Configure

Build History (trend)

#5 Nov 12, 2013 6:23:17 PM

RSS for all RSS for failures

Module tutorial Maven Webapp

Project name: Tutorial/ie.cit.larkin:tutorialapp

add description

Workspace

Last Successful Artifacts

tutorialapp-0.0.1-SNAPSHOT.pom	3.05 KB	view
tutorialapp-0.0.1-SNAPSHOT.war	8.25 MB	view

Recent Changes

Latest Test Result (no failures)

Permalinks

- Last build (#5), 6 min 51 sec ago
- Last stable build (#5), 6 min 51 sec ago
- Last successful build (#5), 6 min 51 sec ago

consoleText

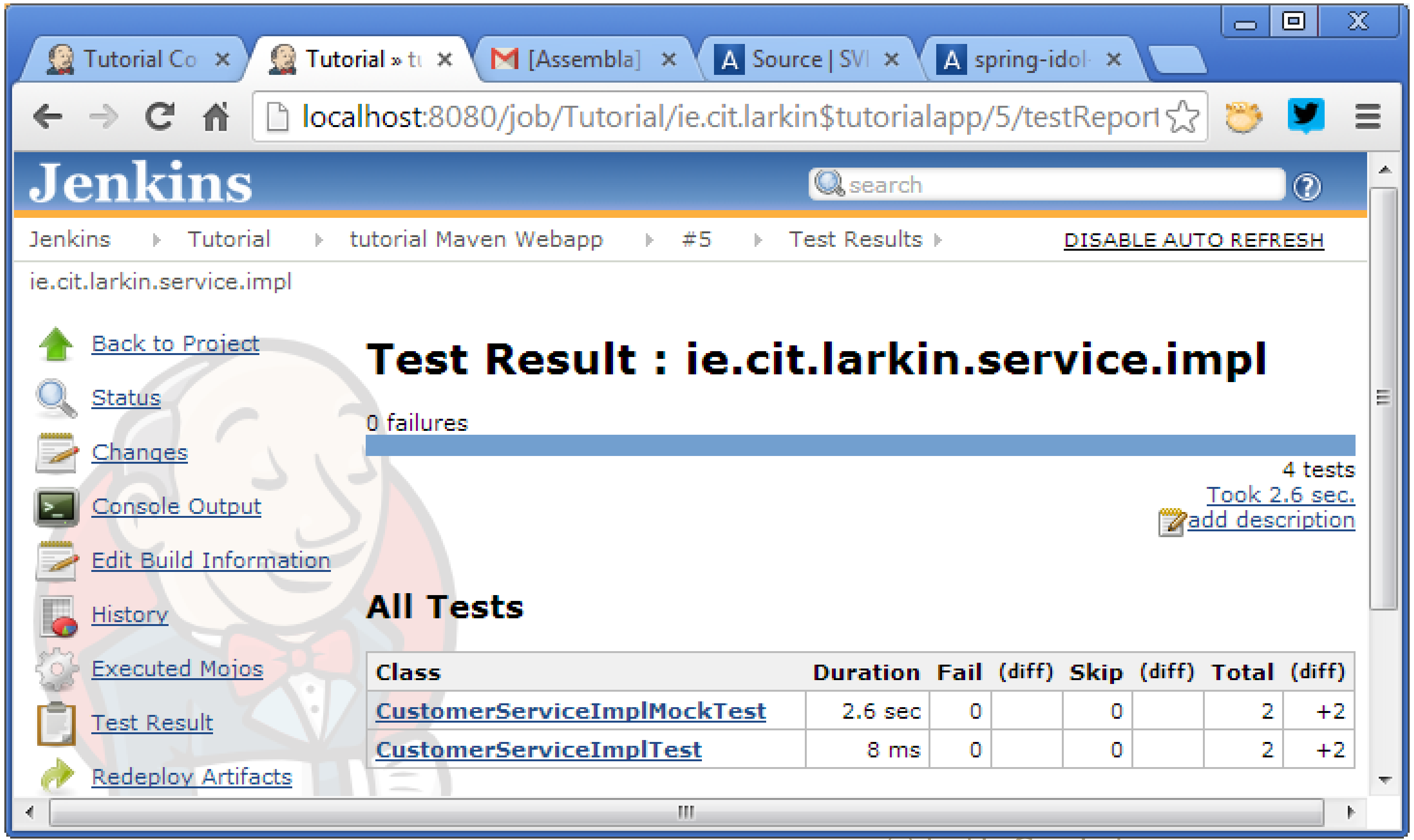
Show all downloads...

WAR file has been built

Can be deployed automatically

Jenkins

- Automated tests run



Tutorial Co | Tutorial » tu | [Assembla] | Source | SVI | spring-idol

localhost:8080/job/Tutorial/ie.cit.larkin$tutorialapp/5/testReport

Jenkins

Jenkins ▸ Tutorial ▸ tutorial Maven Webapp ▸ #5 ▸ Test Results ▸ DISABLE AUTO REFRESH

ie.cit.larkin.service.impl

Back to Project
Status
Changes
Console Output
Edit Build Information
History
Executed Mojos
Test Result
Redeploy Artifacts

Test Result : ie.cit.larkin.service.impl

0 failures

4 tests
Took 2.6 sec.
add description

All Tests

Class	Duration	Fail	(diff)	Skip	(diff)	Total	(diff)
CustomerServiceImplMockTest	2.6 sec	0		0		2	+2
CustomerServiceImplTest	8 ms	0		0		2	+2

Jenkins

- Some links regarding Jenkins and PHP (and Laravel, etc)
  - http://jenkins-php.org/
  - http://programmingarehard.com/2013/09/26/zero-to-jenkins.html

Other CI servers

- Jenkins is a popular free open source CI server
- There are other open source options
  - Apache Gump
  - Apache Continuum
- Jenkins is by far the most popular
- There are also commercial offerings from some big players, particularly in the Cloud
  - Assembla, Microsoft, Atlassian, etc
- These often integrate with other tools, e.g. Agile project management, Version Control, etc

Continuous Delivery

- Facilitated by automated tests and CI
- A suite of tests is devised that, on passing, a system can be deployed to a test environment
- A suite of acceptance tests are run manually and, on passing all of them, deployment is to production
- However, could bypass test (or staging) environment and deploy straight to production – you would want to be sure of your processes

Continuous Delivery

- Whereas CI relies on unit and integration tests, CD relies on business logic / use case tests
- These might be manual with, for example, a number of users typing values into forms
- Or can be automated using an interface test tool, such as selenium or similar

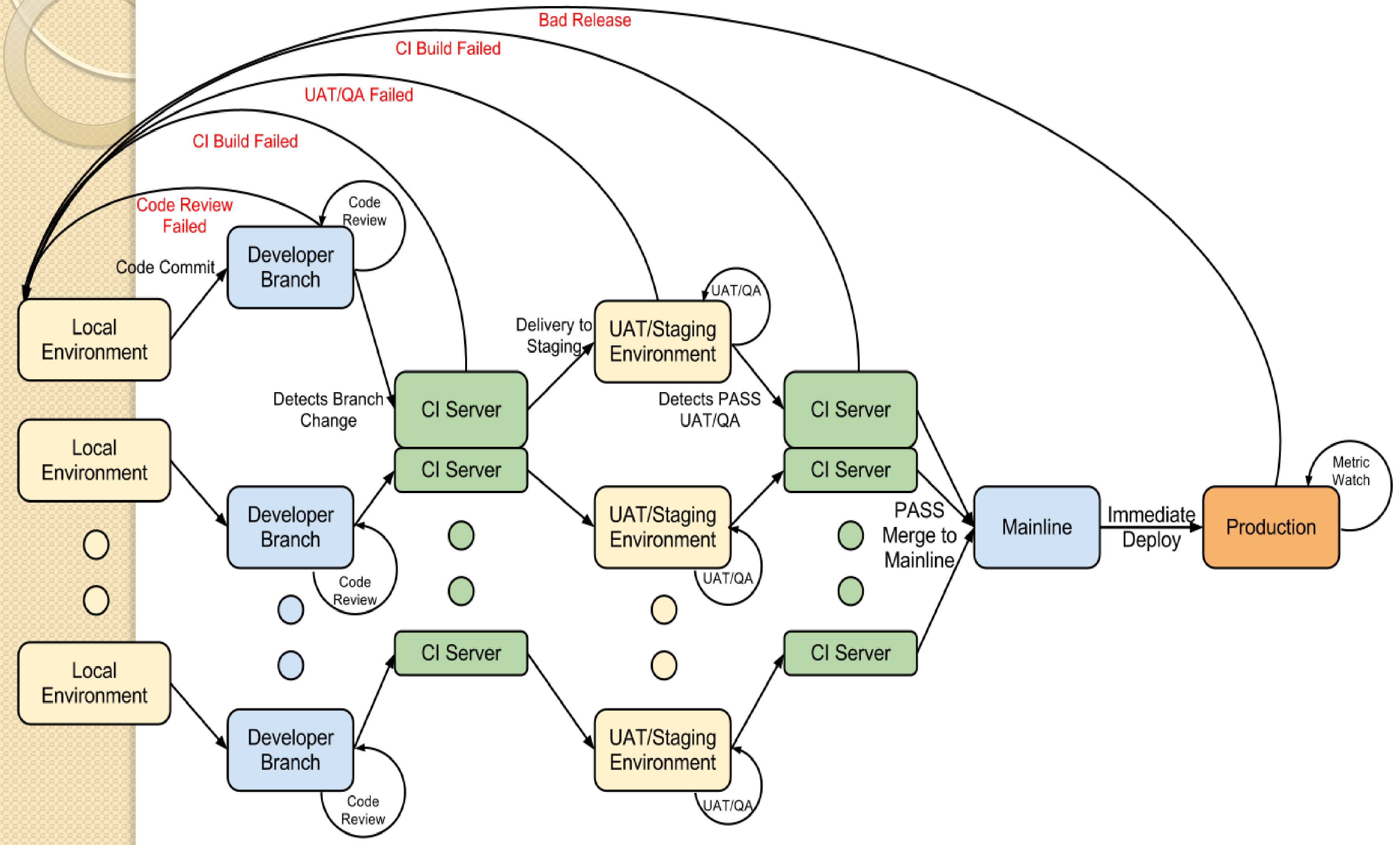
Continuous Deployment

- Another CD!!! (confusing)
- Is it different than Continuous Delivery?
- Generally speaking, Continuous Deployment is automated deployment directly to production
- Automated testing occurs in a production-like environment to determine if code can be merged with production branch in central repository and then deployed

CI, CDel, CDep

- See this blog post comparing CI, CDel, CDep: Click here
- "In the ideal workflow, the entire process could be automated from start to finish: Developer checks in code to development branch, Continuous Integration Server picks up the change, performs Unit Tests, votes on the Merge to Staging environment based on test results, if successful deploys it to Staging Environment, QA tests the Environment, if passed, they vote to move to Production, Continuous Integration server picks this up again and determines if it is ok to merge into Production, if successful, it will deploy to Production environment. This process varies slightly based on needs, requirements and approaches."

CI, CDel, CDep



Bad Release
CI Build Failed
UAT/QA Failed
CI Build Failed
Code Review Failed
Code Review
Code Commit
Developer Branch
Local Environment
UAT/QA
Delivery to Staging
UAT/Staging Environment
Detects Branch Change
CI Server
Detects PASS UAT/QA
CI Server
Local Environment
CI Server
CI Server
Metric Watch
Developer Branch
UAT/Staging Environment
PASS Merge to Mainline
Mainline
Immediate Deploy
Production
Code Review
UAT/QA
Local Environment
CI Server
CI Server
Developer Branch
UAT/Staging Environment
Code Review
UAT/QA

Links

- Perhaps the authorative voice on CD, Jez Humble: http://continuousdelivery.com/
- http://puppetlabs.com/blog/continuous-delivery-vs-continuous-deployment-whats-diff
  - *"Continuous Delivery doesn't mean every change is deployed to production ASAP. It means every change is proven to be deployable at any time" – Carl Caum*

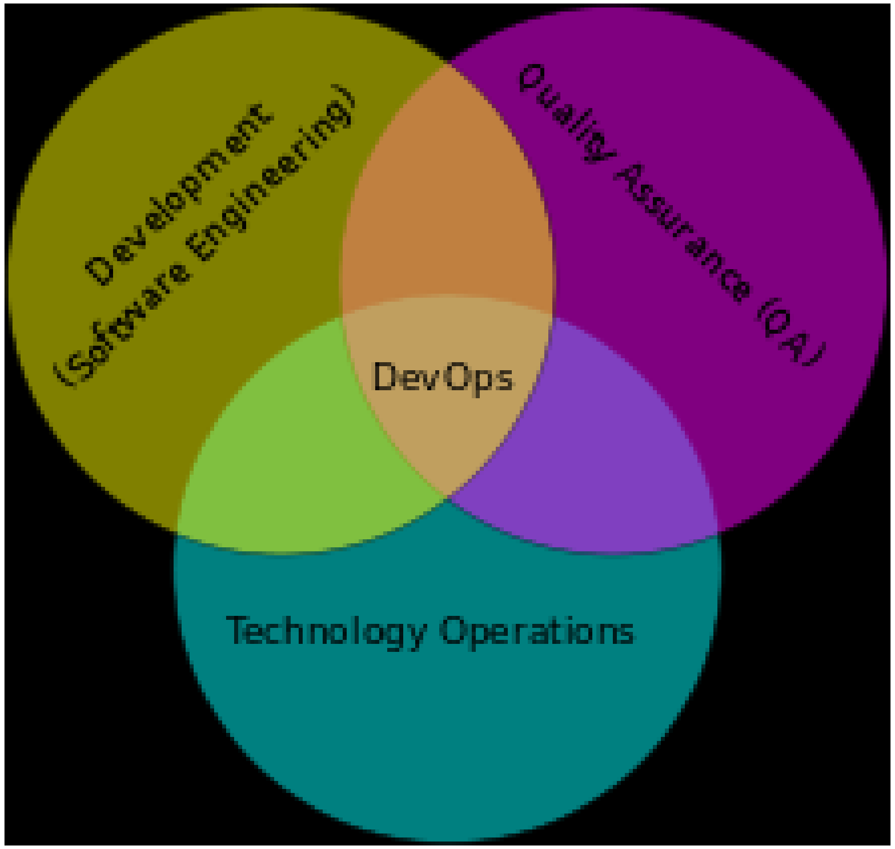
DevOps

- Continous Delivery is central to the emerging concept of DevOps
- DevOps – Development-Operations
- Breaks down the "wall of confusion" between developers and operations staff
- Development and Operations usually in a separate department
- Division of labour
- Each has different goals, different ways of measuring progress, etc.

DevOps

- DevOps takes a "systems approach" to the development, delivery, deployment of software systems
- **Systems thinking** is the process of understanding how things, regarded as systems, influence one another within a whole.
- See http://doras.dcu.ie/17556/1/Computer_Eng_chap.pdf (co-authored by Rory O'Connor of DCU)
  - *Towards A Wider Application of the Systems Approach in Information Systems and Software Engineering*
- In other words, DevOps is a "Holistic" approach
- Seen as particularly important in the Cloud context

DevOps


Development
(Software Engineering)
Quality Assurance (QA)
DevOps
Technology Operations

DevOps

- In the Cloud: so many moving parts
- Developers using lots of different tools – IDEs, test frameworks, CI servers, Database Servers, Message Queues, etc
- Operations managing complex infrastructure – storage arrays, virtualisation, firewalls, clustering/failover/load balancing, etc
- Increasingly developers concerned with infrastructure and operations with development processes

DevOps

- DevOps stresses communications and collaboration
- Being able to see things in the round, big picture, etc
- See IBM article:
- http://www.ibm.com/developerworks/opensource/library/a-devops9/index.html
- Agile DevOps: Breaking Down Silos
- Discusses the idea of the cross-functional team – collaboration versus silos

DevOps

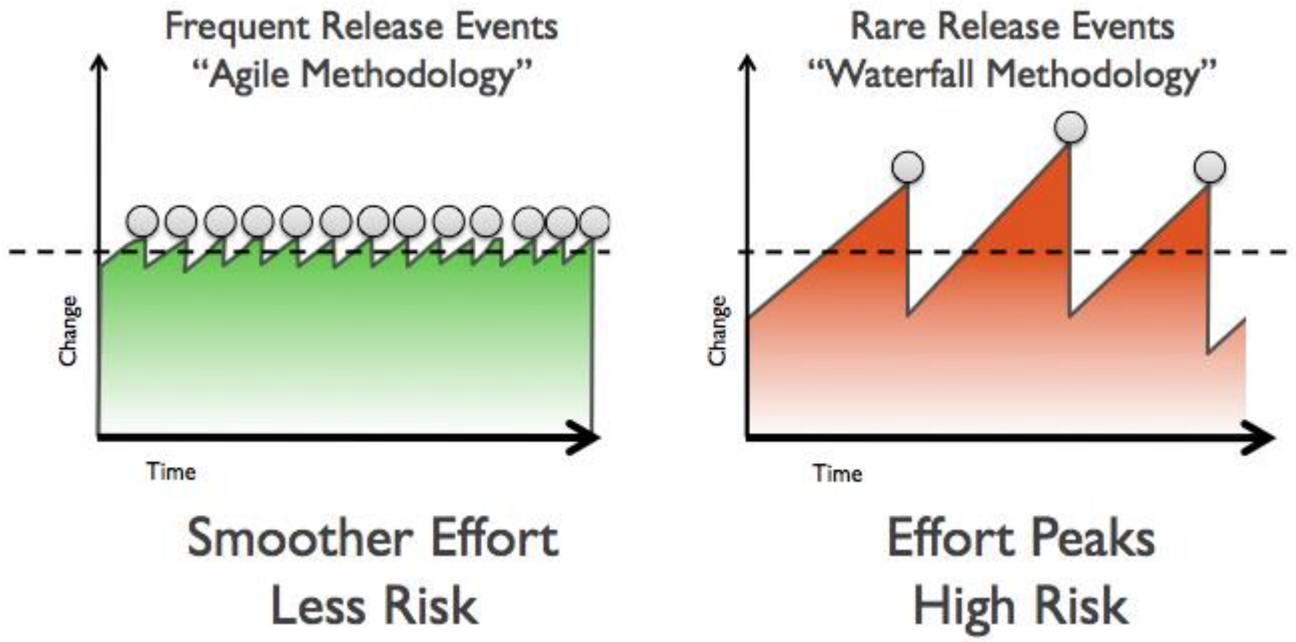
- IBM article suggests the following:
  - Dashboards with real-time information – e.g build status, availability, uptime
  - Colocation – physically break down those walls between the professionals
  - CI, automated deployment, scripted environments (e.g. autoprovision a virtual image), TDD (to include automated performance / load testing)
  - Polyskilled experts (not just jack of all trades, but expert in all trades!)

DevOps

- DevOps involves as much of a cultural shift as a technical shift
- You don't setup "DevOps teams"
- There would need to be a DevOps champion, perhaps someone who is not just a specialist in one area, but a generalist in many areas – can speak the lingo of development and ops – understand software development lifecycle and management of Cloud infrastructure
- Search jobs for DevOps Engineer or Director, for example

DevOps

- Smaller, but more frequent releases = less risk



DevOps

- DevOps champion tries to instil that culture
- Facilitates co-ordination (or ensures that someone is responsible for release co-ordination)
- A dedicated release co-ordinator role recognises that with agile processes, releases are smaller, but more frequent
- Scheduling them is important, as is developing the automation scripts for building and deployment – may need to "crack the whip" to ensure processes are being adhered to

Conclusion

- TDD is integral to CI, which is integral to CD, which is integral to DevOps
- Automation throughout is key
- DevOps is a cultural shift and needs to be championed
- It needs to filter down into possibly colocated teams of multi-skilled professionals
- People need to "get along" and "open up"!!!!