**Year 4**

2014

**IT Tallaght, Tallaght, Dublin 24**

08

**Fall**

Continuous Delivery ICS Adoption Dashboard

Shane Murphy

The purpose of the project was to develop a commit phase dashboard to monitor the adoption of a continuous delivery work flow in IBM’s ICS, with information fetched from RTC, RQM and Jenkins and stored locally. The users would be able to compare how each team progressed towards CD at a glance, aided by graphical representations of the data. Should more details be required, the ability to delve deeper into the teams to view builds and work items at a finer granularity is possible. Users were determined to be both team members and upper management.

Table of Contents

Background 2

Project 3

CD Transformation Checkpoints and Metrics 3

CD Transformation Metrics 4

Commit Phase 4

Acceptance Test Phase 4

Manual Test Phase 4

CD Business Value 4

Technological Research 5

Jenkins 5

RTC 5

Continuous Delivery Dashboards 5

Bower 8

Jasmine 8

QUnit 8

Webstorm IDE 8

MongoDb 8

DB2 8

LESS 8

COFFEE 8

Twitter Bootstrap 8

Groovy & Grails 9

Use Cases 9

View Summary Statistics 9

Order Results 9

Inspect Team Progress in finer Granularity 10

Historical Analysis 11

Dashboard Refresh 11

Team Progress vs Goals 12

Technical Architecture 12

Software Components 12

Groovy 12

Grails 12

DB2 13

Platform libraries 13

Distribution and Deployment 14

Architecture Diagram 15

Class Diagram 16

Overview 16

# Background

Continuous Delivery (CD) is a pattern language used in software development to automate and improve the process of software delivery. Techniques such as automated testing, continuous integration and continuous deployment allow software to be developed to a high standard and easily packaged and deployed to test environments, resulting in the ability to rapidly, reliably and repeatedly push out enhancements and bug fixes to customers at low risk and with minimal manual overhead.

IBM Collaboration Solutions (ICS), which consists of a large range of software development teams, comprising OnPrem, Cloud and Mobile, have made the strategic decision to adopt Continuous Delivery Practices for all software development teams. Each ICS Director is required to report progress on a set of CD Transformation Checkpoints and Metrics to Executives on a regular and scheduled basis.

# Project

Develop a Continuous Delivery ICS Adoption Dashboard, which reports out each CD Transformation and on a per software product development team basis. The data must be fed in from verifiable sources to ensure confidence in the status presented by the dashboard. It cannot be a tick box exercise.

For the purposes of the project, research can be carried out on the source and format of the data feed and how that data is presented to the dashboard. Sample data can be used, but the dashboard must be extensible to include additional metrics once known. The dashboard must be capable of being easily merged with any future common metrics repository and dashboard.

The main purpose of the dashboard is to provide information to team

members, management and executives on how they are doing against defined

goals.

# CD Transformation Checkpoints and Metrics

CD Transformation Checkpoints

Transformation Checkpoint

* Each code submission triggers the pipeline in serial
* Jenkins provides CI pipeline
* CI pipeline is fully automated
* Single source repository/trunk
* All artifacts kept in version control
* All Commit Phase failures stop the pipeline
* SPRs add data to track test escapes

CI pipeline is implemented

* Leverage common metrics repository and
* dashboard
* Push-button automated pre-submission validation
* All new code is code reviewed
* All new code has unit and functional test coverage
* Automated tests added for all resolved SPRs

Developer role and enablement

* SPRs created for all issues outside of the Commit Phase
* All test deployments use production deployment processes and are fully
* automated/repeatable
* All test deployments use production-congruent environments/stacks
* Full regression/compatibility testing of APIs and extensions
* Simian Army testing occurs continuously
* Dashboard metrics gathering is automated
* Continuous on-premises testing (NFR and on-prem only features)
* Zero downtime updates

Continuous delivery

* Additional SCSB usage/defect metrics stored in repository

## CD Transformation Metrics

### Commit Phase

|  |  |  |
| --- | --- | --- |
| Metric | Measurement Value | Anticipated Source |
| Build Duration | Minutes | RTC |
| Manual Testing Duration (Automatable) | Person Weeks | RQM |

### Acceptance Test Phase

|  |  |  |
| --- | --- | --- |
| Metric | Measurement Value | Anticipated Source |
| API Test Coverage (via automated measurement) | % Value | TBD |

### Manual Test Phase

|  |  |  |
| --- | --- | --- |
| Metric | Measurement Value | Anticipated Source |
| Manual Testing duration (non-automatable) | Person Weeks | RQM |
| Time From Final Code submission to Production (Cloud) | Days | DevOps / UrbanDeploy / CHEF / Jenkins |
| Time from final code submission to GA (On-Premises) | Days | DevOps / UrbanDeploy / CHEF / Jenkins |
| SPRs opened per capita (general quality measure) | Number | RTC |

### CD Business Value

|  |  |  |
| --- | --- | --- |
| Metric | Measurement Value | Anticipated Source |
| Cumulative Open & Deferred Defects (Technical Debt) | Number | RTC |

# Technological Research

While deciding on how I would implement the project, I spent some time researching and exploring various technologies before deciding on my final technology stack.

## Jenkins

* Jenkins plugins
* REST API resulting in XML/JSON/Python
* Internal Data model stored as a tree, API calls return a subtree, subtree cuts off at a certain point, this point can be adjusted.

## RTC

Setting up RTC for development (Schoon, 2013)

What API's are available? (Schoon, 2013)

RTC API?

* Javascript (limited)
* Java
* REST

Javadoc (IBM, n.d.)

* Planning
  + Documentation
  + Estimates
  + Widgets/Dashboard??
  + Change Requests
    - Delegation
  + Work Items
  + Build
    - Test
    - History
  + SCM
    - Role Based (privileges useable in dashboard?)
* Streams
  + Project areas
  + Team areas
* Components
  + User areas
  + Project areas
  + Team areas

## Continuous Delivery Dashboards

-Continuous Delivery in the Cloud – Part 6: Create a Dashboard of your System

(Codecentric, n.d.)

Python

Integration with Jenkins

* Query the test-results of a completed build
* Get objects representing the latest builds of a job
* Search for artifacts by simple criteria
* Block until jobs are complete
* Install artifacts to custom-specified directory structures
* username/password auth support for jenkins instances with auth turned on
* Ability to search for builds by subversion revision
* Ability to add/remove/query jenkins slaves

(Python, n.d.)

Ruby

Integration with Jenkins

* Creating jobs by sending xml file or by specifying params as options with more customization options including source control, notifications, etc.
* Building jobs (with params), stopping builds, querying details of recent builds, obtaining build params, etc.
* Listing jobs available in Jenkins with job name filter, job status filter.
* Adding/removing downstream projects.
* Chaining jobs i.e given a list of projects each project is added as a downstream project to the previous one.
* Obtaining progressive console output.
* Username/password based authentication.
* Command Line Interface with a lot of options provided in the libraries.--Creating, listing views.
* Adding jobs to views and removing jobs from views.
* Adding/removing jenkins slaves, querying details of slaves.
* Obtaining the tasks in build queue, and their age, cause, reason, ETA, ID, params and much more.
* Quiet down, cancel quiet down, safe restart, force restart, and wait till Jenkins becomes available after a restart.

(RubyGems, n.d.)

Knockout.js

Data-binding library

* Bind variables in the View Model (DOM)
* Observable Arrays
* Model View View Model (MVVM) Pattern
* shallower learning curve when compared to angular/backbone
* Becomes complicated when you want to do things that are not supported out of the box
* No IDE plugin

(Knockout, n.d.)

Angular.js

* Full Framework
* Useful for Data Driven applications, nice data binding
* JQuery or JQueryLite dependant
* Offers similar data binding to knockout, but also controllers and sets of services
* Dependancy injection is used quite a lot
  + Name function arguments with the same name
* Steep learning curve for more advanced features
* Popular
* Supports automatic testing out of the box
* IntelliJ plugin/VisualStudio plugin
  + Angular Batarang plugin for Google Chrome developer tools
* Large support community

(AngularJS, n.d.)

Backbone.js

* Useful for heavy DOM manipulation (With JQuery)
* Key-value binding and custom events
  + Not Data binding
  + Build own MVC solution using backbone objects
* Faster than Angular/Knockout (http://jsperf.com/angular-vs-knockout-vs-ember/171)
* A lot of boilerplate (code for wiring things together)
  + Free with Angular/Knockout
  + Code easier to read (less 'magic')
  + No surprises
* Need understanding of how Backbones set of objects work together
* Useful for specialised applications

(Backbone, n.d.)

Node.js

Networking

Push updates from the server to the client without having to poll or refresh?

Polling?

(Node, n.d.)

d3.js

Data visualization

(D3, n.d.)

Grunt.js

* Task runner
* Automate repetitive tasks
* Large JS projects
* Preprocessor files

(Grunt, n.d.)

Karma runner

* Test Runner
* Spawns a web server and executes source and test code for each browser connected
  + Results displayed via command line

(Karma , n.d.)

## Bower

* package manager
* Store a bower file, can pull down dependencies automatically.

(Bower, n.d.)

## Jasmine

* Test Framework for Javascript
* Behaviour Driven
* Clear syntax
* IBM Preference

(Jasmine, n.d.)

## QUnit

* IBM Prefer Jasmine, don’t use this

(QUnit, n.d.)

## Webstorm IDE

* Intellisense
* JetBrains
* Git integration
* Educational Licence available
  + Got License off Enda

(Jetbrains, n.d.)

## MongoDb

* Store Results of Jenkins queries for analysis of team improvements etc.?
* Document-orientated Database
* NoSQL
* MongoDb not approved for use in IBM, Brendan suggested DB2 instead

(MongoDb, n.d.)

## DB2

* Support for XML storage

(IBM, n.d.)

## LESS

CSS preprocessing

(LESS, n.d.)

## COFFEE

Javascript preprocessing

(Coffee, n.d.)

## Twitter Bootstrap

For anything not provided by IBM CSS

(Twitter, n.d.)

RQM  
(Jazz.net, n.d.)

## Groovy & Grails

* Compiles to Java bytecode on JVM
* Use of Closures, like C# or Java 8
* Similar to popular framework Ruby on Rails

(Grails, n.d.)

# Use Cases

|  |
| --- |
| View Summary Statistics |
|  |
| |  |  | | --- | --- | | **Primary Actor** | Product Team Member | | **Secondary Actor(s)** | Product team manager  Executive/Director | | **Level** | 1 | | **Story** | As a Product team member, I want to be able to view continuous delivery transformation assessments against agreed goals for my team. | |

|  |
| --- |
| Order Results |
|  |
| |  |  | | --- | --- | | **Primary Actor** | Product team manager  Executive/Director | | **Secondary Actor(s)** | Product Team Member | | **Level** | 2 | | **Story** | As a Manager/Director, I want to order results by General transformation statistics (column) to better allow me to see how teams are progressing with their CD adoption in relation to other teams. This would better facilitate the organisation of cross team knowledge sharing. | |

|  |
| --- |
| Inspect Team Progress in finer Granularity |
|  |
| |  |  | | --- | --- | | **Primary Actor** | Product team manager | | **Secondary Actor(s)** | Executive/Director | | **Level** |  | | **Story** | The ability to view the general statistics in finer detail to determine areas where the team could improve their performance in relation to CD adoption. The most relevant information should be readily visible from a high level, with the option to delve deeper and view additional information and more detail where available. Graphing, where appropriate, would make this information more useful. | |

|  |
| --- |
| Historical Analysis |
|  |
| |  |  | | --- | --- | | **Primary Actor** | Product team manager  Executive/Director | | **Secondary Actor(s)** | Product Team Member | | **Level** |  | | **Story** | As a Product Team Manager/Director, I want to view on a team-by-team basis, how the team is progressing with its adoption of continuous delivery. This would be represented best as graphs of the summary statistics over time. | |

|  |
| --- |
| Dashboard Refresh |
|  |
| |  |  | | --- | --- | | **Primary Actor** | Product Team Member | | **Secondary Actor(s)** | Product team manager  Executive/Director | | **Level** |  | | **Story** | As a general user, I want to see up to date feed information without having to periodically manually refresh the page. An automatic refresh time of ~60mins or a user definable time period would be increase usability. | |

|  |
| --- |
| Team Progress vs Goals |
|  |
| |  |  | | --- | --- | | **Primary Actor** | Product team member | | **Secondary Actor(s)** | Product team manager | | **Level** |  | | **Story** | As a product team member, I would like to be alerted to how our progress with CD adoption compares with our goals and expectations, highlighting areas where we are falling behind. | |

# Technical Architecture

## Software Components

### Groovy

The bulk of the application will be developed using the groovy language and will be built around the Grails framework. Groovy is dynamic statically typed language that runs on the JVM. Is designed to extend the Java language and make programming clearer and less cluttered.

There is a very shallow learning curve getting to grips with groovy when coming from a Java background, infact most Java classes can be run as a Groovy class, and Java can be written directly into a groovy class.

### Grails

Grails is a Framework built on Groovy. It has a Plugin based architecture making it very flexible and extensible. As a default, the following technology stack is used out of the box.

The default database is HSQL run in-memory. However, I will swap this out for a DB2 database to comply with IBM standards. This should be a simple process as Grails stack includes Hibernate out of the box.

#### Spring MVC

It implements Spring MVC application web framework (Though Grails is now capable of more than web developement). This framework seperates the presentation layer from the request handling layer, and the request handling layer from the model. By doing so, Spring MVC allows for cleaner front end code and easier testing and maintainablility.

#### Hibernate

Hibernate provides Object Relational Mapping for Java. Using GORM (Groovy Object Relational Mapping) and POGO (Plain Old Groovy Objects - like POJO for Java) this also works for Groovy. It provides a layer of abstraction from the database used in the applications implementation, allowing databases to be changed easily after deployment, as no database commands are hard coded into the source code. It does this by replacing direct persistence-related database accesses with high-level object handling functions.

#### Sitemaesh View Templates

Sitemesh is used as a framework for providing user views of an application to any HTML page requests using a gang of four decorator pattern. It allows for partial views to be created and embedded inside other views, allowing code reuse and quick building of pages.

#### Javascript

Because of the plugin architecture, Javascript libraries can be plugged directly into the project. This will be particularly helpful in implementing the AJAX calls required to refresh the page content after some elapsed time period. Some libraries that might prove useful in this project are Prototype, Scriptaculous, Compass, Lucene and Dojo. These requirements will become more apparent over time and will be explored in later iterations as development progresses.

## DB2

IBM's DB2 is a database which can handle multiple types of database schemas (Relational, object-relational, Native XML storage, etc.). It supports both SQL and XQuery query languages for database manipulation, allowing for faster access to the natively stored XML using XQuery. I will be using DB2 through the Hibernate abstraction layer provided in Grails, however it also have a number of native APIs for various programming languages.

## Platform libraries

Build Duration - from RTC

Manual testing Duration - Test Case Executions Records

API Test Coverage - TBD - We don't know how API Test Coverage will be

tracked. RQM would represent a tick box exercise

time from final code submission to production (cloud and on-premises) -

Jenkins will drive the commit but UrbanDeploy or CHEF will do the

deployment. How long it takes to execute the pipeline. Jenkins.

Defects opened per capita - RTC Query for Defects

Cumulative open+deferred defects - RTC Query

The data sources for the project will primarily come from the following sources:

* Build Logs
  + RTC has Build Duration time stored, this should be easy to access.
* DevOps
  + To be advised.
* Jenkins
  + REST API available which returns information in XML or JSON format. Again, easily and simply handled by groovy. Using the inbuilt HTTPBuilder class, messages can easily be formatted correctly to be sent accross the network. As an extension to this class there is also a RESTClient class which makes concessions in HTTPBuilders flexibility to make REST operations as simple as possible.
* RTC
  + REST API provided to interact with IBM Jazz based tools
* RQM
  + Still to be researched.

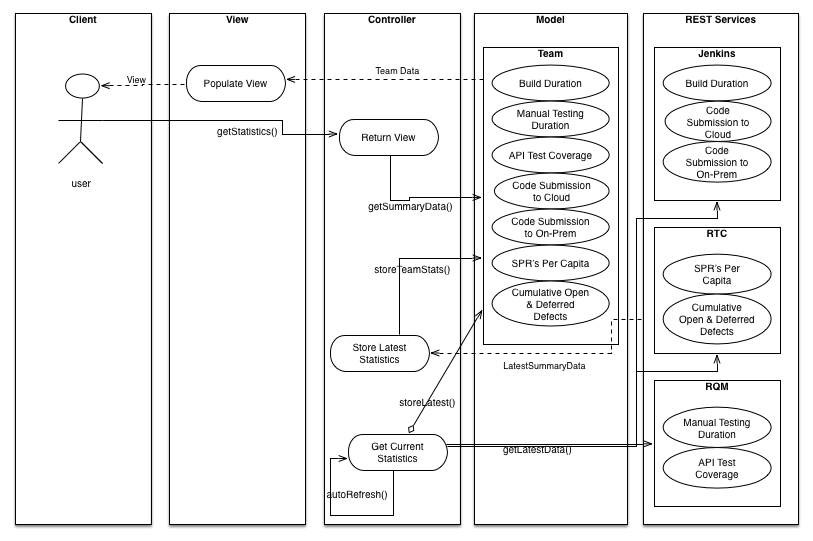
## Distribution and Deployment

* Can be deployed to a server as a WAR file.

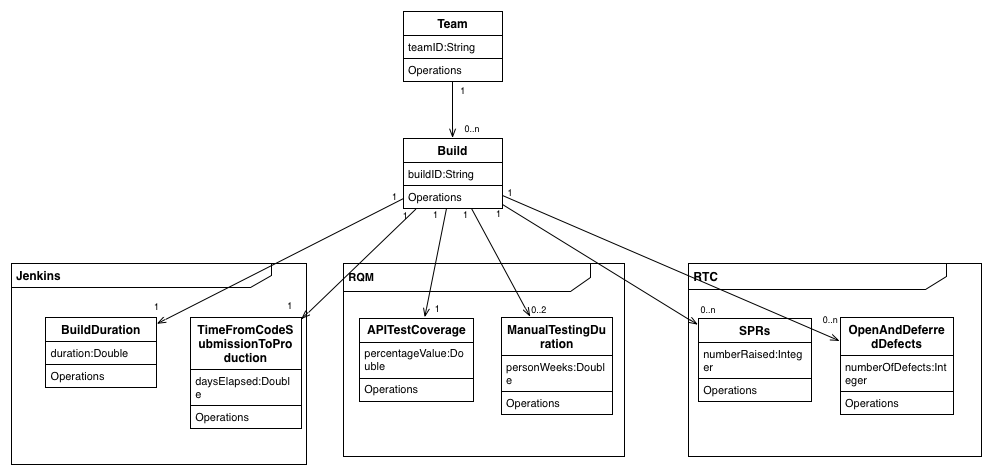
Risks

* Large scope of the project may need to be revised.
* Limited experience of javascript, which will likely be used to implement the graphs.
* A lot of firsts involved in the project
  + New Language (Groovy)
  + New Framework (Grails)
  + New Servers (RTC/DB2)

## Architecture Diagram



## Class Diagram



# Test Cases

|  |  |
| --- | --- |
| Test Case ID | START1 |
| Test Case Name | Purpose | Initiation Criteria | Expected Results |
| Startup With Empty Database | Test expected behavior is carried out when application is started | Database is empty but database area is created | Database is populated with Random Mock Data from Bootstrap, and all RTC Information is pulled in. |

|  |  |
| --- | --- |
| Test Case ID | START2 |
| Test Case Name | Purpose | Initiation Criteria | Expected Results |
| Start with existing data in database | Test expected behavior is carried out when application is started |  | No additional RTC or Bootstrap data is added to the database |

|  |  |
| --- | --- |
| Test Case ID | START3 |
| Test Case Name | Purpose | Initiation Criteria | Expected Results |
| Start with no connection to RTC Server | Tests that the application is stable even if RTC connection lost. |  | Exception caught and logged to error log. Application continues to run. |

|  |  |
| --- | --- |
| Test Case ID | START4 |
| Test Case Name | Purpose | Initiation Criteria | Expected Results |
| Fail if started with no Server Backend | The application requires a server backend in order to start. |  | Fail to start |

|  |  |
| --- | --- |
| Test Case ID | NETWORK1 |
| Test Case Name | Purpose | Initiation Criteria | Expected Results |
| No Internet Connection | Tests that the application can operate without a connection to the internet. |  | Application should function correctly, but graphs on the main page will not display. Google Chart API requires internet access. |

|  |  |
| --- | --- |
| Test Case ID | LOGIN1 |
| Test Case Name | Purpose | Initiation Criteria | Expected Results |
| Login | Tests the ability to login as default users admin and user with password = password |  | User logged in |

|  |  |
| --- | --- |
| Test Case ID | LOGOUT1 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Logout | Tests the ability to Log out from a user account | User is logged in | 1. Select Logout from the black bar at the top right of the screen | User logged in |

|  |  |
| --- | --- |
| Test Case ID | USERPROFILE1 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Add Subscriptions | Tests the ability to add subscribed teams to a users account | 1. User is logged in. | 1. Select User Profile from the black bar at the top right of the screen 2. Click ‘Edit’ 3. Select at least one Team from the checkboxes | Subscriptions added to account |

|  |  |
| --- | --- |
| Test Case ID | USERPROFILE2 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Remove Subscriptions | Tests the ability to remove subscribed teams from a users account | 1. User is logged in. 2. User has subscriptions associated with their account | 1. Select User Profile from the black bar at the top right of the screen 2. Click ‘Edit’ 3. Deselect at least one Team from the checkboxes | Subscriptions updated to reflect this change, team no longer visible. |

|  |  |
| --- | --- |
| Test Case ID | USERPROFILE3 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| View Profile | Tests the ability to view currently subscribed teams | 1. User is logged in. 2. User has subscriptions associated with their account | 1. Select User Profile from the black bar at the top right of the screen | Subscriptions shown as a list |

|  |  |
| --- | --- |
| Test Case ID | USERPROFILE4 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Can’t view other user’s profile | Tests that each users profile is secure from tampering by a 3rd party. | 1. User is logged in. 2. More than 1 user is registered | 1. Select User Profile from the black bar at the top right of the screen 2. Change the URL to the next User ID. | Changing the URL doesn’t load a new page. Only current users profile is loaded, regardless of the URL |

|  |  |
| --- | --- |
| Test Case ID | TEAMINDEX1 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Team Name Link | Tests the text of the Team Name links to the correct page |  | 1. Click the team name. In the Sample data this will be in the format “BootstrapTeamXXX” | Browser loads the Build Info page for that particular Team. If no build information is available, the page should load and inform the user there is no data to display. |

|  |  |
| --- | --- |
| Test Case ID | TEAMINDEX2 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Team ID Link | Tests the text of the Team ID links to the correct page | 1. There is at least 1 commit/ build for this project. Otherwise this link is not present. | 1. Click the team ID text at the top left of the box containing summary information | Browser loads the Build Info page for that particular Team. |

|  |  |
| --- | --- |
| Test Case ID | TEAMINDEX3 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Prompt to log in | User should be prompted to log in if not already | 1. User is not logged in. |  | Prompt for user to log in appears before summary data |

|  |  |
| --- | --- |
| Test Case ID | TEAMINDEX4 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Prompt Login Link | Tests the text of the login prompt works correctly | 1. User is not logged in | 1. Click the text ‘Log in’ in the prompt above the summary data | Log in page loaded. |

|  |  |
| --- | --- |
| Test Case ID | TEMPLATE1 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Refresh | Tests that the Refresh button causes a page refresh |  | 1. Click the Refresh Icon beside the application name in the top black bar | Browser reloads the current page |

|  |  |
| --- | --- |
| Test Case ID | TEMPLATE2 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Footer Home Link | Tests that the footer Home link functions correctly |  | 1. Click the ‘Home’ text in the page footer | IBM company homepage is loaded |

|  |  |
| --- | --- |
| Test Case ID | TEMPLATE3 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Footer Help Link | Tests that the footer Help link functions correctly |  | 1. Click the ‘Help text in the page footer | Help Page is loaded |

|  |  |
| --- | --- |
| Test Case ID | TEMPLATE4 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Footer About Link | Tests that the footer About link functions correctly |  | 1. Click the ‘About text in the page footer | About Page is loaded |

|  |  |
| --- | --- |
| Test Case ID | TEMPLATE5 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Footer Submit Feedback Link | Tests that the footer Submit Feedback link functions correctly |  | 1. Click the ‘Submit Feedback’ text in the page footer | Default Mail application loaded with new mail open to [feedback@ibm.com](mailto:feedback@ibm.com) and subject = CD Dashboard Feedback |

|  |  |
| --- | --- |
| Test Case ID | TEMPLATE5 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Help Icon | Tests that the Help Icon functions correctly |  | 1. Click the Help Icon in the top right corner of the black bar. It is a blue circle with a question mark in the middle. | Help Page is loaded |

|  |  |
| --- | --- |
| Test Case ID | TEAMINFO1 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Graph | Tests that the graph is the same as that displayed on the index page |  | 1. Note the shape of the graph on the index page. 2. Click the Team Name link | Graph should be the same values and shape on each page. |

|  |  |
| --- | --- |
| Test Case ID | TEAMINFO2 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Build Name Link | Tests that the build name link works as expected |  | 1. Click the Build Name link | Correct build info page is loaded. |

|  |  |
| --- | --- |
| Test Case ID | TEAMINFO3 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Team Member Email Link | Tests that the Team Member email link works as expected |  | 1. Click the Email address text | Default Mail application is loaded with the team member as recipient |

|  |  |
| --- | --- |
| Test Case ID | BUILDINFO1 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| WorkItem ID Link | Tests that the WorkItem ID link works as expected |  | 1. Click the WorkItem ID text | Correct WorkItem info page is loaded. |

|  |  |
| --- | --- |
| Test Case ID | BUILDINFO2 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| WorkItem Created date Link | Tests that the WorkItem Created date link works as expected |  | 1. Click the WorkItem Created Date text | Correct WorkItem info page is loaded. |

|  |  |
| --- | --- |
| Test Case ID | BUILDINFO3 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| WorkItem Type Link | Tests that the WorkItem Type link works as expected |  | 1. Click the WorkItem Type text | Correct WorkItem info page is loaded. |

|  |  |
| --- | --- |
| Test Case ID | CRONJOB1 |
| Test Case Name | Purpose | Initiation Criteria | Steps | Expected Results |
| Cron job | Tests that the Cron job to update the database executes |  | 1. Monitor logs, should see Cron execution | Cron job executed after defined time period |

# Test Results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Case ID** | **Test Name** | **Tester** | **Date** | **Result** |
| 1 | BUILDINFO3 | Startup With Empty Database | Shane Murphy | 29/04/2014 | Pass |
| 2 | BUILDINFO2 | Start with existing data in database | Shane Murphy | 29/04/2014 | Pass |
| 3 | BUILDINFO1 | Start with no connection to RTC Server | Shane Murphy | 29/04/2014 | Pass |
| 4 | TEAMINFO3 | Fail if started with no Server Backend | Shane Murphy | 29/04/2014 | Pass |
| 5 | TEAMINFO2 | No Internet Connection | Shane Murphy | 29/04/2014 | Pass |
| 6 | TEAMINFO1 | Login | Shane Murphy | 29/04/2014 | Pass |
| 7 | TEMPLATE5 | Logout | Shane Murphy | 29/04/2014 | Pass |
| 8 | TEMPLATE5 | Add Subscriptions | Shane Murphy | 29/04/2014 | Pass |
| 9 | TEMPLATE4 | Remove Subscriptions | Shane Murphy | 29/04/2014 | Pass |
| 10 | TEMPLATE3 | View Profile | Shane Murphy | 29/04/2014 | Pass |
| 11 | TEMPLATE2 | Can’t view other user’s profile | Shane Murphy | 29/04/2014 | Pass |
| 12 | TEMPLATE1 | Team Name Link | Shane Murphy | 29/04/2014 | Pass |
| 13 | TEAMINDEX4 | Team ID Link | Shane Murphy | 29/04/2014 | Pass |
| 14 | TEAMINDEX3 | Prompt to log in | Shane Murphy | 29/04/2014 | Pass |
| 15 | TEAMINDEX2 | Prompt Login Link | Shane Murphy | 29/04/2014 | Pass |
| 16 | TEAMINDEX1 | Refresh | Shane Murphy | 29/04/2014 | Pass |
| 17 | USERPROFILE4 | Footer Home Link | Shane Murphy | 29/04/2014 | Pass |
| 18 | USERPROFILE3 | Footer Help Link | Shane Murphy | 29/04/2014 | Pass |
| 19 | USERPROFILE2 | Footer About Link | Shane Murphy | 29/04/2014 | Pass |
| 20 | USERPROFILE1 | Footer Submit Feedback Link | Shane Murphy | 29/04/2014 | Pass |
| 21 | LOGOUT1 | Help Icon | Shane Murphy | 29/04/2014 | Pass |
| 22 | LOGIN1 | Graph | Shane Murphy | 29/04/2014 | Pass |
| 23 | NETWORK1 | Build Name Link | Shane Murphy | 29/04/2014 | Pass |
| 24 | START4 | Team Member Email Link | Shane Murphy | 29/04/2014 | Pass |
| 25 | START3 | WorkItem ID Link | Shane Murphy | 29/04/2014 | Pass |
| 26 | START2 | WorkItem Created date Link | Shane Murphy | 29/04/2014 | Pass |
| 27 | START1 | WorkItem Type Link | Shane Murphy | 29/04/2014 | Pass |
| 28 | CRONJOB1 | Cron Job | Shane Murphy | 29/04/2014 | Pass |

# Overview

The purpose of the project was to develop a commit phase dashboard to monitor the adoption of a continuous delivery work flow in IBM, with information fetched from RTC, RQM and Jenkins and stored locally. The users would be able to compare how each team progressed towards CD at a glance, aided by graphical representations of the data. Should more details be required, the ability to delve deeper into the teams to view builds and work items at a finer granularity is possible. The design was kept general, as users could be both team members and upper management. The ability to log in and customize which teams were of interest to you allows unnecessary information to be filtered out. The project was implemented in Groovy and Grails using an MVC architecture sitting above a services layer. I used a combination of d3.js and the Google Charts API to implement the graph elements, Spring Security to provide user logins and profiles and Quartz2 to execute cron jobs.