A Test Driving Approach For Beamline Software Development

Xiaoxu Ren, Paul Gibbons, and Bill Pulford
Diamond Light Source Ltd
Diamond House
Harwell Science and Innovation Campus
Didcot, Oxfordshire, OX11 0DE, UK
Email: {X.Ren, B.Pulford}@diamond.ac.uk

***Abstract*— For more effective and efficient development of bespoke software such as Generic Data Acquisition (GDA) for beamline control and data acquisition at Diamond Synchrotron, a set of Agile methodologies is adopted. In this paper, the test driving development approach used by the Data Acquisition and Scientific Computing Group is introduced. The challenges faced by the beamline software developers and the benefits of introducing TDD approach are summarised, followed by the software testing strategies for GDA in different software development level. The focus of this paper is on the higher level system test and user acceptance test. The simulation environment for the beamline software system level testing is introduced. The testing framework based on the *Fit/FitNesse* is presented. Both the GDA system test and acceptance test are conducted on this Web-based framework. Some test cases and their usage are discussed for demonstration.**

I. Introduction

Software testing is essential to ensure software quality. For bespoke software such as Generic Data Acquisition (GDA) [1] for beamline control and data acquisition at Diamond Light Source, a set of Agile methodologies are adopted for more effective and efficient software development. Among them, Test Driving Development (TDD) is one main part of the practice. Due to the rapidly changing beamline environment and constantly varying beamline commissioning demands, beamline software developers often must deal with requirements that tend to evolve quickly with short notice. They have to adapt the new changes into the existing software framework without the real beamline environment and deploy newly developed features with existing functionalities into real beamline in a race with tight beam time schedule. To meet the changing requirements, add more functionality within limited time and deploy the software in a beamline computational environment with minimum impact on the existing beamline operation are among big challenges for beamline software team. A TDD approach for beamline software development not only helps software developers to grasp the essential requirements correctly before the coding stage, but also assists them to make sure the newly developed features are what scientists want, without losing the existing functionalities or introducing new bugs. As a result, both the developers and the scientists as software customers will benefit from the testing framework with correct functionality, improved software quality and user confidence.

In this paper, the TDD approach used in the Diamond Data Acquisition and Scientific Computing Group (DASC) is introduced. The software testing strategies for GDA in different software development level are summarised. It focuses on the higher level system test and acceptance test. The implementation of the testing methods on these two levels within the *FitNesse* framework and the testing environment are addressed in details. This paper is organised as follows: Seciton II presents an overview of TDD approach for beamline control and data acquisition software. Testing strategies at different software development phase are outlined. The simulated computational environment of beamline is explained, which constructs a simulation environment for the beamline software system level test. In Section III, The testing framework used by DASC group is introduced. Both the GDA system testing and acceptance testing are developed on this framework. In Section IV, some test cases and testing results are presented for demonstration. Finally, some concluding remarks are made based on the experience.

II. Test Driven Development for Bameline Software

Most beamline software development follows the iterative development model (Fig. 1), which means that requirements from beamline users are not fully definable before coding can start. As a result, a working version of the software is built in a series of iterations that each iteration follows the requirements gathering, design, coding and testing. Some common problems associated to this kind of software development approach include:

- Difficult to test due to lack of formal documentation and constant changing requirements;
- Changes made by software developers are difficult to trace back to requirement/software code;
- The implementation of the changes can have unintended results on other part of software.

To counter these problems, test driven development is essential. The functional tests should be written first before coding and testing. Both software developers and the user representatives have to be involved in the testing and robust regression testing has to be performed.

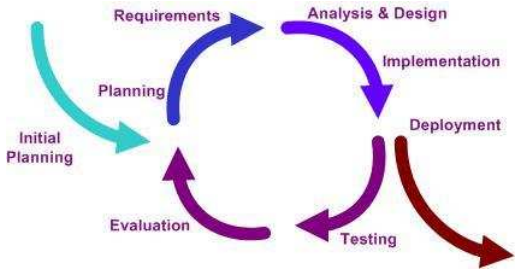

Fig. 1. An iterative development model [2]

Four levels of software test are practiced through the GDA development by the Diamond DASC group to insure the beamline software is being developed in the right way (verification) and once deployed in the beamline, the software will meet the user needs (validation):

- Unit testing
- Integration testing
- System testing
- Acceptance testing

The unit and integration level testing is of the DASC software developer's responsibility to ensure that GDA code written for the component meets its specification prior to its integration with other components, and works with other components together via interface interactions. The system level testing is for checking the software functionality from an end-to-end perspective. For GDA, because the real beamline is usually not available due to tightly scheduled beam time or new beamline hardware commissioning, the system testing is conducted in a simulated beamline environment.

The simulation environment includes:

- Simulated beamline computing environment
- EPICS simulation for beamline hardware control

The simulated beamline computing environment is build on the virtualisation concept, which essentially lets one computer do the job of multiple computers by sharing the resources of a single computer across multiple environments. For every Diamond beamline, there are virtual counterparts of its control servers, storage servers and workstations in the simulation environment. Each new GDA release is tested with its base configuration and all in-build dummy devices before the release is deployed into different beamline simulations, each with customised beamline configurations and following its own deployment rules. The system level testing conducted in the simulated beamline environment can reveal errors due to interactions across the whole beamline system, or those due to environmental issues.

By building a virtual beamline environment for GDA system testing, an objective assessment of the software can be made in a representative live environment. Both the functional and non-functional requirements defined in the functional specification can be verified without real operational beamline. Software developers can run the test at any stage of their development if necessary. Through the frequently conducted testing, potential problems arise from group development can be identified earlier and it helps the developers focus on the real beamline requirements.

An acceptance test is a black-box test involving a suite of tests on the software deployed in the working beamline environment. Each test, known as a case, targets a specific beamline operational scenario. Ideally, each test case should be associated with a formal description of the required beamline condition, the activity to conduct and the expected results. A passed case should provide a matched result. If there is a correct match for each test case, the software passes its user acceptance tests.

The purpose of beamline acceptance test is to provide both the DASC group and the beamline scientists with confidence that the beamline software will function according to their expectations. Referring to the iterative software development model, acceptance tests are carried out after the deployment, based on the requirement specifications as a basis for test. Beamline scientists as software users of the system perform these tests. At Diamond the basic test cases are derived before coding by the DASC group, based on the mutually agreed user requirement specification with beamline scientists. Beamline scientists can also create more specific test cases before or after development stage if needed. Once the acceptance tests are completed successfully, the beamline scientists will sign off on the deployed software release as tested to indicate its readiness for practice. This tested software can then be used for the next beamline run with newly enhanced functionalities. Any unsatisfactory test cases will be recorded, followed by discussion and evolution between beamline scientists and software developers to trace the causes and provide possible new solutions. All the failed tests form together new requirements in the next iterative development cycle.

III. Framework for System and Acceptance Testing

Automating of testing process at all levels is crucial for the TDD approach. For unit testing and integration testing, there are widely accepted solutions such as JUnit for Java or PyUnit for Python/Jython development, At Diamond, both JUnit and PyUnit are used extensively by the DASC group in GDA software development. For system testing and acceptance testing, the Fit/FitNesse framework [3] [4] has been chosen by the DASC group as the base testing platform. As a framework of creating, organising, and running Fit tests, FitNesse provides a set of flexible tools for software testing at both system and functional levels. Main reasons of choosing Fit/FitNesse include:

- Flexibility and extensibility. Custom fixtures can be easily added to assist the test, as illustrated in Fig. 2. For GDA and other beamline software in Diamond, a set of fixtures has been developed to facilitate the testing so that the coverage of the testing is satisfactory.
- Easy implementation. It is easy to setup a FitNesse test engine as a Web server and all the testing activities can be conducted over the Web interface. Developers and

beamline scientists can run single test or a set of tests as a test suite remotely via Web browser (3).

- Simplified test case generation. Fit uses tables to create test and report results. FitNesse provides a Wiki so that both software developers and beamline users can create and edit Wiki pages for test case generation and organising.
- Effortless testing handling. Running a test or a suite of tests in FitNesse is simple. Beamline users click the "Run" button to perform a single testing or the "Suite" button for a group of testing. For developers, it provides a command line interface to run the tests. There are also Maven plugins available for integrated development, so that a single command can compile the code, run the tests and create a test report. All of these features encourage both users and developers to run the test more frequently to improve the software quality.

Figure 2 illustrates the architecture of FitNesse. For GDA testing at Daimond, both the Fit *Column Fixture* and *Action Fixture* have been extended so that the main part of GDA logic can be covered. A *Command Line Action Fixture* is also developed for testing the execution of different GDA components, as well as some basic EPICS connectivity tests.

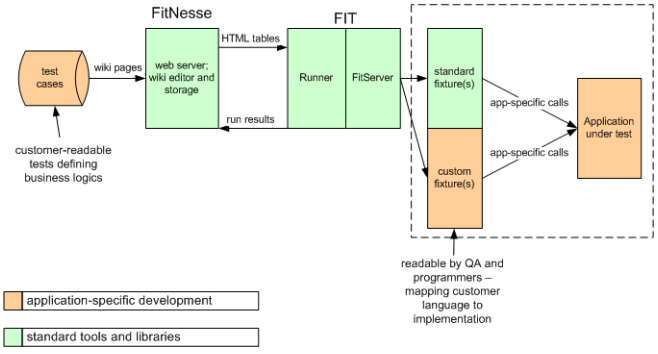

Fig. 2. FitNesse Testing Framework with Customised Fixtures [5]

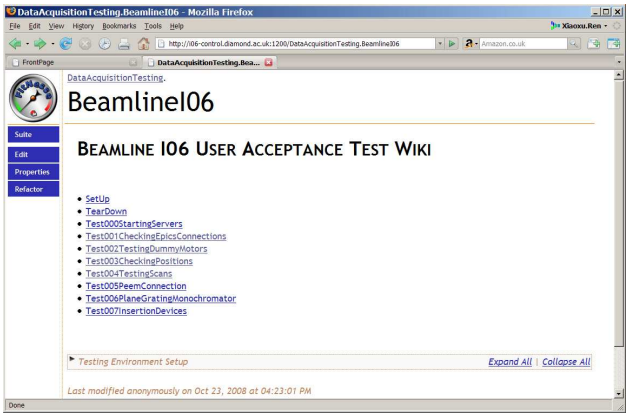

Fig. 3. FitNesse Test Engine behind Web Server with Wiki

IV. Implementation

The Fitnesse based testing platform is used by Diamond DASC group for both GDA system testing and acceptance testing. The basic test cases for GDA can be divided into three categories depending on the running environment:

- the generic GDA testing based on its base configuration
- the beamline configured GDA testing in a beamline simulation environment
- the beamline configured GDA testing running on real beamline

It is easier for the DASC group to perform the testing in the first two categories as part of the system testing effort both during and after the coding implementations. Extensive testing in this level gives the DASC group more confidence that the code refactoring or any inevitable changes in the code due to requirement changes have not broken any other code in the process. It also helps the developers to focus on the goal of each specific development task and let them know whether these tasks have been completed. The third category falls into the user acceptance testing level and can be conducted by the beamline scientists upon a real beamline setting. Testing in this category give the beamline scientists firsthand experience of the newly deployed software. Not only can they test the software functions, but also to learn and validate the software behaviours before visiting beamline users. Below are two simple examples of test case by using GDA test fixtures:

Check Diagnostic Motors:

uk.ac.diamond.dlsfitnesse.GDACommandColumnFixture							
deviceName	getValue?	deviceName	getValue?	deviceName	getValue?	deviceName	getValue?
d4x	-1.5<_<47.0	d5x	-10<_<48	d6y	-44<_<0	d8x	-5<_<45
d7x	-2.0<_<46.0	d7ax	-30<_<71	d9y	-46<_<21		

Check Counter Timers:

uk.ac.diamond.dlsfitnesse.GDACommandColumnFixture							
deviceName	getValue?	deviceName	getValue?	deviceName	getValue?	deviceName	getValue?
ca11	>0	ca12	>0	ca13	>0	ca14	>0
ca21	>0	ca22	>0	ca23	>0	ca24	>0
ca31	>0	ca32	>0	ca33	>0	ca34	>0
ca41	>0	ca42	>0	ca43	>0	ca44	>0

Fig. 4. Test Case 1, Using Column Fixture for GDA Testing

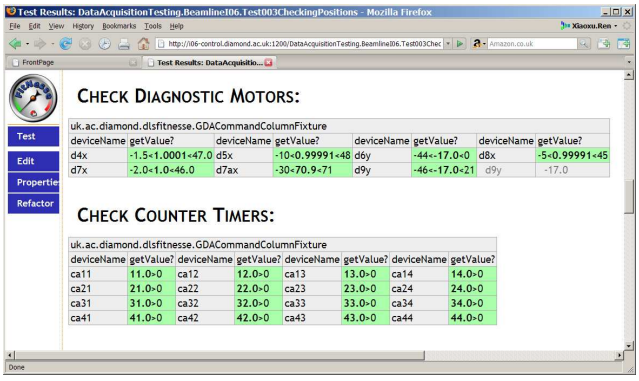

Fig. 5. Result of Test Case 1

1-D SCAN ON TWO DUMMY MOTORS

uk.ac.diamond.dlsfitnesse.GDACommandActionFixture	
whichFile	
noFileExists	
run	scan testMotor1 10 0 -1 testMotor2 10 1
pause	10
fileExists	
compareWith	http://files/samples/scan02.dat

2-D SCAN ON TWO DUMMY MOTORS

uk.ac.diamond.dlsfitnesse.GDACommandActionFixture	
whichFile	
noFileExists	
evaluate	scan testMotor1 0 10 1 testMotor2 1 2 0.5 None
pause	5
fileExists	
compareWith	http://files/samples/scan03.dat

Fig. 6. Test Case 2, Using Action Fixture for GDA Testing

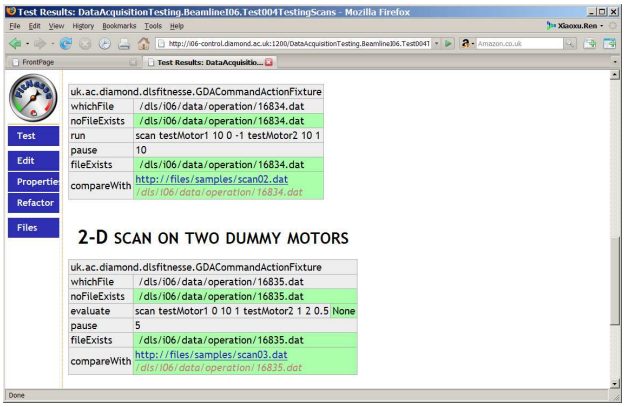

Fig. 7. Results of Test Case 2

Test Case 1: *"GDA should get correction position of seven beamline diagnostic motors and thirty-two counter timer channel readings."* Figure 4 shows the Fix Table representation of this test case by using the Column Fixture. The test result is illustrated in Figure 5, with colour highlighted test results.

Test Case 2: *"GDA should do one dimensional scans and two dimensional scans with results saved in the right place in correct format."* Figure 6 shows the Fix Table representation of this test case by using the Action Fixture, which is a sequence of actions. It first to check that there is no file name conflict before a scan, then run a GDA scan command with motors and check the existence of the file again before compare the file content with a pre-saved sample file. The test result is illustrated in Figure 7.

V. Concluding Remarks

Test driving development is crucial for the success of bespoken beamline software like GDA that follows the iterative development model. Setting up correct testing environment and an automated testing framework are critical for the collaboration among developers and between developers and beamline scientists. Conducting system level testing whenever possible is of great value for software developers to focus on what their software should do and get things right early in the development. Involving the beamline scientists in the acceptance testing and making it important part of the software development cycle provides significant value to increase the scientists' confidence in the beamline software.

Acknowledgment

All colleagues from Diamond Data Acquisition and Scientific Computing Group have their contributions to this teamwork.

References

[1] F. Yuan, R. Woolliscroft, and B. Pulford, "GDA an Integrated Science Experiment Environment for synchrotron users at Diamond and SRS," in *NOBUGS 2006*. Berkeley CA, USA: Lawrence Berkeley National Laboratory, October 2006.

[2] Wikipedia, "Iterative and incremental development — wikipedia, the free encyclopedia," 2008, [Online; accessed 22-October-2008]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Iterative_and_incremental_development&oldid=246912429

[3] R. Mugridge and W. Cunningham, *FIT for Developing Software: Framework for Integrated Tests*. Prentice Hall, 2005.

[4] FITNESSE, "FrontPage," 2008, [Online; accessed 22-October-2008]. [Online]. Available: http://fitnesse.org/

[5] FITNESSE, "FitNesse: OneMinuteDescription," 2008, [Online; accessed 22-October-2008]. [Online]. Available: http://fitnesse.org/FitNesse.OneMinuteDescription