AUT UNIVERSITY

Redefining the Use of Augmented Reality

Development Standards

Version 1.0
20 April 2015

Jason Gerbes	Joshua Son	Paul Lee	Sean Young
1274664	1388288	1264218	1302108

Contents

0.0 Version History

Version 1.0

Version 1.0 is the original version of the Development Standards document. This version of the document was created as part of the Quality Assurance Plan Version 1.0.

1.0 Introduction

Extreme Programming (XP) Development leverages a number of quality assurance methods to enforce good programming practices. Being an agile development practice, it is essential that quality assurance is taken into consideration at all phases of development — as to enforce an expected of deliverables (Balkanski, 2003).

2.0 Unit Tests

According to Osherove (2014), a unit test is an automated piece of code that invokes a unit of work in the system and then checks a single assumption about the behaviour of that unit of work. A unit of work is a single logical, functional use case in the system that can be invoked by some public interface (in most cases). A unit of work can span a single method, a whole class or multiple classes working together to achieve one single logical purpose that can be verified.

A good unit test:

- Is able to be fully automated
- Has full control over all the pieces running (use mocks or stubs to achieve this isolation when needed)
- Can be run in any order if part of many other tests
- Runs in memory (no DB or File access, for example)
- Consistently returns the same result (You always run the same test, so no random numbers, for example. save those for integration or range tests)
- Runs fast
- Tests a single logical concept in the system
- Is readable
- Is maintainable
- Is trustworthy (when you see its result, you don't need to debug the code just to be sure)

Under XP Development, all code must be bound by appropriate unit tests. All relevant tests must pass before the code can be release. If any bugs are found in the system, tests must be created to fix it. The unit tests are run often, and the score is published.

3.0 Test-First Development

XP utilises a test-first development practice. Unit tests are created first, before the code has been written. The code is then written with the intention of passing the unit tests. Creating a unit test helps a developer to really consider what needs to be done. Requirements are nailed down firmly by tests. Specifications cannot be misunderstood written in the form of executable code (Wells, 2000).

Unit tests give the developer immediate feedback as they work. It is often not clear when a developer has finished all the necessary functionality. Scope creep can occur as extensions and error conditions are considered. If we create our unit tests first then we know when we are done; the unit tests all run.

A test-first approach also benefits system design. It can be very difficult to unit test some software systems. These systems are typically built code first and testing second, often by a different team entirely. By creating tests first, the design will be influenced by a desire to test everything of value to the client. The design will reflect this by being easier to test.

4.0 Peer Programming

All code sent into production in XP development is created by two people working together at a single computer. Peer programming increases software quality without impacting time to deliver. Peer programming assures that the entire source code is reviewed all the time.

Wray (2010) explained peer programming using the metaphor of one programmer being the “driver” and the other the “navigator.” In this metaphor, the driver controls the keyboard and focuses on the immediate task of coding, and the navigator acts as a reviewer, observing and thinking about more strategic architectural issues.

5.0 Code Integration

Only one pair integrates code at a time. (because of parallel integration there is a combination of source code which may not have been tested together before. This is likely to lead to problems. Strictly sequential (or single threaded) integration by developers themselves is a simple solution to this problem. All new code is released to the source code repository by taking turns. That is, only one development pair integrates, tests and commits changes at any given moment. Single threaded integration allows a latest version to be consistently identified.)

Developers should be integrating and committing code into the repository every few hours, whenever possible. Continuous integration often avoids diverging or fragmented development efforts, where developers are not communicating with each other about what can be re-used, or what could be shared. Everyone needs to work with the latest version. Changes should not be made to obsolete code causing integration headaches.

6.0 Code Standards & Refactoring

All code must be written to agreed standards (camel casing etc). Committing to use agreed standards allows for consistent code that is easy for the entire team to read and refactor. A refactoring practice is agreed and adhered to, where duplicated code is removed, code integration is increased and the mixture of the code is reduced.

7.0 Collective Ownership

Collective ownership encourages everyone to contribute new ideas to all segments of the project. Any developer can change any line of code to add functionality, fix bugs, improve designs or refactor. No one person becomes a bottle neck for changes. To do this, developers are to create unit tests for their code as it is developed. All code that is released into the source code repository includes unit tests that run 100%. Code that is added, bugs as they are fixed and old functionality as it changed will be covered by automated testing.

8.0 Client Interaction

The client is always available during development for consultation. XP development utilises an iterative development practice where deliverables are produced during each iteration. These deliverables are communicated to the client, and the client can provide valuable feedback.

The client is accessible by all members of the team, and frequent communication can be expected. Regular meetings are planned, and communication via other means (e.g. email) will be used for questions and concerns mid-development.

9.0 References

Balkanski, P. (2003). *Quality Assurance in Extreme Programming*. Retrieved from http://www.foibg.com/ijita/vol10/ijita10-1-p17.pdf

Gray, C. (2010). *Quality Assurance and Assessment of Scholarly Research.* Retrieved 14 April 2015 from www.rin.ac.uk/quality-assurance

Osherove, R. (2014). *Unit Testing Lessons in Ruby, Java and .NET.* Retrieved 14 April 2015 from http://artofunittesting.com/definition-of-a-unit-test/

Wells, D. (2000). *Test First.* Retrieved 14 April 2015 from http://www.extremeprogramming.org/rules/testfirst.html

Wray, S. (2010). *How Pair Programming Really Works*. Software, IEEE , vol.27, no.1, pp.50,55, Jan.-Feb. 2010, doi: 10.1109/MS.2009.199