Report Three: Software Validation

Corina Ciobanu Iskander Orazbekov Mir Sahin
Paul Grigoras Radu Baltean-Lugojan

December 8, 2011

Abstract

proTrade is a professional tennis trading environment. As such testing plays an important role...

Contents

1 Introduction

At the moment there is a lack of good tennis trading environments. proTrade means to fill this gap, by providing an information rich application. Because of the risk associated with some user actions (e.g. placing bets with real money) reliability and stability are key factors for the application. As such testing and other validation methods play a particularly important role in delivering a quality product.

2 Testing

2.1 Preliminaries - Project Structur

2.2 Unit Testing

JUnit - wite functional tests using junit

2.3 Acceptance Testing

Separate tests that measure progress from tests that catch regression

new acceptance tests will not pass until the feature is implemented

acceptance tests for completed features catch regressions and should always pass (might take longer to run)

once an acceptance test has passed, if it fails again that means a regression (the existing code has been broken)

SWTBot

2.4 Regression Testing

2.5 Integration Testing

unit and integration tests should always pass (and run quickly)

2.6 Continuous Integration

2.7 Measuring Test Coverage

Cobertura We use cobertura for test coverage statistics.

Cobertura also generates cyclomatic complexity which measures the number of independent paths through the control flow graph of the code - 1 is minimum, the smaller ther better. [2]

2.8 Code Sanity Measures

2.9 Logging

2.10 Test Guided Design

Refactoring

Used extract superclass, pull up (for tests), extract class

2.11 At which development stages did you use tests

2.12 Which portions of your code base were tested in this manner

2.13 what bugs did testing reveal

3 Managerial Documentation

General Validation: to decribe any other methods you may have used to validate your executable deliverable; the following list is only suggestive but should give you an idea of what things you could do: Did you validate your user interface design? Did you use lint or similar tools? Did you conduct a manual code inspection (not by the person who wrote the code)?

not but i looked at the code Did you exert your software to stress testing? Did you test the GUIs of your software?

to give a formal account of group management and group activities: Collaboration tools used (eg svn)

git, github Management policies (eg code change/validation policies) Management of knowledge transfer within the group

dropbox, google docs A table of the group meetings - including dates, format and which members attended A table of the hours spent per week on which tasks or activities by each member on the project The Log-Book

Evaluation Criteria for Report Three: the same as for Reports One and Two.

[1] The book on testing [2] $http://en.wikipedia.org/wiki/Cyclomatic_complexity$

Rferences

The cyclomatic complexity of a section of source code is the count of the number of linearly independent pathsthrough the source code. For instance, if the source code contained no decision points such as IF statements or FOR loops, the complexity would be 1, since there is only a single path through the code. If the code had a single IF statement containing a single condition there would be two paths through the code, one path where the IF statement is evaluated as TRUE and one path where the IF statement is evaluated as FALSE. Mathematically, the cyclomatic complexity of a structured program[note 1] is defined with reference to the control flow graph of the program, a directed graph containing the basic blocks of the program, with an edge between two basic blocks if control may pass from the first to the second. The complexity M is then defined as:[2] M = E N + 2P where E = the number of edges of the graph N = the number of nodes of the graph P = the number of connected components

References

[1] Jonathan Rasmusson, *The Agile Samurai*, Pragmatic Bookshelf, October 2010.

[2] Jeff Langr and Tim Ottinger, *Agile in a Flash*, Pragmatic Bookshelf, January 2011.

[3] Robert Chatley's course on Software Engineering Methods *http://www.doc.ic.ac.uk/ rbc/302/*, Imperial College London

[4] Article on Software Development Risks, *http://fox.wikis.com/wc.dll?Wiki SoftwareDevelopmentRiskFactors*