CS 5500: Managing Software Development

SCG Project: Test-Plan

Team Name: PRX
Team Members: Liang Yu, Parvathy Unnikrishnan Nair, Reto Kleeb, Xinyi Wang

Continuous Testing

Experience shows that a separation of test and development (http://en.wikipedia.org/wiki/Software_testing#Finding_faults_early) leads to extremely high costs. This "over the fence to QA" mentality has to be prevented at all costs. This means that the first level of (unit) testing should happen while the code is written. To ensure that the tests fulfill the quality standards, they have to be reviewed with the same rules the apply to normal source code.

Procedures / Unit Test organisation

Each component of the system is tested in a dedicated Test-Class in the test directory. The package of the test class is the same as the one of the SUT (System Under Test). The test class is prefixed with the word Test.

The class tests each (reasonable) public method of the class with at least the following scenarios:

- Common Case
- Extreme Case
- Invalid / Unexpected Case

Central Repository

The development takes place in one central repository, this repository contains the complete sources and has no further external dependencies. Each developer can check out the whole repository at any point in time.

Continuous Integration

To guarantee that the code base and the related tests continue to remain stable and passing over the course of a project, a solution to continually integrate the source code is essential. Every single commit should automatically lead to the execution of the complete test set. If at any point any of these tests fail, the focus of the developers has to switch from adding features, to discovering and solving the issue

Used Utilities

The mainly used tool in this project is Junit, it allows a consistent evaluation of the state of a test-code base and is widely supported by other tools (IDE, build environments, etc) on the JVM platform.

Challenges for the SCG Tests

A fair amount of the SCG functionality is based on network based communication between separate processes. The required integration tests, to test the complete stack are more expensive to maintain and run.
The optimal compromise for this problem would be to isolate functionality into simple, testable units (with the support of Mock objects) and only execute the extensive integration tests at defined project mile stones.

Final Tests

The final tests are execute before every rollout of the SCG game.

Performance

The performance tests mostly affect the admin and its capability to host concurrent games. In this test, we measure the following aspects of the admin:

- Response time of the Admin Status Web interface
- Response time for new registrations

While these measurements are being taken, we steadily increase the number of games running on the admin (using multiple connecting clients)

User acceptance

The very technical nature of the SCG reduces the importance of classical user acceptance tests. But even with these conditions, test-implementers should be interviewed and observed while they use the provided tools. The informal feedback of these tests should be collected, evaluated and could lead to feature requests.

Reporting

Weekly (In class, integrated into general weekly status meeting)

Summarize weekly testing activities, issues, risks, bug counts, test case coverage.

After a Phase Completion

The test report includes:

- The total test cases, number executed, number passed / failed, numbers yet to be executed.
- The number of bugs found (date, solutions).
- The discussion of unresolved risks.
- The discussion of schedule of progress.
- The final report and sign-off

After the Final Test

To generate a final test report and wait for sign-off.

Each defect has to be reported in the chosen bug tracking system (presumably Trac). The bugs should be reported as clear and complete as possible. The architect of the SCG game has then to decide if the reported bug is actually considered a bug and then assign the task to the developer that is responsible / most familiar with the affected part of the system.

Bugs should be divided (according to the following list) by their severity and assigned a priority number for fixing.

Severity List

Severity ID	Severity	Severity Description
1	Critical	bugs which can cause non-recoverable condition, like system crashes, or database of file corruption, or potential data loss, program hangs.
2	High	bugs which can cause a lack of functionality, or insufficient or unclear error message. bugs which can prevents other areas of the app from being tested.
3	Medium	bugs which cause incorrect functionality of component or process.
4	Minor	Documentation errors or notation errors.

Priority List

Priority	Priority Level	Priority Description
5	Must fix	must be fixed immediately; the product cannot ship with this bug
4	Should fix	should be fixed as soon as possible or else it will ruin the company's reputation
3	Fix When Have Time	fix it when id does not delay the shipping date
2	Low Priority	fix it after all other higher

		levels are done
1	Trivial	fix it to make the product perfect