Institute of Computer Sciences
Research Group Software Engineering

Freie Universität Berlin

Improving the Quality of the Test Battery in Saros

Advisor: Franz Zieris

First Reviewer: Prof. Dr. Lutz Prechelt

Second Reviewer: Prof. Dr. Claudia Müller-Birn

Suzana Puscasu

Freie Universität Berlin

Contents

1. Introduction
2. Analysis of the Test Battery
3. Approaches to improve the quality of the test battery
4. Conditional testing – suitability for Saros
5. Conditional testing – implementation
6. Set-up Equivalence Classes –suitability for Saros
7. Set-up Equivalence Classes - implementation
8. Analysis of the implementations
9. Conclusion

Introduction

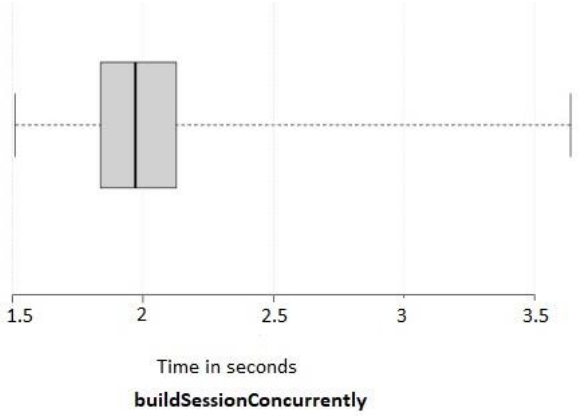
- Saros = Open Source IDE plugin for distributed collaborative programming
- **S**aros **T**est **F**ramework
  - easy writing of the tests for Saros
  - 5 testers : superBot and remoteBot
- Test Battery: 78 written tests

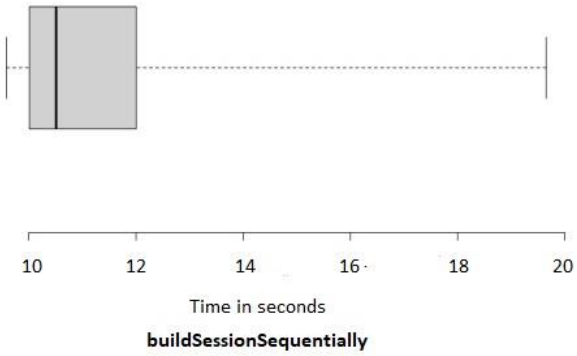
→Topic: Improve the quality of the test battery in Saros

Analysis of the Test Battery

1. **Execution time** – most used methods
2. **Similarities of the set-up and clean-up methods of the tests –** which tests need the same environment?
3. **Important test cases**

- “important“ = test cases, from which other test cases derive;





Approaches

Conditional Testing:

- identifying the main test cases, from which others derive;
- Running the derived tests according to the results of the main tests;
- Different levels of conditional testing.

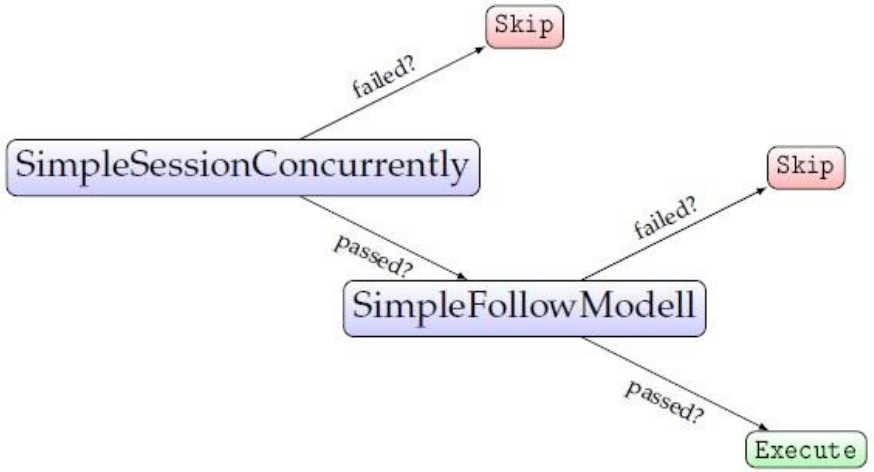
Set-Up Equivalence Classes:

- Set-up environment suitable for more tests;
- Organizing test cases into categories, according to the set-up methods;
- For each category, the set up is run only once.

Conditional Testing: Suitability

- Saros has different functionalities that depend on each other
- If tests for main functionialities fail, no point in running the depending tests
- Organized structure of the tests´ execution

Example:



Decision tree for FollowModeDisabledInNewSession

Conditional Testing: Implementation

Problems:

•No direct way for dynamical conditional testing in JUnit;

•Reason why a test was skipped → no overview of the tests´execution.

Solutions:

- TestWatcher as JUnit @Rule for each main test;
- Checking in depending tests, if the main test was successful;
- Method for all skipped tests that shows why the test was skipped.

Set-Up Equivalence Classes: Suitability

- A Saros session is built for each test;
- A Java project is needed for most of the tests;
- Four main set-up equivalence classes according to the number of testers;
- More possibilities to build up a Saros session;
- The Saros session can be built with an empty Java project.

Set-Up Equivalence Classes: Implementation

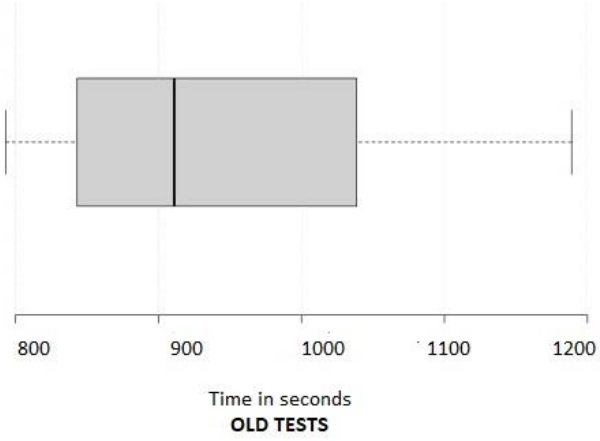
1. Identifying the set-up and clean-up methods, testing whether the set-up and clean-up methods work properly, and use this test as a condition for all other tests;
2. Defining the test categories and packing each category into a TestSuite:
   - *First test*: set up Saros session with an empty Java project;
   - *Last test*: tear down the Saros session, disconnect all users;
   - *Other tests*: no set-up and tear-down of the environment;
3. Adapting the clean-up methods for newly written set-up methods.

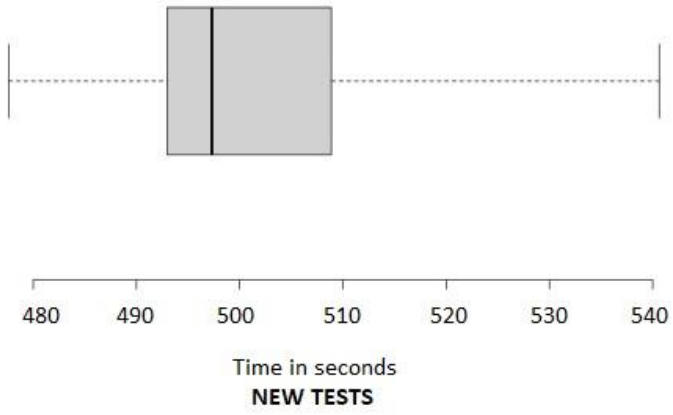
Disadvantages:

- Independence of the tests is affected;
- When writing new tests, one should be careful to the existing order .

Analysis of the Implementations

- Analyzed data: tests with 2 testers (47% of all tests);
- Number of executions: 20;
- Results: aprox. 45% less time.





Freie Universität Berlin

Conclusion

- Conditional testing – good practice;
- Set-up equivalence classes – good practice that saves up much time, but affects the indepence of tests.

Freie Universität Berlin

Thank you!