An Empirical Study of Writing Parameterized Unit Tests in an Open Source Project

Ben Hall  
Department of Computer Science  
North Carolina State University  
Raleigh, NC 27695-8206  
thall@ncsu.edu

Tom Fay  
Department of Computer Science  
North Carolina State University  
Raleigh, NC 27695-8206  
tfay@ncsu.edu

Abstract

Note that this sample paper skeleton serves as a guideline for helping produce a high-quality project report. If you think that you have a better or alternative way for presenting your project results (not fully following the guideline here), you shall contact the instructor.

In the abstract, you shall describe briefly parameterized unit tests, the name and characteristics of your open source project under test, and the findings of your empirical study.

1. Introduction

You shall describe with more details on the background of parameterized unit tests (PUTs) [3, 4] and Pex [2], how you can write parameterized unit tests, the name and characteristics of your open source project under test, and the findings of your empirical study.

You shall list the structure of the rest of the paper.

2. Example

You shall give an example parameterized unit test that you wrote (which is preferred to be from your open source project under test) and illustrate the procedure of writing parameterized unit tests with this example.

3. Open Source Project Under Test

You shall give details on what the project is about, and the characteristics of the project’s production code base (such as the number of classes, the number of methods, the number of public methods, and the lines of code), and the project’s test code base (such as the number of test classes, the number of test methods, and the lines of test code). Note that you may use some existing tools for producing these metrics; use your Google skills to find out these tools for you to use!

Normally you are expected to write new PUTs for the classes where you generalize conventional unit tests in the first milestone. If you intend to write PUTs for other classes, you need to consult the instructor and TA.

4. Benefits of Writing New Parameterized Unit Tests

You shall describe the number of new PUTs that you write beyond the PUTs that you generalized in the first milestone, and the percentage of the new PUTs among all the PUTs (including both new PUTs and PUTs generalized from conventional unit tests).

You shall list and compare the code coverage achieved by the PUTs generalized from conventional unit tests and all the PUTs (including both new PUTs and PUTs generalized from conventional unit tests). If you observe some new abnormal behaviors indicating potential faults (such as new uncaught exceptions or unexpected assertion violations), you can also describe them as benefits of the new PUTs. When you list code coverage, you can list details (test class by test class or test method by test method) or just list the aggregated statistics for all the test classes or test methods. How much detailed you want get into depends on the page limit and how you want to devote the limited space for showing the best (most interesting) results or findings from your course project.

5. Categorization of New Parameterized Unit Tests

You shall categorize your new PUTs into the test patterns proposed by de Halleux and Tillmann [1]. You shall list the statistics of your PUTs falling into each pattern.
You shall also propose new patterns to accommodate those new PUTs that you cannot categorize into any of the patterns proposed by de Halleux and Tillmann [1]. You shall describe the definition of these new test patterns and the example PUTs for these patterns. If you run out of space, you can refer the readers to the wiki entry links for the details of your new test patterns. Note that you shall at the same time prepare your wiki entries for your new test patterns no matter whether you describe your new patterns here in details.

You shall list the statistics of your PUTs falling into each of your new pattern being proposed by you.

6. Methodologies for Writing Parameterized Unit Tests

In lectures, we discussed some existing methodologies of writing parameterized unit tests such as (1) writing your own new conventional unit tests first and then generalize them to your new PUTs, (2) using a specific test pattern to guide the PUT writing, and (3) using input space partitioning to guide the PUT writing (i.e., finding out the needed combinations of blocks in characteristics, then writing assertions for each block combination, and finally combining those block combinations for sharing the same assertions).

You should list your experience of writing your new PUTs: what methodology do you use to write how many of these new PUTs? how do you feel about a specific methodology?

It would be good if you could propose or invent your own methodology for writing PUTs based on your experience.

Using some examples would be helpful for explaining your experience of using some existing methodology or your new methodology.

7. Helper Techniques for Parameterized Unit Tests

7.1. Factory Methods

You shall summarize the cases where you use factory methods to help Pex to generate better test inputs for your new PUTs. Again, it would be better if you can summarize patterns or categories for these cases.

If you find no such cases, you can state so.

7.2. Mock Objects

You shall summarize the cases where you use mock objects to deal with some complications that you face in writing new PUTs. You shall categorize these cases into the mock object patterns proposed by de Halleux and Tillmann [1]. If you cannot categorize them into these patterns, propose new patterns and document them in wiki similar to the preceding guidelines for documenting your new normal PUT patterns.

If you find no such cases, you can state so.

8. Limitations of Pex or PUTs

You shall summarize the limitations of Pex in terms of supporting test generation for your new PUTs. You shall also summarize the limitations of PUTs (e.g., some behavior cannot easily be expressed in PUTs and call for new types of tests or specification forms).

If you find no such cases, you can state so.

9. Conclusion

Your conclusion can be structured similar to the structure of your abstract. But do not just copy your abstract here.

References