Project: MUTMUT 🐢

(Latest Update: 2024-03-06)

Mutmut is one of the mutation-testing systems developed for Python, with a strong focus on ease of use. If you do not know what mutation testing is try starting with this article.

Project Repository

mutmut

Context for Strategic Refactoring

Let us suppose the hypothetical Company X wants to create their own version of mutmut. While they like the functionality of the current version of mutmut, they would like to have it ready for implemention of additional mutation operators (for example, the ones defined in this other tool) and parallel mutant executions. Thus, they have the following functional requirements they wish to implement.

1. **Design Patterns**
   The Chief Technical Officer of the company highly values well-designed projects. Thus, for their version of mutmut, the development team must alter the way the mutation operators are implemented so that they follow the Iterable and Strategy design patterns. For the rest of the code, they may implement other patterns from this list.

2. **Parallel Mutant Executions**
   The final item on the list of functional requirements for their version of mutmut is parallel mutant execution. This means that mutant operations must run in parallel to save time and energy.

Your assignment as a software reengineering team working for Company X is to restructure the current code of mutmut to support these requirements and show how your refactoring would support the implementation of the rest. This implies you must have a detailed refactoring plan.

Since Company X trusts its software engineers, it leaves the design of their version of mutmut up to you **but expect a detailed demonstration of the design decisions made to implement these requirements**. It would help if you considered that in your reengineering activities.

Instructions to get started

Please pay attention to the following instructions.

1. You need to send an email to Onur Kilincceker and Mutlu Beyazıt with the following.
2. Subject
   2.1. **"Reengineering 2024 - Mutmut"** if you intend to work on this suggested software; or **"Reengineering 2024 - Custom Project"** for a custom project proposal. Please try to contact us sooner than the deadline (March 22, 2024) if you want to work on a custom project.
3. Message Body

    3.1. The full name of the members in your group (including yours). Remember, a **maximum of 3** people, but you can work with less than 3 if you want.
    3.2. If you choose to work on a **custom project**, then you will need to **explain/motivate** why this project would allow you to demonstrate your reengineering skills.
    3.3. Attach the **pre-conditions report (PDF format)** to your message.
4. Your Pre-conditions Report should contain the following.
    4.1. Project Name (to be sure you are working on the suggested project or a custom one)
    4.2. Full Name of all the members in your group
    4.3. Link to your GitHub repository (which show to us you already forked the repository)
    4.4. The members are set as collaborators on the GitHub project.
    4.5. The teaching assistants are invited as collaborators (Onur's GitHub Id and Mutlu's GitHub Id).
    4.6. If you plan to automate the integration exercise, it is advisable to create a separate GitHub repository to host the automated patch integration tool. Please invite all assistants as collaborators.
    4.7. Demonstrate the ability to build the project. For this, we want a statement from the group stating they managed to successfully build the project. You can also attach a screenshot of your IDE with the project source and a message like "build successful".
    4.8. Simple Class Diagram of the target classes of the refactoring operation. A simple class diagram has only the name of the class and its interactions with the other classes. This aims to reinforce your initial understanding of the system. You need only focus on the target classes and those to which they are strongly coupled. There is no need to go deeper into the class structure (i.e., if Target Class calls Class X, and Class X calls Class Y, then you do not need to show Class Y since it is not being called directly by Target Class). We are not going to evaluate your strictness to the proper UML notations, therefore focus on modeling and understanding class interactions. For **custom projects, this will be a Simple Class Diagram of your system**. If your system is big, we can negotiate a smaller subset of the system for this diagram. In any case, you will need to contact the assistants sooner to set up the details.
    4.9. **[Optional]** A rough planning of the scope and goals. As you have noticed the refactoring project is open to interpretation (this is on purpose). Therefore, it is up to you, the students, to plan the scope of your project. It is entirely possible for distinct groups to have different scopes and planned activities based on this assignment.

General Coding Instructions

To work on this assignment, the following coding/repository instructions apply.

1. Fork the GitHub project and clone the source code for your team to work on it. In the documentation, you can find further instructions on how to build the software.
2. If you are working on this assignment as a group, then all members should be added to the repository as collaborators.
3. Add the teaching assistants as collaborators to your fork (See above for the Ids).
4. Commit/push your changes regularly providing information on the activity performed. Any "single" activity that requires file maintenance must be committed as a single commit with a simple description of the maintenance performed.
    4.1. For example, if you change the system to a refactor god class, your commit should be *"refactoring god class or fixing god class (Extract Class) or fixing god class (removing code clones)."*

4.2. Let us suppose you also introduced new tests along with the refactoring. *"Refactoring god class + new tests added."* Of course, pushing the tests on a separate commit would also be acceptable (actually, it would be better to do so).

4.3. It is not considered good practice to commit a substantial chunk of modified files without providing a reason that explains why those files have been modified. Therefore, try to split your commits into smaller units (that may help with the grading).

4.4. Your GitHub commit history will be evaluated. Therefore, be sure to commit and push regularly.

4.5. Be sure to commit/push the **final version** of your reengineering project **before the final deadline**. The final commit will be considered for evaluation as part of your assignment submission.

Development Activities

Then, more specifically, we ask you to perform the following activities and report about these in your project report.

1. **Design recovery**

Describe the current design implementation of the selected feature in the current Software. Clearly indicate how this design is implemented within the architecture of the project.

2. **Redesign**

Compose a generic design that describes how the new functionality / feature should be integrated and how the design handles the interaction with the rest of the system. It should be clear that the new design not only supports the new feature but also does not severely impact the code quality.

It will be necessary to redesign the test suite in such a way that it can cope with the new feature and design.

3. **Management**

Estimate the effort required for (i) refactoring towards the new requirements; and (ii) changing/extending the tests.

4. **Refactoring**

Refactor the current implementation of the Software such that it can handle the new feature. Adjust/extend the tests of the project to preserve their effectiveness and coverage during and after refactoring.

You will be required to perform several techniques presented during the lab sessions. These are:

- Analysis
- Metrics and Visualization

- Duplicated Code Analysis
- Software Repository Mining.
- Restructuring: Testing
- Refactoring.

Please use the techniques that you deem applicable to your problem. Within your report, you must convince us why you decided to use/exclude some of the techniques.

This project emphasizes the sound, systematic analysis of the presented problem, the associated solution space, and the chosen solution(s). The software reengineering sessions are composed this way to prepare you for such a project. We stimulate you to assess the benefits and drawbacks of the techniques presented in the lab sessions and ask you to exploit the analysis techniques wisely. You are free to use alternative analysis techniques and tools as much as you deem necessary.

Concerning the refactoring part, we emphasize the use of tests. Our minimum requirements are given below:

- Determine the extent to which the current tests provide feedback on your future refactoring-steps. **Quantify** this (i.e., show coverage information).
- Compose an argument discussing why the tests are adequate/inadequate for your chosen refactoring scenario. If inadequate, adjust the tests as needed. Be efficient with the time invested in testing.

General Evaluation

To show that you have passed the assignment, you will have to demonstrate the following.

1. You possess the knowledge to plan and select the appropriate reengineering patterns for your project activities.
2. You have selected analysis techniques (e.g., duplicated code analysis, mining software repositories, metrics and visualization as seen in the lab sessions, but others are allowed as well), and have applied these techniques in a sound, systematic manner. You have indicated clearly (using screenshots, results of the interpretation of the output of the techniques) how you have used the results of these analysis techniques.
3. You have performed the above activities (decomposed into (i) Design Recovery; (ii) Redesign; (iii) Management; and (iv) Refactoring) and discussed them in your project report.
4. The restructurings you have applied are behavior preserving.
5. You can demonstrate the mapping between each of the classes from the original structure with the new structure.
6. The compilation process succeeds flawlessly.
7. The tests run without flaws and show increased testing coverage making it more reliable.
8. The introduction of the new design clearly indicates the project is ready to be released in a language of choice. You are not supposed to carry out the refactoring process completely. Select and execute a set of refactoring tasks that sufficiently illustrate your proposed solution.
9. The report is written in a clear manner detailing all the steps and reasoning for the project. Remember that the report is the document that registers all your work. Thus, it is the most important artifact for the evaluation process.

For a more precise on-point view of the evaluation criteria, please look over the checklist (for each report) on the course's main page.

Report

Aspects that we typically like to see addressed in the final report are as follows:

1. **Context:**

   Briefly discuss the context in which you are running your project (do not just copy verbatim the text on 1. Contextualization).

2. **The Problem at Hand:**

   Clarify the problem at the base of the project, and indicate its intrinsic difficulties (again, do not just copy the assignment problem description, elaborate based on your chosen interpretation, goals, and scope).

3. **Reengineering Patterns:**

   You explicitly state the patterns (from the OORP book) that you selected and used throughout the project.

4. **Project Management:**

   Demonstrate how you have organized the work, and how you are controlling it (instead of the work controlling you!)

   4.1. **Scope:**

   What are the boundaries of your project? What is not included in the project?

   4.2. **Risks:**

   Which risks were envisioned, and which have been mitigated? What is the priority of the risks that still need to be mitigated? E.g., which external dependencies might affect your outcome? Which alternatives have you prepared in case this risk instantiates?

5. **Software Reengineering:**

   5.1. **Tests:**

   How can you verify that you satisfy the requirements? Which testing strategy have you selected, and what are the arguments for this selection? How confident are you that your solution satisfies the requirements?

   5.2. **Quality Assurance:**

   What are the non-functional requirements? E.g., how do you differentiate between a good and a bad solution?

   5.3. **Refactoring:**

   Which refactoring tasks did you perform on the project? Why is it better now? How does your refactoring help to support the new intended features?

These are aspects we like to see addressed/tackled/discussed/explained/presented in the **Final Report**. In the **Intermediate Report**, we expect less detail. However, groups that try to start addressing some of the above concerns most often have a better Intermediate Report. In the **Pre-conditions Report**, although not necessary, it might be better if your group starts to plan the scope and goals for the project.

Please also keep in mind and check the Report Guidelines and the Evaluation Checklist on the main page.

Final Remarks

If you have any questions about the project or the report, please contact the teaching assistants.

It is possible to submit your own project proposals based on Mutmut or another software system. These proposals will be approved if they provide a well-structured exercise on the reengineering techniques presented in the lab sessions.

To be on the safe side, use source code of the latest stable release.