Milestone 5: Robustness Testing

Released: Wednesday, Apr 10, 2024

NOTE: This milestone has two parts with separate deadlines.

Learning Objectives

- Evaluate a software component for robustness, scalability, and security based on an API specification and a running instance.
- Generate, communicate, and evaluate design options to improve the robustness of a service while considering trade-offs with other quality attributes.

Recommended Resources

- Design for Testability Lecture
- Design for Robustness Lectures
- https://www.geeksforgeeks.org/robustness-testing/
- https://www.geeksforgeeks.org/security-testing/
- https://www.geeksforgeeks.org/software-testing-scalability-testing/
- Fuzz Testing:
    - https://www.freecodecamp.org/news/web-security-fuzz-web-applications-using-ffuf/
    - https://www.fuzzingbook.org/html/WebFuzzer.html
    - https://medium.com/@cuncis/fuzzing-made-easy-how-to-use-wfuzz-for-efficient-web-application-testing-d843e5b089bf
    - https://github.com/marmelab/gremlins.js
- Bug Report Writing: https://www.softwaretestinghelp.com/how-to-write-good-bug-report/

Milestone Tasks

Part A: Testing

Due: **11:59 pm Wednesday, Apr 17, 2024**

In this milestone, your team will evaluate the robustness of the service developed by another team by trying to “break” it. Your team will be assigned the service to test in a separate message from the course staff.

Task 1: Designing Robustness Tests

Given the API specification of the service under test (SUT), identify faults that could break the SUT. These could be malformed requests, unexpected corner cases, malicious inputs, or creating an overload situation. Your goal is to identify multiple ways in which the SUT might break and describe potential ways for you to inject these faults into the system using the API provided to you.

Be **creative:** Think of many different ways to break the SUT. You can consider possible robustness issues, security attacks, or overload situations that expose scalability issues.

Try to think of **potential implementation shortcuts** the development team might have taken: Given the time constraints, the development team most likely identified certain inputs as not likely to happen and did not consider them in their design. If you would have been on their team, which inputs would you have ignored? These are potential candidates for you to test!

Also, to increase your likelihood of finding issues, we recommend you to think of ways to **increase the coverage** of your robustness tests (i.e, efficiently covering many possible inputs, see the ressources on fuzz testing above). How can you generate a large range of inputs that are likely to break the SUT? We do not have requirements for minimum coverage. This note is just to help you find issues more efficiently.

Task 2: Executing Robustness Tests

Execute the tests that you described above. These can be automated, semi-automated, or manual.

Please be courteous to the team that has built the SUT. Before running your tests, please let them know ahead by sending them a message. If you actually manage to crash the SUT, request a restart of the service and try to avoid running the same test again to avoid forcing restarts more often than necessary.

You have successfully found an issue if:

- the SUT crashes
- the SUT returns factually incorrect results or results that are different from the API specification
- the response takes much longer than expected or the SUT does not respond at all
- the SUT returns an error message even though the inputs conformed to the API specification
- you gained unauthorized access to data within the service, commands within the service, or the VM inside which the service runs (please do not attempt to extract real personal information, such SSH keys, credentials, or personal files, from the VMs of other students)

If possible, please try to identify **at least two** issues in the SUT. The issues can be of the same general type (i.e., service crashes on incorrect input), but should be meaningfully different from each other (e.g., instead of “failing on -1, -2, -3” is considered one the issue “failing on negative inputs” instead of three issues, but “failing on negative inputs” is considered a different issue from “failing for extremely large inputs”). You will receive **bonus points** for discovering issues that are from multiple categories (e.g., issues for robustness, security, and scalability).

Task 3: Reporting Results

Send a report containing your tests and results to the team that built the SUT. Provide them with steps to reproduce the issues that you identified (e.g., sequence of API requests or user inputs). If you used scripts to simulate requests, please also provide them with these scripts to make it easier for them to reproduce the results.

In the unlikely scenario that you were not able to find any issues with the SUT, send a report containing your tests only. In your report, provide an argument that your tests cover a diverse set of scenarios and inputs.

Part B: Design Improvement
Due: 11:59 pm Monday, **Apr 22**, 2024

Task 4: Design Improvement

For the issues that the other team found in your service:

- Describe at least two design options that improve the robustness of your design. If plain text does not clearly communicate the differences, model them using appropriate models that communicate the essential aspects of each design option.
- Identify possible trade-offs between robustness and other relevant quality attributes for each design option.
- Justify which option you would pick if you had to implement the improvement.

Given time constraints, you are **NOT** required to implement the robustness improvements that you come up with. But if you fix them, you will receive **bonus points**.

In the unlikely scenario where the other team has not found any issues in your system, please discuss possible ways in which you could further improve the robustness and/or security of your service by following the same approach described above.

Deliverables

Submit a report as a single PDF file to Gradescope that covers the following items in clearly labeled sections (ideally, each section should start on a new page). **Please correctly map the pages in the PDF to the corresponding sections.**

1. **Test Report (2 pg max):** Describe your test design, results that you found, and steps to reproduce the issues. Your test report should describe how you selected the inputs that you test, how you execute the tests, and how you assert that an issue is present. For creating bug reports and reproduction steps, please see these guidelines for writing effective bug reports: https://www.softwaretestinghelp.com/how-to-write-good-bug-report/. Please send this

report to the development of the SUT by Wednesday and attach a copy of this to your gradescope submission.

2. **Design Improvement (2 pg max)**: Describe and evaluate design options for two issues following the usual approach for discussing design decisions taught in this course.

Grading

This assignment is out of **100** points. For full points, we expect:

- **(50 pt)** A testing report that clearly describes your approach to identify potential robustness issues with the service of another team, presents your findings (including at least two found issues), and clearly describes steps to reproduce the issue. **5 bonus points** for issues from multiple categories found.
- **(40 pt)** A description of updated design that clearly describes the improvement for two issues that were identified, considers trade-offs between quality attributes, and discusses alternative design improvements. **5 bonus points** for implementing at least one improvement.
- **(10 pt)** A team contract that describes (i) the division of tasks among team members and (ii) a set of intermediate milestones, their target date, and expected goals.
- **(5 pt)** Bonus social points (same as the previous milestones).