Challenges and guidelines on designing test cases for test bots

Linda Erlenhov, Francisco G. de Oliveira Neto, Martin Chukaleski, Samer Daknache

Chalmers and the University of Gothenburg

Gothenburg, Sweden

linda.erlenhov@chalmers.se,francisco.gomes@cse.gu.se,{guschuma,gusdaksa}@student.gu.se

ABSTRACT

Test bots are automated testing tools that periodically run a set of test cases that check whether the system under test meets the requirements set forth by the customer. The automation decreases the amount of time a development team spends on testing. As development projects become larger, it is important to focus on improving the test bot's by designing more effective test cases because otherwise time and usage costs can increase reasonably and misleading conclusions from test results might be drawn, such as false positives in the test execution. However, literature currently lacks insights on how test case design affects the effectiveness of test bots. This paper uses a case study approach to investigate those effects by identifying challenges in designing tests for test bots. Our results include guidelines for test design schema for such bots that support practitioners in overcoming the challenges mentioned by participants during our study.

CCS CONCEPTS

• **Software and its engineering** → *Software maintenance tools*; *Software design engineering*.

KEYWORDS

test bots, devbots, case study

1 INTRODUCTION

Testing is an essential activity performed throughout software development and maintenance. However, increasing complexity of software-intensive systems, in addition to resource constraints, hinder test effectiveness. A combination of software automation tools and bots help decrease the time that development teams spend on software testing and support testers/developers to make smarter decisions related to testing activities [1]. Literature refers to such tools as *test bots*, and have been applied to leverage test coverage, test flakiness, and test planning [1]. Test bots are part of a wider

range of software bots, particularly DevBots, since they can be seen as an artificial software developer which is autonomous, adaptive, and has technical competence [2].

However, current literature lacks studies focusing on the challenges related to designing and applying test bots, such as the hindrances or utilities of test bots when designing test cases or planning test executions. Therefore, our goal is to investigate the current industry practices and challenges with software testing aided by test bots, particularly, test design practices. We perform a case study with an automotive company in Sweden where we (i) interview industry practitioners and (ii) analyse test artefacts to, respectively, identify their current practices and associated challenges, as well as to propose guidelines on how to design tests for their test bots. In short, our research questions are:

RQ1: What are the main challenges when designing and executing system tests on software bots?

RQ2: To what extent does the test design affect the effectiveness of the test bot?

Our contributions are: (i) a list of challenges reported by industry practitioners related to designing test cases for a test bot, and (ii) a guidelines with six items to support practitioners in designing test cases when using test bots. Particularly, factors such as execution time, cyclomatic complexity of test code and usage of synchronous/asynchronous programming affect the cost and maintenance of test bot in software testing. However, our findings are limited to the context of our case study, i.e., test bots used in system testing.

2 RELATED WORK

According to Lebeuf et al. [3] software bots can help improve the efficiency of every phase of the software development life cycle, including test coding. The paper outlines the different types of bots and how they can respectively help improve software development. While the paper is beneficial for outlining the difference between bots, it does not dive deeper into the different instances of bots, but rather provides an overview of how these bots can be beneficial. In turn, Erlenhov et al.'s [2] study proposes a taxonomy focused on DevBots while also providing definition and vision of future DevBots. Both taxonomies are relevant within our study as they provide insight on specific properties that helps distinguishing test bots from other software bot applications (e.g., chat bots). For instance, test bots in our study fall in the group of productivity bots, because they improve the development team's productivity by automating the execution of testing tasks [3], whereas other facets such as language or interaction [2] are not central at the current stage of our investigation.

Moreover, our contributions align with existing literature in test design practices, but targeting the specific aspects of test bots such as automation and autonomy. For instance, Tsai et al. [4] gives

Table 1: Case study planning according to guidelines by Runeson et al. [7]

Objective	Description
The context	Black box, end-to-end system testing
The case	One project from the automotive industry
Theory	Test case design, DevBots
Research questions	RQ1, RQ2
Methods	Direct and independent data collection
Selection strategy	Project using test bots for system testing
Analysis	Thematic analysis of interviews
	Qualitative assessment of artefacts

an overview of how to design tests, and depicts the creation of test scenario specification, test case generation and tool support. Authors illustrate the effort or time spent when creating end-to-end (E2E) system tests and the portion which is taken by the integration tests. Both types of test cases target verification of distinct parts and properties of a System Under Test (SUT) which, ultimately, impacts the test costs. Similarly, Mockus et al. [5] investigate how test coverage affects test effectiveness and the relationship between test effort and the level of test coverage, whereas Laventhal et al. [6] discuss in their paper the relevance of negative and positive tests and how testers show positive test bias, which can affect the quality of testing. Note that the common theme between our research and the aforementioned studies is the examination of test effectiveness along with test coverage, however, we aim to align/contrast those test design practices with the usage of test bots in industry.

3 RESEARCH METHODOLOGY

The summary of our case study planning is depicted in Table 1. Our case company is a relatively mature company that provides Services as a Product (SaaP) for different car manufacturing companies. Our research is going to be performed only within the scope of one project responsible for developing scalable software solutions for a specific car manufacturer.

The software under test uses Amazon Web Services (AWS) and Microsoft Azure as their cloud service providers. The developed software is deployed in a virtual private cloud (VPC) [8] in order to offer customer companies the benefits of a private cloud, such as a granular control over virtual networks and an isolated environment for sensitive workloads and service isolation. The organization develops their own in house system which is responsible for building the software artefacts that are constantly updated by the developers. It supports multiple languages like Java, Python, GO, among others.

We performed a semi-structure interview with selected participants following standard protocols for data collection where participants were asked for consent to use their data and all the collected data was anonymised. Participants were also given the opportunity to opt-out of the study at any time. The interviewees were selected based on having previous experience with the test bots used at the project. Four participants agreed to join our study, namely: one software architect, two senior and one junior software developer.

Our list of questions is available at: https://tinyurl.com/botse2020. The interviews were recorded, transcribed (upon consent from participants) and coded. We performed thematic analysis [9] on the interview data in order to find patterns in the raw data later
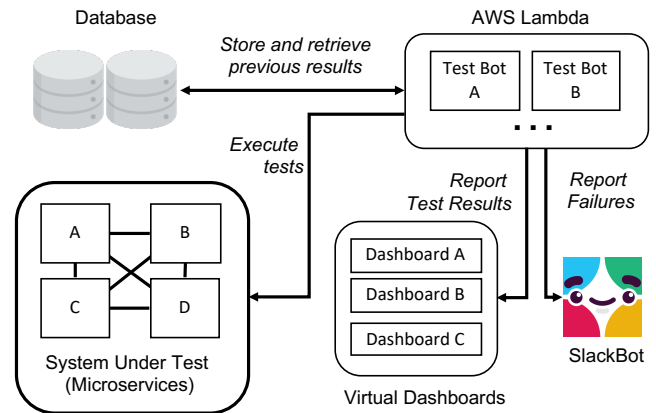
Figure 1: Overview of the application of test bots.

used as the base for the coding [10]. The outcome were categories which summarized the data gathered and expressed key themes and processes related to their usage of the test bots. Lastly, our findings were later presented to the participants of the interview to clarify and validate our understanding of their process.

Additionally, we also collected data from software artefacts which included test bot code, test case code and requirements for the SUT in order to investigate the design of the test bots and their test cases. Moreover, those two data sources (interviews and development artefacts) offer insights that enable us to answer, respectively, RQ1 and RQ2. The next section comprise our findings and discussion based on the interviews done with practitioners and the analysis of the artefacts related to the test bot. Based on our collected data, we begin by explaining how the test bots are used at our case company, followed by answer to our research questions and validity threats to our study.

4 FINDINGS AND DISCUSSION

The company creates various test bots for load testing, integration testing and system testing, however, for the scope of this study, we are going to focus only on test bots that are performing system testing. Figure 1 shows an overview of the test bots and the systems and tools it interacts with. The test bots used in the program have the task of performing end-to-end tests with a specific rate on different functionalities of the system. Depending on the testing context, whether the system that needs to be tested is back-end or front-end oriented, different programming languages will be used to write the test cases and the test bot.

Consequently, in order to design the test cases, practitioners are required to have knowledge on the corresponding programming languages and an overview of the architecture of the component, in order to be able to understand the flow between the different microservices. The regular workflow of the bot involves the execution of test cases, logging the test results and, in case of failures, include detailed error logs to make debugging easier. Lastly, the test bot also submits the results data to virtual dashboards shown to the entire team indicating the status of test activities. If the test fails, the test bot triggers a slack-bot to notify the corresponding team about the status and details of test execution.
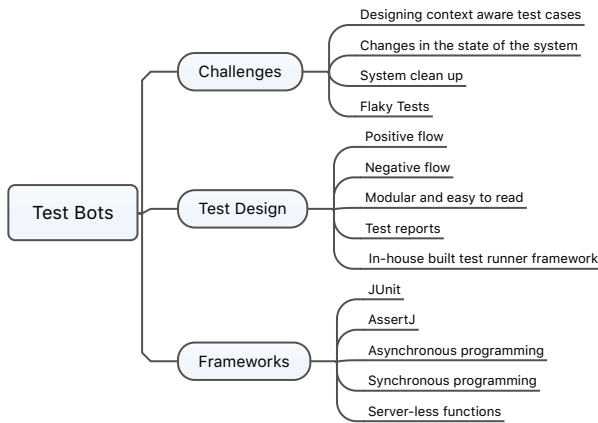
Figure 2: Themes and codes from the interview data.

4.1 Analysis of RQ1

Figure 2 shows the resulting themes and codes from our thematic analysis of the interviews. The data revealed different aspects about the test bots, pertaining their composition (e.g., test frameworks), usage (e.g., design of test cases for the test bots) and, in connection to our research question, the main challenges in applying them into the company's software testing activities.

C1. Designing context aware test cases: Tests can have dependencies to other tests meaning that a particular test t_a can only start executing once another test t_b has finished successfully. Performing this specific chaining on test cases can become quite complex and timely to achieve using typical test frameworks, such as JUnit. Issues can arise in the event that a test case fails, thus the following dependent tests will be affected by the previous success rate, hence being hard to determine whether the test failed because of some dependency or faults in the code. This dependency should be avoided when designing the test cases, such that modular tests are preferred to yield more independent test executions.

C2. Changes in the state of the system: Issues can occur when a test suite has only partially executed leaving, then, corrupt or invalid data within the system, such as incomplete data models which can later cause system errors or null pointer exceptions. To mitigate this problem, the test bot needs to perform roll back techniques in order to remove the invalid data that has been generated.

C3. System clean up: In scenarios when all of the test cases have completed successfully, the test bot needs to cover the case of cleaning up after themselves, since the test bots run on the deployed production environment. Consequently, there is a risk that the test bot activity could get mixed up with the customer activity, hence confusing developers while monitoring the application logs to debug a problem. For instance, consider a scenario where the test bot is sending 100 requests per second and flooding the system with the test associated logs, thus making it difficult to find other issues that could be user related in the logs. While challenge C2 addresses recovering from corrupt data of individual tests, the clean up for this challenge would be the equivalent of a tear-down of all tests. Automated test frameworks often support creation of tear down methods where the system state is restored after executing the test suite.

C4. Flaky tests: Flaky tests are false positives, i.e., test cases that fail when, in reality, there are no faults in the system and the functionality is working correctly. Consequenetly, flaky tests consume a lot of the tester's time and effort [11]. At the case company, the test bot evaluates a system which requires real time connections with different vehicles in order to collect their status, and often the tests fail due to a lacking connection as opposed to a fault itself. Therefore, when performing automated system testing, the stability of third party incorporated systems within the environment must be taken into consideration since they can affect the test results. One way to mitigate this problem is to mock the vehicle behaviour on a separate cloud server instance with very high uptime. Conversely, this can be costly and time-consuming to develop.

> **RQ1:** In summary, the following challenges were identified for system-level test bots are: (i) designing context aware test cases, (ii) monitoring and controlling changes in the state of the system, (iii) instrumenting system rollbacks and (iv) detecting flaky results.

4.2 Analysis of RQ2

Based on the artefacts analysis, we identified two different approaches to implement the test bots using, respectively, synchronous and asynchronous programming. Even though the different designs tested identical functionalities of the system, usage of asynchronous programming was more beneficial due to the following factors: (i) faster execution time, (ii) fewer false positive results and (iii) reduced cyclomatic complexity of the test code.

In terms of test execution time, asynchronous allowed different tests to run in parallel (when applicable) leading to faster test execution. Consequently, testing was cheaper for our industry partner due to the pay per use model of server-less applications where the test bots are hosted. Moreover, asynchronous approaches motivated testers remove dependencies between tests, hence mitigating issues with flaky tests. Finally, the artefact analysis also revealed that less complex tests (regardless of being synchronous/asynchronous) were easier to maintain by practitioners making it easier to also maintain the test bot.

Therefore, asynchronous programming yields better test cases for a test bot, however, such approaches limit the implementation since they are coupled with the instrumentation offered by testing frameworks. In order to overcome this limitation, practitioners created a simple test runner class which would handle/monitor all the test cases, together with the creation of the test reports.

> **RQ2:** Design choices affect execution time, flakiness and complexity of test cases executed by the test bot. Particularly, using asynchronous programming benefits the bot since it enables (i) parallel and faster execution of tests, and (ii) diligence in designing context-aware and independent test cases.

4.3 Guidelines for Designing Tests for Test Bots

Using the data collected from interviews and artefacts, we created guidelines (Table 2) that should mitigate the challenges discussed in the previous section. The guidelines foster good system test

Table 2: Guidelines for creating good system test case design

ID	Description	Reason behind guideline	Suggestion on how to implement it
G1	Use asynchronous programming methods to invoke system endpoints.	According to the data analyzed, asynchronous programming methods allows the tests to continue testing independent functions simultaneously and can thus reduce execution time.	With the Java framework CompletableFuture test can be executed with the method. supplyAsync(), in this case the framework will run the task asynchronously and return the result from the test without blocking the execution of other test.
G2	Cover test dependencies by chaining dependant test via the specific callback asynchronous functions.	For dependant tests, tests which require the completion status of previous tests, this would allow the system to wait before executing the next tests, while for independent tests, these can be unchained thus allowing all the non dependant test to run within their order.	With the Java framework CompletableFuture test *A* can be executed with the method supplyAsync(), and test *B* needs to be chained to the first future using the function thenComposeAsync(), this way test *B* will execute once *A* has finished without blocking other test executions in the test suite.
G3	Create test that are small, modular and readable.	By creating smaller and more readable tests, developers can ensure that future alterations to the tests are easier to implement, as well as less time will be needed to analyze the existing setup and develop the new test.	Create test cases that test a specific functionality of the system rather than multiple flows.
G4	Start clean-up process after the execution of tests.	In order to avoid adding corrupt or unnecessary test data to real environments, developers should include a clean-up process after the execution of tests within the test bot.	Use existing (or implement) functionality to remove data which was stored in the system by the test suite (e.g., after testing the user creation functionality, use the delete user service to remove the test data).
G5	Make use of proper logging techniques which differentiate the test data from real data.	In order to make it easier to distinguish the *test activity logs* from the actual *user activity logs*.	Use different prefix for test data attributes, thus making it easier for developers to distinguish test data from real data in real environments (e.g., username starts with "TEST").
G6	Implement both positive and negative flow testing techniques.	According to Laventhal et al. [6], software testing theory suggests that tests should test inside and outside specification (expected versus unexpected values) in order to test thoroughly.	Use both invalid and valid test data as input.

cases for test bots similar to the ones used at our industry partner. We also provide a reason for including each guideline along with suggestions on how to implement it.

Additionally, our findings also relate to existing literature on guidelines to design test cases, such as creating reusable and readable tests (G3) [4] or targeting both positive and negative flows for tests (G6) [6]. In contrast, literature does not consider unique aspects of test bots, such as autonomy or interaction with users, when discussing test design practices to an extent where one can argue whether current design practices are relevant to a DevBot.

The discussion around challenges also triggered participants to share what kind of future improvements they would like to add to the test bot functionality. An example is to predict the time needed for the system to perform each functionality in the test suite. Later, this information can be combined with the history of test results to reveal patterns able to describe how usage spikes can affect system performance. This data is valuable and can be used to extensively configure the application for improved performance.

4.4 Threats to Validity

In turn, *construct validity* is associated to our choice of artefacts, participants, themes and corresponding codes. One of the main risks is that literature lacks consolidated constructs or definition of a bot hindering the proper selection of bots for a study [2]. We mitigate this threat by using existing taxonomies [2, 3] to identify whether the investigated test bots have properties based on the facets of those taxonomies.

Our *internal validity* is related to the interview process and the analysis of artefacts, such as a obtaining limited answers from participants, or overlooking relevant aspects of their answers. In order to mitigate risks, the interviews were recorded and transcribed. Moreover, we used a semi-structured format such that participants were allowed to slightly stray from the questions in order to convey their own understanding of the process that the questions may fail

to capture. Moreover, the review of artefacts and interpretation of the test bot process described in the interviews was later validated with practitioners at the case company.

In turn, our *external validity* is limited, since our study comprises a case study and conclusions and findings are connected to the case company's context. Future studies investigating similar aspects of test bots applied to domains beyond automotive industry can confirm/contrast our findings. Moreover, clearer definitions of a bot and their role in software development can support generalization in future studies and enable researchers to find the commonalities between the applicability of bots across different domains of software development.

5 CONCLUSION

Our case study revealed a set of challenges faced when designing test cases for system tests executed by a test bot. We obtained these challenges through interview with industry practitioners and analysis of software artefacts (test bot code and corresponding test cases). Particularly, the main challenges identified are: designing context aware tests, monitoring and controlling system states, performing system clean-ups and identifying flaky tests. Moreover, our qualitative analysis of the collected data yields a list of guidelines that support practitioners in designing test cases for test bots similar to ones used in our context where bots are hosted in a cloud infrastructure communicating with vehicles. For instance, when comparing test cases designed using, respectively, synchronous or asynchronous approaches, our results indicate that usage of asynchronous programming improves the effectiveness of a test bot since (i) it reduces time to execute tests and the overall testing costs, and (ii) brings awareness to issues with dependencies between tests, hence mitigating risks with flaky tests. Future studies include the investigation of other types of bots used by our industry partner, as well as investigating the impact of those design practices in other types of test bots focusing, e.g., integration and performance testing.

REFERENCES

[1] M.-A. Storey and A. Zagalsky, "Disrupting developer productivity one bot at a time," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 928–931. [Online]. Available: https://doi.org/10.1145/2950290.2983989

[2] L. Erlenhov, F. G. de Oliveira Neto, R. Scandariato, and P. Leitner, "Current and future bots in software development," in *First Workshop on Bots in Software Engineering, (BotSE ICSE)*, 2019.

[3] C. Lebeuf, M.-A. Storey, and A. Zagalsky, "Software bots," *IEEE Software*, vol. 35, no. 1, p. 18–23, 2018.

[4] W.-T. Tsai, X. Bai, R. Paul, W. Shao, and V. Agarwal, "End-to-end integration testing design," in *25th Annual International Computer Software and Applications Conference. COMPSAC 2001*. IEEE, 2001, pp. 166–171.

[5] A. Mockus, N. Nagappan, and T. T. Dinh-Trong, "Test coverage and post-verification defects: A multiple case study," *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009.

[6] L. M. Leventhal, B. E. Teasley, and D. S. Rohlman, "Analyses of factors related to positive test bias in software testing," *International Journal of Human-Computer Studies*, vol. 41, no. 5, pp. 717–749, 1994.

[7] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, p. 131, 2009.

[8] N. Serrano, G. Gallardo, and J. Hernantes, "Infrastructure as a service and cloud technologies," *IEEE Software*, vol. 32, no. 2, pp. 30–36, Mar 2015.

[9] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, p. 77–101, 2006.

[10] M. Maguire and B. Delahunt, "Doing a thematic analysis: A practical, step-by-step guide for learning and teaching scholars." *AISHE-J: The All Ireland Journal of Teaching and Learning in Higher Education*, vol. 9, no. 3, 2017.

[11] D. Firesmith, "Common testing problems: Pitfalls to prevent and mitigate," in *A1AA Case Conference*, 2013.