Automated Test Coverage of User Scenarios Analyzing System

Ekaterina Ushkova
UrFU
Ekaterinburg, Russia
ushkova@skbkontur.ru

Abstract

The article describes a system that allows you to automatically measure the coverage of an existing program code by autotests based on the analysis of user scenarios. The pros and cons of this system are considered, as well as different approaches to the analysis of test coverage are compared.

1 Introduction

Testing is an essential part of the application's quality control process. There are several kinds and types of testing (for example, manual and automated, functional and non-functional, white and black box testing, etc.).

Testing allows you to minimize the risks and losses from defects in the program that may occur during its operation. The sooner a defect is found, the cheaper it will be for the development team and for the whole company to correct it. [4]

Regression testing is needed to ensure that the old main functionality of the product is not broken by the new changes.

At the release an update time, we need to be sure that not only the change itself has been tested, but the rest of the system that could have been affected by this change, or the regress, has been tested too. If the new functionality in the update can be checked manually by testers, the regress is checked by the robot – autotests.

But how to make sure that autotests are checking enough? How to evaluate the test coverage of the entire system? Can we talk about the overall quality of the product, not being able to assess the degree of coverage of real users scenarios?

It was decided to find a way to accurately and correctly measure the test coverage, which imitates the user's actions in the system (UI-tests, user interface tests), the actual actions performed by users in the system.

The relevance of the topic is due to the need for software development companies to reduce labor and time resources for analyzing the completeness of autotest coverage of user scenarios.

According to the testing pyramid, UI tests are the most expensive to write and maintain, so they have to check only what cannot be verified by unit tests or functional tests. [1] For example, UI tests often cover complex scenarios where integration with different services is involved.

Test coverage for different levels of tests is measured differently, and the lower the test level, the easier it is to measure. The most difficult to measure test coverage are UI-tests.

One way to measure it is to measure the coverage manually. If the service is small, and all its functionality can be described using an intellectual map (mind-map), it is easy to check which scenarios are covered by autotests and which are not using this map. If the application is large enough, old, integrated with several services and its functionality is difficult to describe completely, then it becomes very difficult to measure test coverage manually. [2]

Therefore, it was decided to develop an automated test coverage analyzing system. This system was developed taking into account the features of the application under test, namely, a microservice web application with a large share of legacy code (old code that is difficult to maintain and almost impossible to cover with unit tests).

2 System Requirements

Usually, a UI test script looks like some complete user action — sending a report, responding to a request, adding a new user, etc. A UI test usually simulates several real user actions — navigating through pages, filling fields, pressing buttons, and checking the application's response — whether all actions in the system led to the expected result. [3]

At the stage of developing the concept of coverage measurement, several questions arise:

1. What is considered an important (main) user scenario?
2. How to determine the coverage of the script auto tests?
3. How to automate the detection of such scenarios and their verification of coverage?

Coverage measurement should use a tool that meets certain requirements:

- measurability of the coverage;
- objectivity of coverage;
- ease of use;
- automated process for obtaining coverage;
- the results should help find directions for improving and improving the automation process;
- minimum costs for further support of the system;
- scalability.

The result of the system work should be simple and clear to any person, for its interpretation it should not be required to dive into the context. The program should give the following information:

- coverage percentage;
- covered scenarios;
- uncovered scenarios;
- scenarios that need to be covered with priority.

3 Approaches to the assessment of autotest coverage

3.1 Requirements coverage

To implement this method, detailed requirements for all new, as well as existing functionality should be described. Auto tests are written for these requirements.

Pros:

- coverage estimation algorithm – the requirements pool ideally should coincide with the test pool.

Minuses:

- writing and supporting detailed analytics (especially regression) will require a lot of manpower, in addition, a new problem will arise – you need to somehow assess the functionality coverage of analytics.

3.2 Code coverage

There are many ready technical means for measure the coverage of a code coverage with tests. Basically, this method is used to measure the level of unit tests. During the research, the coverage metric of the cards was also considered [5]. It allows you to measure the coverage of methods modified during the implementation of the card at the level of ui-tests and manual tests, but still about code coverage.

Pros:

- simple estimation algorithm;
- automated process.

Minuses:

- code coverage does not mean that the user's scenario has been tested.

As a result, we see the percentage of autotest coverage. But code coverage does not provide information about user scenarios, so the result of such a measurement requires additional analysis – what actions in the system correspond to covered or uncovered methods.

The coverage measurement options described above work from the development side and from the product side. Therefore, it was decided to try to measure coverage from the point of view of users and their scenarios of work in the system.

3.3 User scenarios coverage

In this approach, a problem arises – where to get information about user scenarios?

One of the methods that allows analysts and developers to decide on further work on the product is the research of basic user scenarios, their simplification and acceleration. To do this, metrics are used – special get-requests that are sent to each action marked in the system. Due to the peculiarity of the system, it does not used Yandex or Google metrics. For the product its own metrics are developed. It is more flexible and convenient for further analysis and work with them.

There are many cases for using metrics. Metrics allow for a wide range of product and user analysis. That is why the full picture of the actions of users in the system, it was decided to get on the basis of metrics.

Pros:

- simple estimation algorithm;
- automated process;
- few resources required for support.

Minuses:

- a metrics mechanism is required;
- coverage measurement algorithm is required.

4 Implementation

When processing metrics, you can make chains of user cases (Figure 1). But the question arises: what is considered a user case?

It is difficult to measure coverage for the full user scenarios in the system – there are many unique scenarios, the scenarios are long, they are inconvenient to store and handle. However, most of the actions in the chain of scenarios are duplicated, so it was decided to evaluate the scenarios from the point of view of user cases — small atomic scenarios — transitions from one state to another.

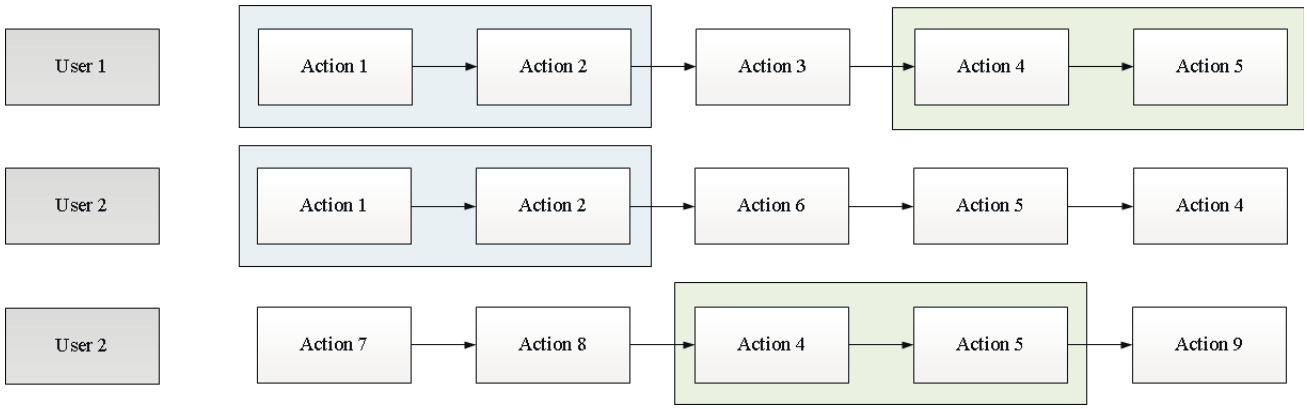

Figure 1: User cases chains

State-transition pairs are collected for each user, then the number of identical pairs for all users is calculated. The result is a sorted by frequency list of the most popular user actions in the system.

Coverage assessment algorithm:

1. We count the top user transitions

2. We consider the transitions from the test site after the autotest run
3. Compare transitions and calculate the percentage of coverage

Advantages of the system:

- the system is automated and requires a minimum of support;
- coverage percentage can be quickly obtained at any time;
- You can choose the period for which is considered the top user transitions.

Minuses:

- a metrics mechanism is required;
- coverage assessment algorithms are required.

Transitions must be clear – no additional work should be done to match the transition and the real user action, so metrics must send clear, human-readable data about every action in the system. Another disadvantage is that in order to ensure the completeness of the obtained data on user actions, metrics should provide 100% coverage of actions in the system.

To solve this problem, automatic marking of all pages with metrics was done – each button that you can click on, when you click, sends data which can uniquely identify a button (content, parent name, url, etc.). This event information remains to be sorted, summarized and used as input data.

Baseline data is placed into a table in the ClickHouse database. The coverage analysis system aggregates this data, taking into account the time of the event and the user id, with the help of which transition pairs are built.

The result of the system work is a table consisting of the following columns: state 1, state 2, degree of coverage (true / false), the number of transitions for the selected period.

The algorithm of the program is shown in Figure 2. The percentage of coverage obtained as a result of the program's operation allows, quite accurately and based on real data, to provide information about the autotest testing of the main user scenarios.

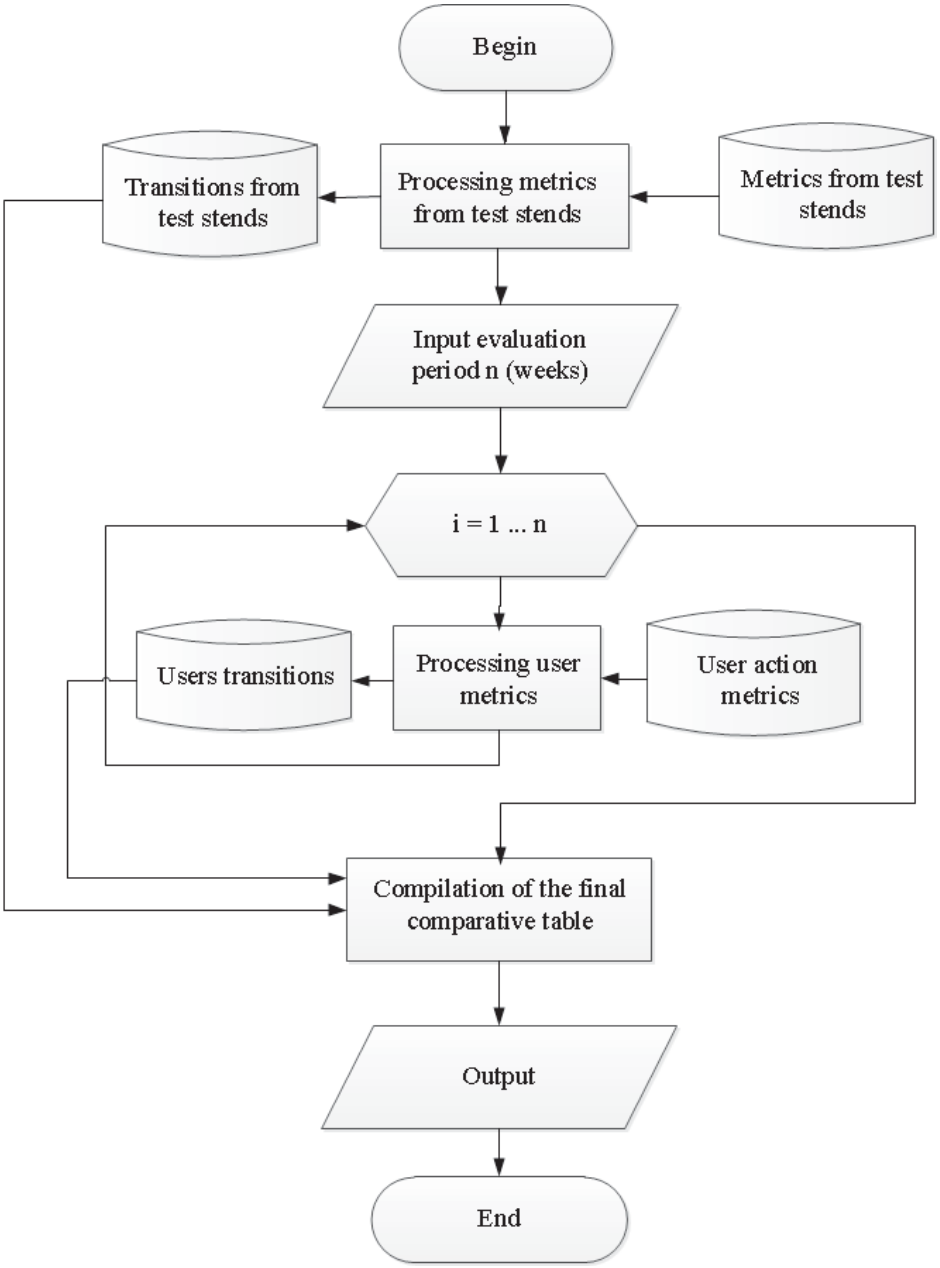

Figure 2: The algorithm of the program

5 Conclusion

The developed system allowed us to measure the coverage of user scenarios with ui-tests, based on the data about the real actions of users in the tested web application. The system meets the requirements specified in the first stage of its development. As a result, a percentage of test coverage was obtained, as well as a list of all user scenarios, sorted by the popularity of the case. For each scenario, a coverage criterion was obtained.

References

1. Mike Cohn. Succeeding with Agile: Software Development Using Scrum { Addison-Wesley, 2009.
2. J.M. Voas, K.W. Miller. Software Testability: The New Verification. IEEE Software, May 1995
3. S. Jungmayr. Identifying test-critical dependencies. In Proceedings of the International Conference on Software Maintenance { IEEE Computer Society, October 2002.
4. Rob Patton. Software Testing { 2nd Edition, 2005
5. Jakob Rott, Rainer Niedermayr, Elmar Juergens, Dennis Pagano { IEEE/ACM 8th Workshop on Emerging Trends in Software Metrics (WETSoM), 2017