The following tasks have been developed of achieve this goal:

1) Perform a performance analysis of React and Vue.js.

2) Conduct load testing by adding different numbers of identical components. 3) Analyze the methods of rendering JSX format in React and Vue.js. 4) Debugging and exploring alternative methods of rendering JSX format.

References:

1) **Google Trends -** Statistics as of the time of writing https://trends.google.ru/trends/explore?q=VUE,React

MODELING AND THE USE OF GRAPHS AND GRAPH THEORY ALGORITHMS IN SOFTWARE TESTING

Svietlinskyi O.A., student,

Golian N.V., Associate Professor,

Kharkiv National University of Radio Electronics

Graph theory is one of the basic tools of mathematics that is used to represent sets and the relationships between them. Most programs and algorithms can be represented as a graph. Coverage of graphs in testing is useful because using different criteria it is possible to detect errors. This article describes several graph theory methods, where they come from, and how they can be used to improve software testing. For example, in black-box testing, graph theory can be of great importance if you have to test application states and transitions, entity state graphs, etc [1].

Traditional software testing entails a tester studying the system and running test scenarios over it to test the system. Each product's test cases are created separately.

When the same pesticide is used against them, a similar occurrence occurs with some insects. As a result, the insects acquire resistance to the pesticide, rendering it worthless. Therefore, regardless of the sort of test being done, test kits must be adjusted and updated.

The first thing we need to do to test with graphs is to identify the states. To do this, we take the initial application point (the initial state) and denote it as the first vertex. Actions on the screen are shown as graph edges and states as vertices. It is

worth noting that an edge from a vertex can lead to the same vertex. For example, such action as refreshing a page - after this action, the state comes to the source [2]. You can make change at almost every step, which generates an additional state for every screen where possible. Making changes is important because they must be preserved when returning to previous screens. In this way, we get a graph (Fig. 1).

Because of our study of various algorithms on graphs, we have chosen several ones that solve the shortest path problem. The basic idea is that we need to get a set of chains in a graph, which implies a software testing algorithm.

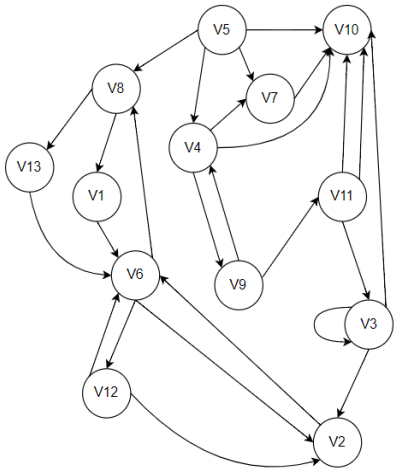

Fig. 1 - example of a graph for software testing

There are different approaches to solve this problem, for example, in the situation when we should always start from the first vertex of the graph (a program launch) and end at the last one (a program closure or getting the desired end result), i.e., to make a traversal of the functionality in depth. This variant is suitable for applications using transactions that are either executed completely or not executed at all.

The Dijkstra algorithm makes it possible to find the shortest paths in a graph between given vertices. The Bellman-Ford algorithm is based on dynamic programming. By filling the table of paths from between different vertices you can reuse them when calculating a new path [3]. Dijkstra's algorithm with the modified

binary heap is the same algorithm, but finding the minimum and updating an element takes O (log(V)), where V is the number of vertices in the graph.

Since the example graph provided by figure 1 has a small number of vertices and edges, it was decided to change the sum of algorithm running times after a hundred iterations for the average result, as it is quite possible that because of the load on the system where the algorithm is executed once, too slow results will be obtained that do not reflect reality. As a result, we get the results (Fig. 2).

The case of finding a path / Algorithm	The first	The second	The third
Dijkstra's Algorithm	8.34ms	6.78ms	7.7ms
Bellman-Ford Algorithm	25.75ms	20.34ms	23.71ms
Dijkstra algorithm with modified binary heap	8.01ms	6.96ms	7.7ms

Fig. 2 - The execution time of the three algorithms for all pathfinding cases

As can be seen from the data above, in general, the relationship between the time costs of the algorithms is predicted by their complexity, so the slowest algorithm is the Bellman-Ford algorithm, but it is worth repeating that the algorithms used edges without negative values, otherwise, this algorithm would have been the only preferred one.

Software models are a fantastic method to represent it. They're straightforward and make updating tests a breeze. Graph traversal can be used to describe testing. As a result, graph theory techniques can be applied to model development. On the same model, tests might keep changing. Methods for traversing are generic and can be used for a variety of models.

References

1. Harry Robinson, Graph Theory Techniques in Model-Based Testing - 1999 International Conference on Testing Computer Software – 10 c.

2. Apfelbaum, L. (1997) “Model-Based Testing”, Proceedings of Software Quality Week 1997 – 14 с.
3. Євстигнєєв В.А., Касьянов В.М. Теорія графів: алгоритми опрацювання безконтурних графів. - Новосибірськ: Наука, 1998. – 360 с.

MOBILE AUTOMATION TESTING

Rusanov I.S., student,
Suknov M.P., PhD, Associate Professor,
Kharkiv National University of Radioelectronics

Mobile testing involves using tools or open-source frameworks to assess the functionality, usability, and performance of mobile applications. Whether it's smartphones, tablet PCs, or the latest iPhone version with minor updates, conducting mobile testing and automated testing becomes essential to identify regression bugs and ensure the delivery of high-quality applications[1].

First of all, it is very important to choose the right automation tools for mobile testing. We must pay attention to the following factors: speed, ease learning and using, level of coverage of the main tasks of mobile testing, possibility of integration with CI/CD systems or other native platforms. Today there are many mobile testing tools: Katalon, Appium, Espresso, XCUI Test, Robotium, Robot Framework, Selendroid, Xamarin.UITest etc. The choice depends on the factors mentioned above and the specific tasks that you want to solve with mobile automation. However, in general cases, Appium is the best option. It is fully open-source and free tool, which allows test and code reusability between iOS, Android, and Windows, and easy adoption for those experienced with Selenium. But the main advantage of Appium is a big set of supported languages. Thus, we can choose a programming language that will be convenient for us, write test scripts and do not pay attention to other factors.

Appium is primarily utilized in the realm of software test automation to ascertain whether a given application's functionality functions as intended. Unlike other forms of software testing, UI automation empowers testers to write code that simulates user