The essence of testing

TESTNAVIGATOR

TESTNAVIGATOR

Crucial questions of testing

How should test cases be prioritized?

Test cases play a crucial role in the software development process, as they validate that the program and its functions work as intended. With thousands of test cases, it is particularly important to rank them appropriately to maximise the effectiveness of the tests focusing on:

- testing the most critical and high-risk areas of the code
- prioritizing tests that provide the most value to the users
- code-level changes that have the highest risk to delivery

What are the changes and which test cases are affected?

When changes are made to the code, it is important to understand how these changes may affect the existing test cases and what test cases should be re-run to cover the modified code fragments in order to ensure that the code is still functioning correctly. If the coverage data is available for the previous test case executions, then the test cases that affect the changed code fragments can be easily selected depending on the code-level change detection.

Has the current software version been thoroughly tested for changes?

Ensuring that software is thoroughly tested before the current version is released is an essential part of the software development process in order to ensure that the software is functioning correctly and meets the updated requirements. The crucial question is how to determine the modifications of the new software version and how to ensure that all these code-level changes have been properly tested?

Is there a tool to guide the QA team to cover all code-level changes?

TESTNAVIGATOR

Key features

Method, branch level measurement

By measuring coverage at both the method and branch levels, TestNavigator provides a deeper understanding of your Java code during the testing phase to release it with minimal risk. The slowdown free branch level measurement helps to test all possible paths through the code, ensuring that all relevant scenarios are covered and no potential issues are missed.

Change detection

TestNavigator's advanced change detection technology, an essential part of our comprehensive coverage measurement solution, ensures that critical code-level modifications are never missed, and Java code is thoroughly tested to the highest quality. By accurately detecting byte-code level changes made by programmers, TestNavigator gives the confidence to release new versions of Java application with minimal risk by recommending test cases for all changes.

Test selection

Using a sophisticated test selection algorithm, TestNavigator helps to prioritize testing efforts, ensuring that the most important test cases are executed first. By automatically scoring the potential impact of changes and code-level information on test cases, such as complexity and coding issues, testers can work much more efficiently in a new so-called coverage-driven testing process.

Test automation

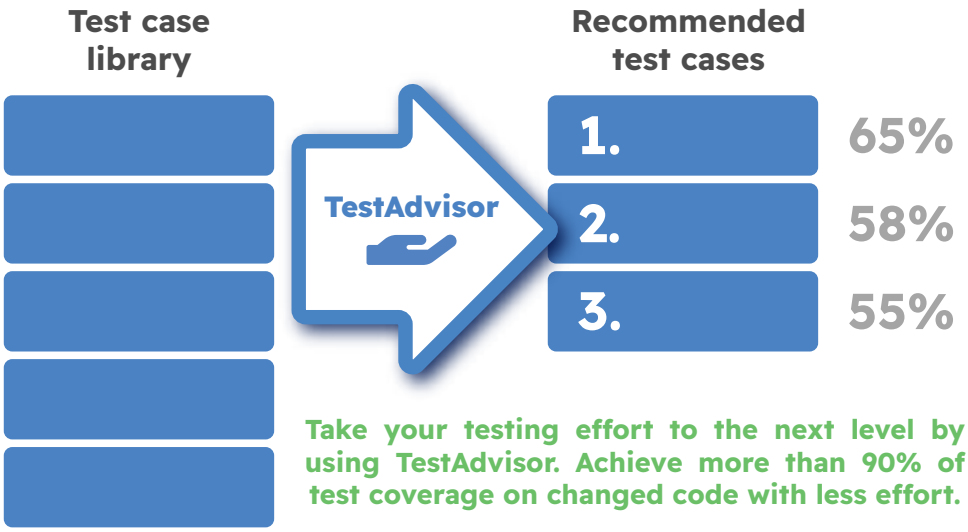
TestNavigator has a full-featured public API layer that allows to leverage the powerful coverage measurement and test selection features of TestNavigator in your own test automation system. Integrating TestNavigator can improve the efficiency and effectiveness of the testing process, ensuring that the current version of the application is thoroughly tested, taking into account code-level changes.

Low-level coding issues

Through the QualityGate integration, TestNavigator provides a number of code-level issues that are derived from QualityGate' deep code analysis, whose execution can have a serious impact on the application under test such as a possible nullpointer exception. If these high-priority coding issues are not covered and tested, how much does the risk of release increase?

TESTNAVIGATOR

Optimized test selection



Take your testing effort to the next level by using TestAdvisor. Achieve more than 90% of test coverage on changed code with less effort.

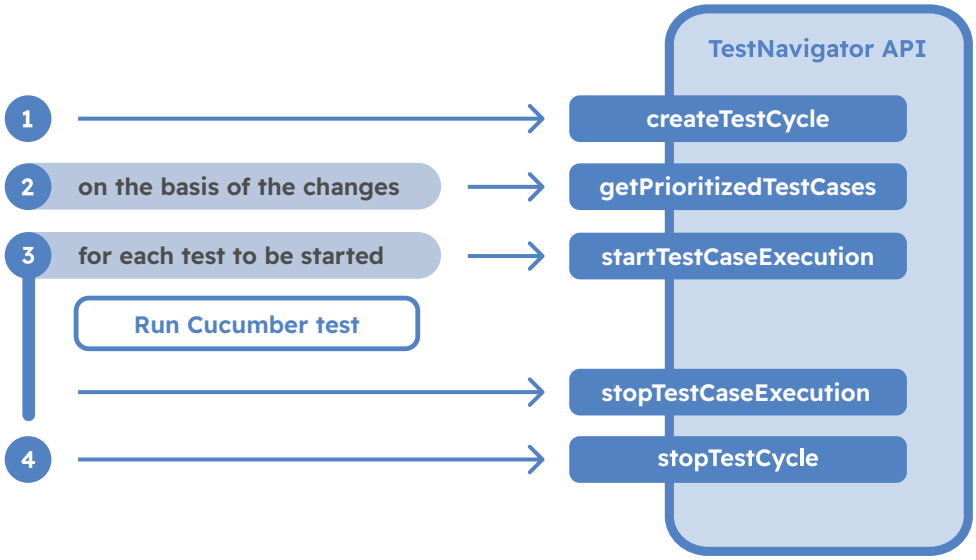
Optimize your test with a more sophisticated test case selection strategy. Rank the test cases with a so-called TestAdvisor score based on:

- Coverage results of previous tests
- Modified code likely to be affected by the test case
- Critical code violations
- Complexity of the code fragments

Testers receive TestAdvisor scores for all test cases in real time, without any user interaction, which helps them to select and execute test cases that have the greatest impact on the system under test. In order to make testing more thorough and well-targeted, one of the most important parameters for calculating the score is the expected execution of detected code-level changes based on historical results, which allows us to test new versions of the software more thoroughly than ever before. After each test case execution, TestAdvisor updates the scores which is a powerful assistance for executing uncovered and complex code fragments, achieving professional coverage-driven testing encouraged by TestNavigator.

TESTNAVIGATOR

Revolutionize Test Automation with TestNavigator



Quality Assurance teams often make the mistake of not performing a deeper analysis of the true effectiveness of their available regression testing system. A huge number of automated test cases used, but how effective are they at detecting bugs caused by code-level changes? Is there a way to select and prioritize the existing test case library to avoid wasting time, effort and money?

"You can't control what you can't measure"

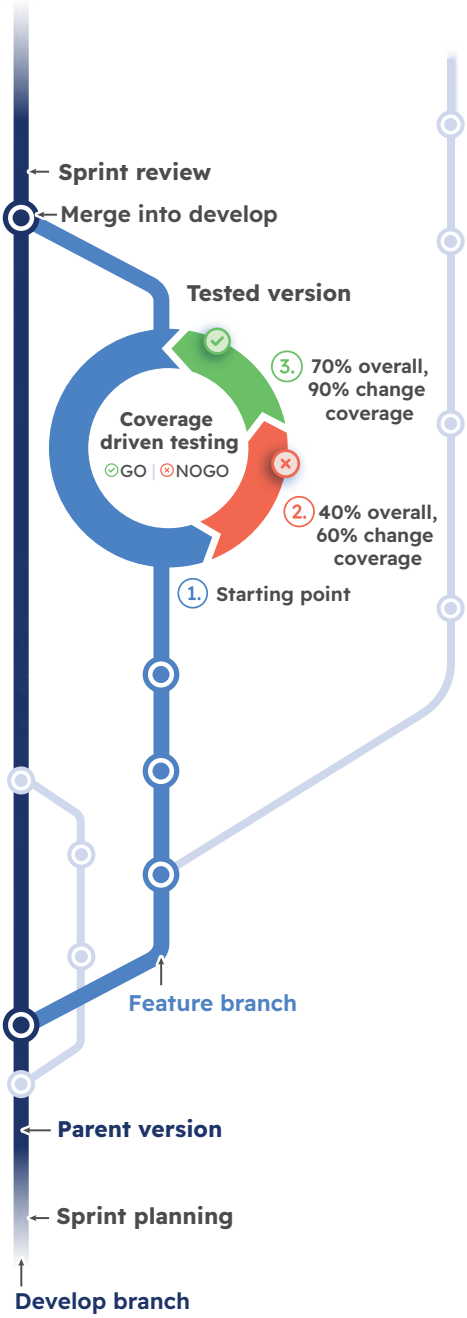
Integrating coverage measurement is the first and most necessary step to be able to monitor the true effectiveness of our test cases. TestNavigator has a public API layer that allows it to be effortlessly embedded in any automated system such as Cucumber, Selenium, Robot framework, Protractor. By using a few straightforward API endpoints in the background, TestNavigator can measure the code snippets affected by the running test case without any user interaction.

Focusing on changes

TestNavigator can detect all code-level changes, allowing TestAdvisor function to use a sophisticated selection algorithm to pre-sort test cases affected by the change before test execution. With an effortless CI/CD integration, this leads to a de facto optimized test case execution strategy. Fewer test cases, faster execution: a cost-effective solution for more professional testing.

TESTNAVIGATOR

Supporting agile



Coverage driven testing

Optimize sprint backlog-based test design by analyzing non-covered code snippets in order to create new test cases for reaching uncovered decision points. Make change-based test case execution more efficient by running high priority test cases recommended by TestAdvisor.

GO | NOGO

Coverage-based quality gate for professional testing.

- Need more test cases.
- All coverage-based exit criteria are met.

1. Starting point

Start testing the current feature under development by analyzing acceptance criteria and creating test cases for them. Acceptance criteria are the conditions that a software product must meet to be accepted by a user, a customer, or other systems.

2. Measuring point

After the first round of testing, additional test cases should be created based on the evaluation of the coverage-based results.

3. Exit point

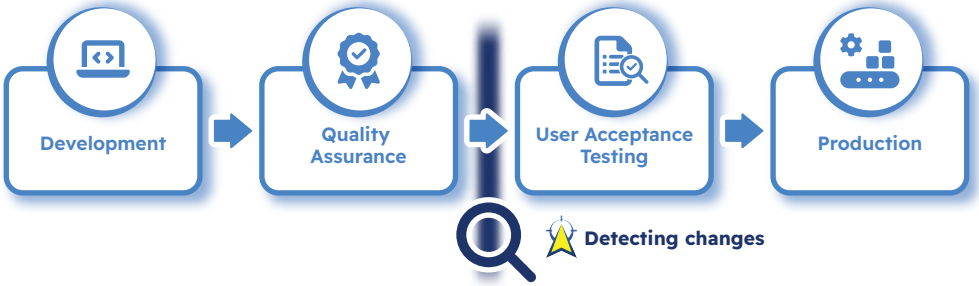
If coverage targets are met, testing can be completed as the function has been thoroughly tested.

Focusing on changes

TestNavigator can detect all code-level changes, giving a more accurate picture of the testing results for the current development. The outcome is clear and unquestionable. Have you been able to thoroughly test the recent changes, or are there any uncovered code fragments?

TESTNAVIGATOR

User Acceptance Testing



The essence of UAT

The main Purpose of UAT is to validate end-to-end business flow based on the following de facto phenomenon: sometimes the functions delivered by the supplier do not meet the enterprise-level system requirements. Stability and fault-proneness are other key issues that can increase the risk of production deployment. Question is simple from the perspective of anyone operating custom software: have the functions involved been thoroughly tested?

Handover problems

In the context of custom software, there is always a crucial question of whether the functional modifications match the code-level changes. What happens if there are hidden code-level alterations that could affect the use of the released product? UAT is carried out in a separate testing environment with production-ready data setup but there is no point in getting closer to the production environment without knowing everything about the current release. The undeniable truth is it that missing or incomplete information leads to inefficient testing, which is the most critical point of the UAT.

Coverage driven testing

As a surveillance system, TestNavigator easily supports UAT process by detecting all executed parts of the software under test at runtime. With information about uncovered parts of the code, UAT testers can apply new test cases, increasing coverage and reducing the risk of live bugs, which always result from uncovered parts and incomplete testing. Several useful metrics are available for the uncovered parts detected, such as complexity or the number of untested code-level issues that may lead to errors.

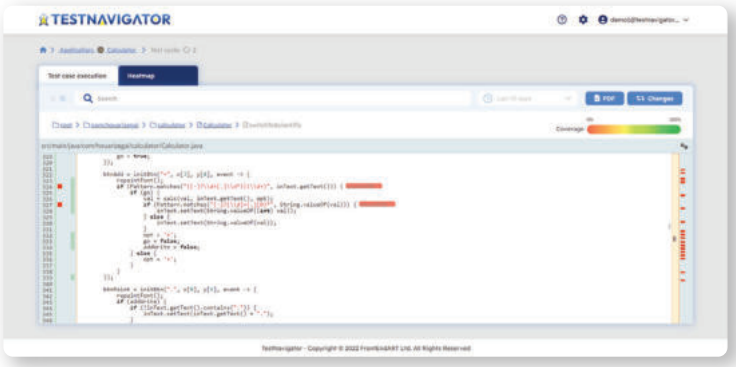
Focusing on changes

As a customer-side quality gate, TestNavigator can detect all code-level changes, giving a more accurate picture of the testing results for the current release candidate. The outcome is clear and unquestionable. Have all delivered high- and low-level changes been thoroughly tested, or are there any uncovered and risky code fragments?

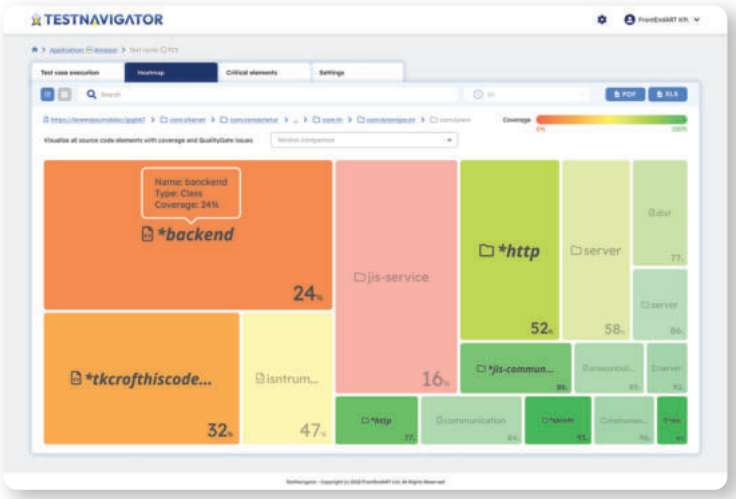
TESTNAVIGATOR

Collaboration between DEV and QA teams

TestNavigator provides a low level source code view to support collaboration between teammates. With this simple but powerful feature, testers can create new test cases by thoroughly reviewing code that is not yet covered.



TestNavigator allows developers and testers to check low-level branch coverage to identify the parts that need to be covered in the testing phase. It shows which parts of the code have been tested and which test cases have been executed to reach that part.



A well-defined heat map can be used to walk through the entire code base in order to identify the parts of the code need to be tested in a more sophisticated way. More useful information for testing: lines of code, code complexity, critical and blocker warnings.

TESTNAVIGATOR

A practical extension of the 7 principle of testing

1. Testing shows a presence of defects

The purpose of software testing is to show the presence of bugs and not the absence of bugs, therefore cannot prove that the software is bug-free. TestNavigator ensures that the most important code fragments, such as changed parts, will be covered during the testing phase. Increasing the number of detected bugs, thereby indirectly improving/increasing quality and reducing the risk of software crashes.

2. Exhaustive testing is not possible

Testing all possible (valid or invalid) inputs and preconditions is known as exhaustive testing, which is impossible in principle since it has infinite cost. TestNavigator highlights which parts of the system still need to be tested in order to reach the desired quality gate, also known as GO/NOGO exit criteria without executing redundant or unnecessary test cases and overloading our testing resources.

3. Early Testing

Fixing a bug detected in the early phase of the software development process is significantly less expensive than a software crash on the customer's live system. TestNavigator provides application-level version management to support agile development such as the scrum methodology, making coverage measurement a key factor in the development phase to test all parts of the system early and reduce the cost of bug fixing.

4. Defect clustering

According to the Pareto principle of software testing, 80% of software bugs come from 20% of the modules. Most of the bugs are usually hidden in the changed and more complex code fragments. TestNavigator can show how thoroughly the changed and complex parts of the application have been tested so far and can recommend test cases to be executed for more thorough and efficient testing.

5. Pesticide paradox

Repeating the same test cases over and over again does not necessarily lead to the discovery of new bugs. The phenomenon is that software under test can "build up" resistance to testing, therefore the test cases no longer work as effectively. Based on changes and low-level code metrics, TestNavigator calculates a score for each test case to give a more accurate priority for more efficient and well-aimed testing.

6. Testing is context-dependent

The testing approach depends on the context of the software being developed, therefore different types of software require different types of testing. TestNavigator allows you to tailor testing to your specific needs based on the context of the software under development, saving you time and ensuring that test cases are relevant and effective.

7. Absence of errors fallacy

This should be always kept in mind that just because at the time of testing, defects were not found in the software, doesn’t mean that the software is ready to be shipped. TestNavigator can easily show untested and uncovered code fragments, making it clear whether further testing is needed or not. In most cases, analysis of the non-covered code fragments is the key to moving forward and making testing more efficient.

Developed by FRONTENDART

+36 30 4381 533
+36 30 4780 980

info@frontendart.com

H-6720 Szeged,
Somogyi utca 19. II. emelet