Test Automation Technology for Analyzing Test-activity Data and Detecting Bugs

Haruto Tanno, Hiroyuki Kirinuki, Yu Adachi, Morihide Oinuma, and Tatsuya Muramoto

Abstract

There is a growing demand for the early release of software while holding down costs. Software testing, which makes up a large portion of overall development costs and is essential to ensuring a certain level of quality in software, can be viewed as the cornerstone of quality, cost, and delivery in the development process. This article describes technology undertaken by NTT Software Innovation Center for achieving a dramatic leap in testing efficiency and discusses the future outlook for this technology.

Keywords: software testing, exploratory testing, test script

1. Importance of software testing

The software-development process is summarized in **Fig. 1**. In this process, software defects that could not be removed during testing are released in that state to the user, so testing is a critical process essential to software quality assurance. Yet, attempting to do all testing manually is extremely costly. Users' needs have been changing rapidly, and software and hardware that constitute the operating environment have likewise been evolving at a rapid pace. This makes it necessary to revise software quickly as the need arises and release software updates frequently at short intervals. However, to conduct software releases frequently while maintaining a certain level of quality, testing cannot be limited to just the new additions even in a small-scale update. It is also necessary to conduct regression testing with respect to all existing features at every release to check whether they have been adversely affected by new features or a new operating environment. This need can also incur high costs. NTT Software Innovation Center seeks to revolutionize testing—the cornerstone of quality, cost, and delivery in software development—and achieve a quantum leap in productivity in the software-development process.

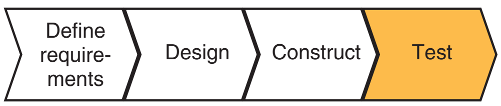


Fig. 1. Software-development process.

2. Current state of software testing

The purpose of software testing includes checking that the software is behaving normally and reducing the number of software defects. As shown in **Fig. 2**, the testing process can be broadly divided into five tasks: test planning, test design, test execution, test management, and test reporting. Test planning involves decisions on test period, resource allocation, etc. based on an overall development plan. Test design involves identifying test variations that should be carried out, exhaustively designing test cases, and for each test case, refining a specific procedure for executing the test and creating a script for automatic execution. Next, test execution involves providing

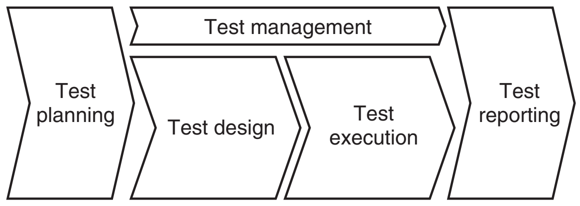


Fig. 2. Testing process.

input data for each test case, running the software, and checking whether the software is behaving as expected. Test management involves managing test-execution conditions as needed and revising the plan if conditions warrant. Then, once all tests have been completed, the final task is to compile test results and issue a test report, thereby completing overall testing. Among these tasks, test design and test execution play major roles in the testing process.

The first issue in traditional exhaustive testing is the high costs incurred in both design and execution. Pursuing completeness in testing with the aim of improving quality can take a massive amount of time while extending the period until release. Moreover, evaluating completeness is inherently difficult. For example, using specifications as a standard for completeness depends heavily on the quality of those specifications, and the content of the test itself cannot be evaluated on the basis of code coverage. In addition, simply clearing certain index values does not mean that the degree of quality improvement is well understood.

The second issue is the costs incurred in automating regression testing, which is required when issuing releases frequently in short cycles. Although many frameworks and libraries, such as JUnit and Selenium, are currently available for automatically executing tests, scripts must be created for such automatic execution, which can be very time consuming. To make it worse, a completed script does not mean that no more work is needed since it must be revised together with any revisions made to the software targeted for testing. This type of maintenance work is also labor intensive.

3. World we aim for

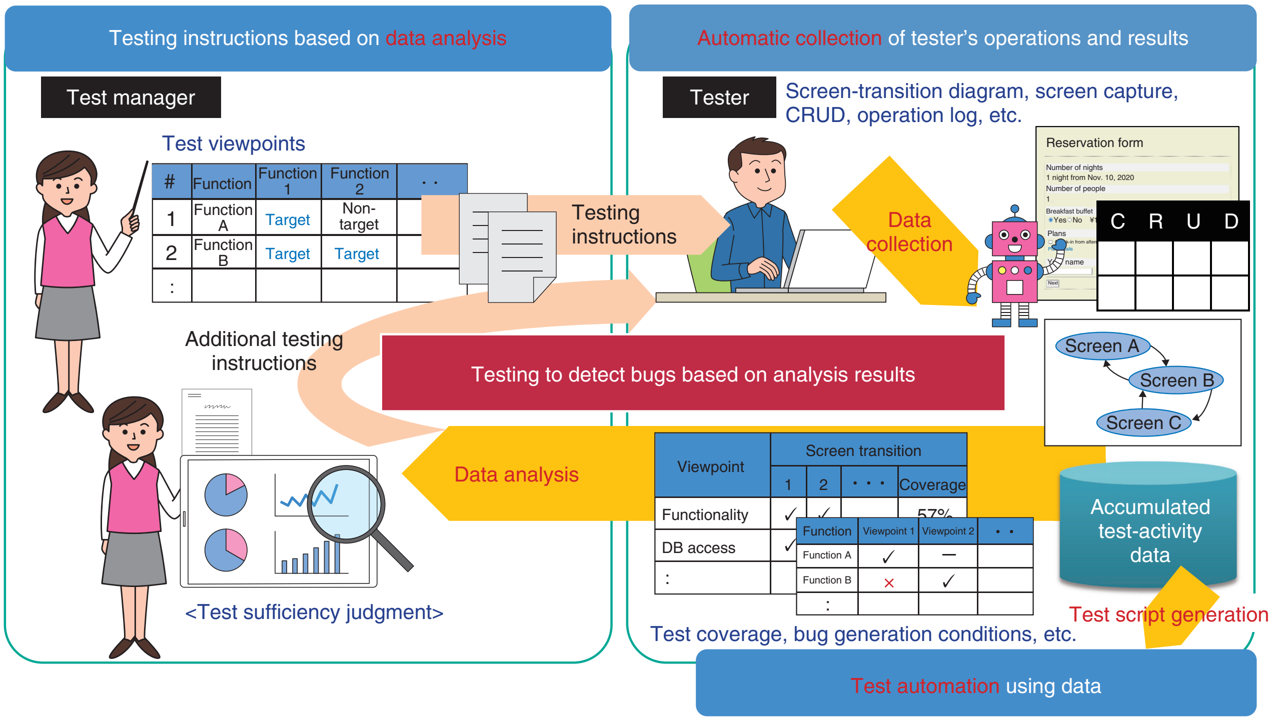
To achieve ultra-high-speed development that can handle increasingly diverse and vague business requirements and rapidly evolving businesses, the approach taken by NTT Software Innovation Center is to establish artificial intelligence (AI) development technology that can substitute or even excel in some human work and to develop software through human-AI cooperation. The world we aim for to achieve such ultra-high-speed development is shown in **Fig. 3**. In our approach, instead of haphazardly pursuing completeness, we select the locations that should be tested and concentrate our efforts there. In addition, we make successive judgments as to what locations to select and concentrate on by collecting and analyzing test-execution conditions and results. This approach solves the problems surrounding traditional exhaustive testing and achieves a quantum leap in testing efficiency. Furthermore, by using test-activity data accumulated over time and automatically generating easy-to-maintain test scripts, it has become relatively easy to automate regression testing, which makes for prompt releases after making software updates.

The NTT Group develops many business applications that use web applications as front ends and tests these applications through integration testing. In this article, we describe LatteArt as a technology targeting integration testing, which has a great need for efficiency improvements.

4. LatteArt: Technology for analyzing test-activity data and detecting bugs

A business application has many use-case scenarios, features, and screens, and each screen may have many combinations of input patterns. This incurs high costs in traditional exhaustive testing. While there are tools that support test design through automatic design of exhaustive testing on the basis of some type of model (e.g., software design model), executing all required tests is still labor intensive, thereby placing a limit on the degree to which overall testing can be made more efficient. In addition, automating the testing of web applications requires the creation of test scripts for executing screen operations automatically. In addition to the fact that the manual creation of such test scripts drives up costs, a test script must be revised whenever the associated web application is updated, which is also costly from a maintenance point of view. In this regard, there are capture & replay tools (e.g., SeleniumIDE) that can be used to create scripts even without advanced skills, but creating scripts in this manner still requires work. Moreover, because test scripts recorded by capture & replay are not modularized, they suffer from low maintainability.

To solve these problems, we developed a technology


Testing instructions based on data analysis
Automatic collection of tester's operations and results
Test manager
Tester
Screen-transition diagram, screen capture, CRUD, operation log, etc.
Test viewpoints
Testing instructions
Data collection
Reservation form
C R U D
Additional testing instructions
Testing to detect bugs based on analysis results
Screen A
Screen B
Screen C
Data analysis
Viewpoint
Screen transition
Coverage
Functionality
DB access
Accumulated test-activity data
<Test sufficiency judgment>
Test script generation
Test coverage, bug generation conditions, etc.
Test automation using data

CRUD: create, read, update, delete
DB: database

Fig. 3. World we aim for.

called LatteArt for analyzing test-activity data and detecting bugs. LatteArt has the following features.

(1) Collection of test-activity data

This step involves automatically collecting test-activity data consisting of the tester's operation log and web-application screenshots as well as test objectives input by the tester while executing tests, discovered bugs, findings, etc.

(2) Analysis of test-activity data

In this step, the test manager gives instructions on test content at a general level in the manner of combining the test viewpoint and test-target feature without creating a detailed test [1, 2, 3]. The tester then conducts tests based on those instructions. In addition, test-activity data that have been automatically collected are analyzed and visualized using a variety of data models, enabling the test manager to determine if the test is sufficient or give instructions for additional tests. For example, the models shown in the sequence diagram and screen-transition diagram in **Figs. 4** and **5**, respectively, can visualize test-activity data. Furthermore, as shown in **Fig. 6**, the test manager can focus on a specific screen transition using the screen-transition diagram to check a list of input patterns that occur when making that screen transition. Selecting locations that should be tested and concentrating on those locations in this manner facilitates testing that can detect bugs with good efficiency.

(3) Application of test-activity data

Test-activity data can be used to automatically generate test scripts that modularize screen-element locators based on the concept of page-object patterns. This approach simplifies the maintenance of test scripts even when revising an application and enables regression testing to be automated, eliminating the labor that would otherwise be required.

5. Achievements and future outlook

We aim to become a leader in test technology based on the analysis of test-activity data and change the conventional way of doing testing in the system integration industry. We are currently evaluating the application of LatteArt through joint experiments with NTT operating companies and are receiving

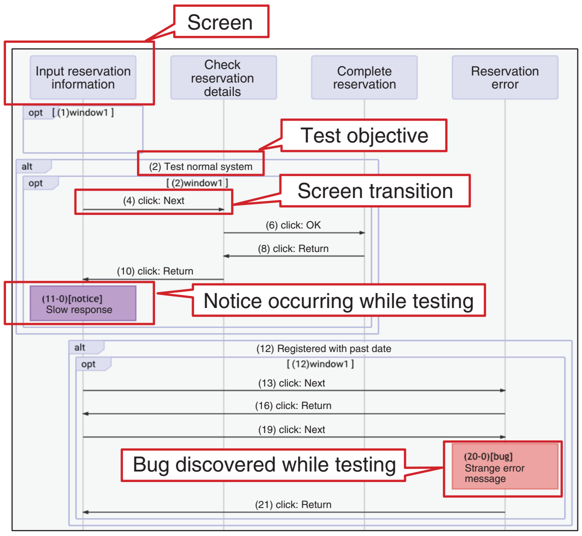


Fig. 4. Sequence diagram.

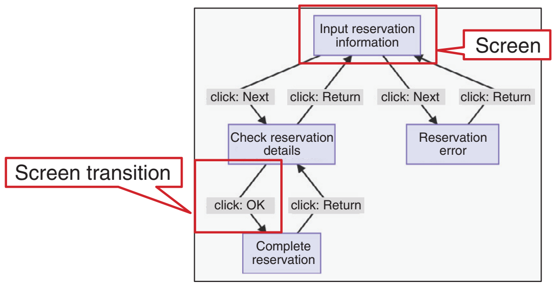
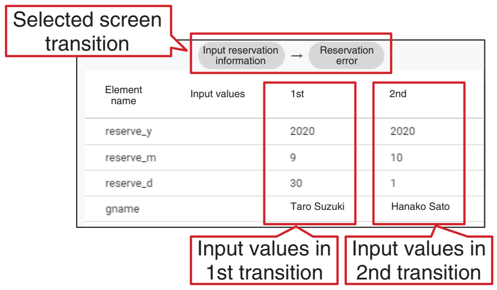


Fig. 5. Screen-transition diagram.


Selected screen transition
Input reservation information → Reservation error
Input values in 1st transition
Input values in 2nd transition

Element name	Input values	1st	2nd
reserve_y		2020	2020
reserve_m		9	10
reserve_d		30	1
gname		Taro Suzuki	Hanako Sato

Fig. 6. Input patterns during screen transition.

positive responses from development sites. We have also received high evaluations for LatteArt through presentations at academic societies in Japan and have received a number of awards for LatteArt.

In addition to the analysis and visualization of test-activity data and the automatic generation of test scripts, as introduced in this article, we can consider a variety of research directions in the use of test-activity data automatically collected and stored using LatteArt. The following are examples of these directions.

- Test recommendations: A test with a high probability of discovering bugs can be automatically recommended by analyzing which type of test was conducted by a tester when he/she discovered bugs.
- Test education: Test-activity data of experienced testers can be used as educational material. In addition, analyzing and comparing test-activity data of multiple testers should make it possible to measure testing skills.
- Application to areas other than testing: Test-activity data could be used to restore the specifications of software targeted for testing, automatically generate manuals, etc.

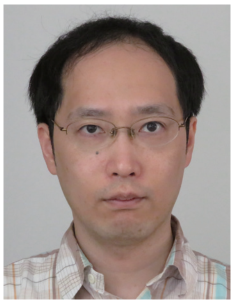
Going forward, our goal is to create an ecosystem for revolutionizing testing and software development centered around LatteArt. We will achieve this by collaborating with companies and universities outside the NTT Group to incorporate various types of industrial and academic knowledge in our research. For this reason, we plan to study the conversion of LatteArt to open-source software and work on widely disseminating this technology. We are committed to making steady progress in researching and developing this technology based on feedback from development sites. Our ultimate goal is to create a world in which AI can conduct testing automatically by collecting massive amounts of test-activity data.

References

[1] I. Kumagawa, A. Mineo, H. Tanno, H. Kirinuki, and T. Kurabayashi, "SONAR Testing, New Testing Method that Ensures Both Efficiency and Objectivity," Software Quality Symposium 2019, Tokyo, Japan, Sept. 2019 (in Japanese).

[2] H. Kirinuki, T. Kurabayashi, H. Tanno, and I. Kumagawa, "Poster: SONAR Testing – Novel Testing Approach Based on Operation Recording and Visualization," IEEE International Conference on Software Testing, Verification and Validation (ICST) 2020, pp. 410–413, Porto, Portugal, Oct. 2020.

[3] H. Kirinuki, T. Kurabayashi, H. Tanno, I. Kumagawa, and K. Nagata, "Novel Testing Approach Based on Exploratory Testing and Operation Recording," IEICE Tech. Rep., Vol. 119, No. 56, KBSE2019-7, pp. 43–48, May 2019 (in Japanese).



Haruto Tanno
Senior Research Engineer, Software Engineering Project, NTT Software Innovation Center.
He received an M.E. in 2009 and Dr. Eng. in 2020 from the University of Electro-Communications, Tokyo. He joined NTT in 2009. His research interests include software testing and debugging. He is a member of the Information Processing Society of Japan (IPSJ).

Morihide Oinuma
Senior Research Engineer, Software Engineering Project, NTT Software Innovation Center.
He received a B.E. and M.E. in electrical engineering from Keio University, Kanagawa, in 1984 and 1986. He joined NTT in 1986 and his current research interest is in software engineering. He is a member of IPSJ.

Hiroyuki Kirinuki
Researcher, Software Engineering Project, NTT Software Innovation Center.
He received an M.E. from Osaka University in 2015 and joined NTT the same year. His research interests include software testing and empirical software engineering. He is a member of IPSJ.

Tatsuya Muramoto
Senior Research Engineer, Supervisor, Software Engineering Project, NTT Software Innovation Center.
He received an M.E. from University of Tsukuba in 1996 and joined NTT the same year. His current research interest is in software engineering.

Yu Adachi
Researcher Engineer, Software Engineering Project, NTT Software Innovation Center.
He received an M.E. from the University of Electro-Communications in 2009 and joined NTT the same year. His research interest is in software engineering.