Test Automation Technology to Promote Early and Frequent Releases of Software at Low Cost

Haruto Tanno, Morihide Oinuma, and Katsuyuki Natsukawa

Abstract

There is growing demand to speed up the software release cycle while holding down costs in order to rapidly deploy services that meet the changing needs of end users. Considerable interest has been focused on technology for software testing, which accounts for a large share of total development costs, to ensure a certain level of software quality. In this article, we introduce some concrete measures at NTT for supporting test design and verification of test results in software testing.

Keywords: software testing, test design, verification of test results

1. Introduction

Software development is divided into processes, as illustrated in **Fig. 1**. Software errors not detected in the testing phase go out to end users in the release, so testing is clearly very important to ensure the quality of the software. As long as testing is done manually, however, it will be extremely costly. User needs and software/hardware development of the operating environment have been evolving at an ever more rapid pace in recent years, and this requires early and frequent releases of new or revised software to meet these needs (**Fig. 2**). To maintain quality through repeated software release cycles, regression testing must be done to make sure new portions of software—including portions implementing new functions and new operating environments—do not have an adverse effect on existing software capabilities, and this testing of legacy capabilities whenever software is released is also extremely costly.

In order to pursue software development that constantly improves upon quality, cost, and delivery (QCD), the NTT Software Innovation Center is researching and developing technology that contributes to a set of test automation tools as part of the

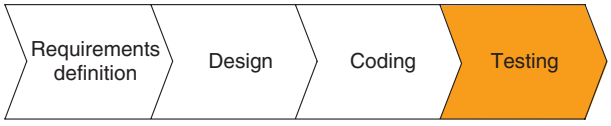


Fig. 1. Software development process.

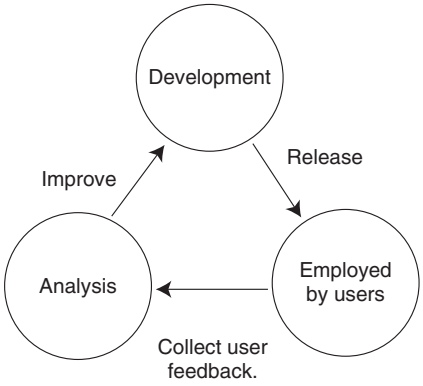


Fig. 2. Feedback loop.

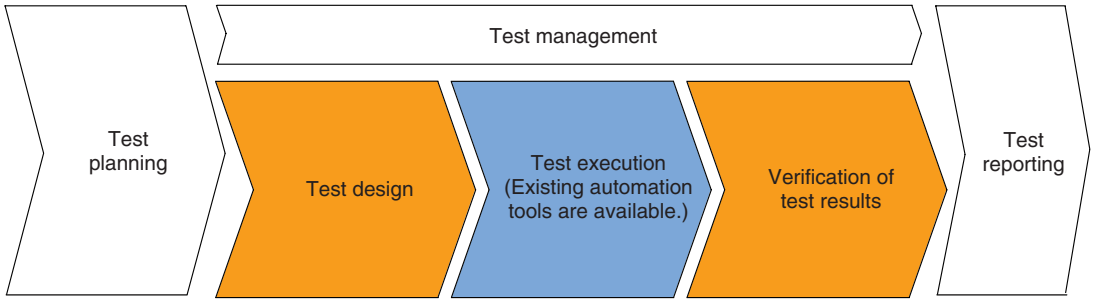


Fig. 3. Testing procedure.

Macchinetta tool suite [1]. The idea is to promote test automation that performs tasks by machine that were previously done manually.

2. Current state of software testing support

The objectives of software testing are to verify that software has been implemented according to the design and specifications, and to reduce the number of defects. The testing process mainly consists of five tasks: test planning, test design, test execution, test reporting, and test management (**Fig. 3**). In test planning, issues such as the time frame and allocation of resources for testing are decided based on the overall development plan. Test design involves clarifying the various tests that must be done, designing test cases comprehensively, refining specific test feasibility procedures for each test case, and creating scripts for automatic execution. In test execution, test data are input for each of the test cases, the software is run, and test results revealing how the software behaves for each of the test cases are recorded. These results are then referenced against verified test results to ensure the software behaves according to design. In test management, management of the state of test execution is carried out as needed, and the test plan is revised if necessary. When all tests have been executed, the results are summarized in test reporting, and the process is complete.

Three of these test processes are especially important tasks: *test design*, *test execution*, and *verification of test results*. Once test cases are produced in the test design, the tests must be implemented precisely with no missing test cases so they can be repeatedly used not only as new tests but also as regression tests in cases where software is patched or improved. Test execution and verification of test results must be carried out repeatedly against all legacy functions when dealing with software enhancements and new operating environments, and the burden increases exponentially as the scale of software increases. Therefore, these three tasks are areas in which the effects of automation are significant for maintaining software quality, cutting costs, and implementing early and frequent releases.

Considerable progress has been made in automating unit testing used to verify the functional operation of small individual units making up software, but automation has made little headway in dealing with integration testing of larger software programs combining multiple modules that include user interface (UI) screens or in dealing with system testing to catch system-level errors.

A number of tools have become generally available in the development workplace for automating integration testing and system testing. One example is SeleniumWebDriver, which automatically executes a web application test based on a prepared test script. However, we note that currently, test design and verification of test results still involve a considerable amount of manual labor. While tools supporting the test design of some functional testing are available, there are major barriers to introducing these tools in the development workplace. Obstacles include the need for testing staff to have specialized knowledge of the tools and the testing technique that the tools use, and the need to write descriptions in an unfamiliar language. In addition, verification of test results requires a great deal of visual inspection by technicians to ensure screens are displayed correctly and so on, and manually checking a large number of test trials one by one is extremely costly.

3. Research vision

With the goal of improving the QCD of test processes,

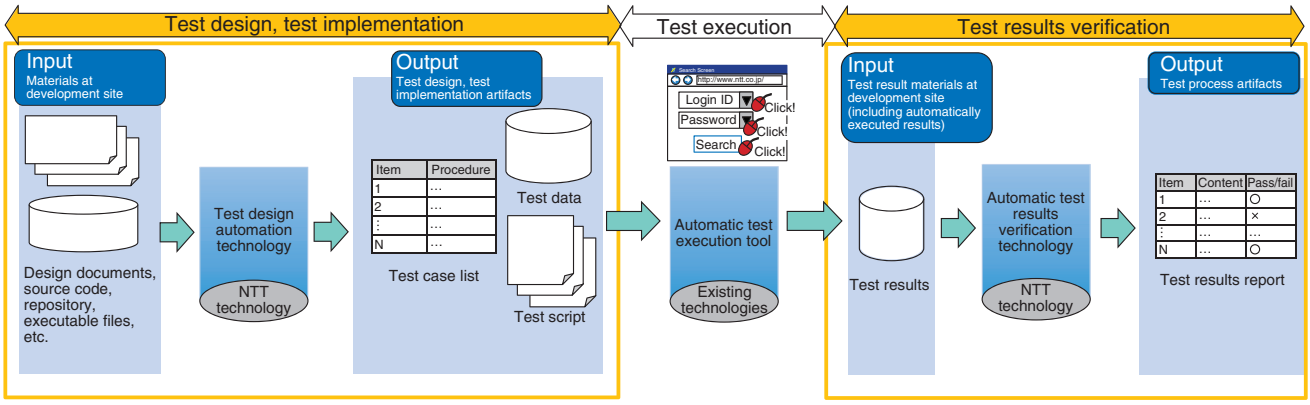


Fig. 4. Research vision.

our vision is to automate all testing during and after integration testing from test design to verification of test results, as shown in **Fig. 4**. We highlight the following two noteworthy features of this approach:

(1) Automating test design using design documents, source code, executable files and other materials available at the development worksite means that everything needed for the test execution—test case list, test data, and execution script files—is automatically generated. This effectively generates tests without errors or omissions at relatively modest cost.

(2) Any problem or error in the test execution results for each test case (application screenshots, etc.) is automatically detected. This markedly reduces the amount of visual verification work, prevents omissions from occurring, and thus improves the quality of applications.

In the following sections, we introduce two tools for automating the test design and verification of test results: the integration testing design support tool *TesMa* and the UI layout test support tool *ULTDiff*. We discuss these tools in the context of an enterprise application featuring a front-end web application developed using the Macchinetta framework.

4. Integration testing design support tool: TesMa

In order for end users to input data in a field on a web application screen, we must ensure that the software behaves as designed no matter what data are entered in the field. For example, if the correct input in the field must be a 10-digit number, the test design must test for correct entries such as “0123456789” but also test for the full range of potential incorrect entries: “012345678a” (violates the numerical requirement), “01234567890” (violates the number of digits requirement), and “01234567890a” (violates both the numerical and number of digits requirements). It is challenging to implement such a test design without errors or omissions even with highly skilled technicians, and the cost can be excessive.

To resolve this issue, we developed the integration testing design support tool called TesMa [2] that automatically generates test cases and test data needed for integration testing enterprise applications from the software design documents (**Fig. 5**). The latest version of TesMa goes beyond generating test cases and test data to automatically generate execution scripts to automatically run the test cases and test data. TesMa has the following features:

(1) The input for the tools is a set of design documents written according to set descriptive rules. These documents are the results of the design process, which is part of the existing development process. It thus has the advantage of being easy to introduce into the development workplace.

(2) The tool generates a comprehensive set of test cases, test data [3], and executable script files [4] based on processing patterns and input data variations. This helps to prevent omissions from occurring in manually created test designs and also generates the test data required to execute each test case, making test execution much easier.

These features of the test tool reduce the cost of integration testing, while maintaining software quality

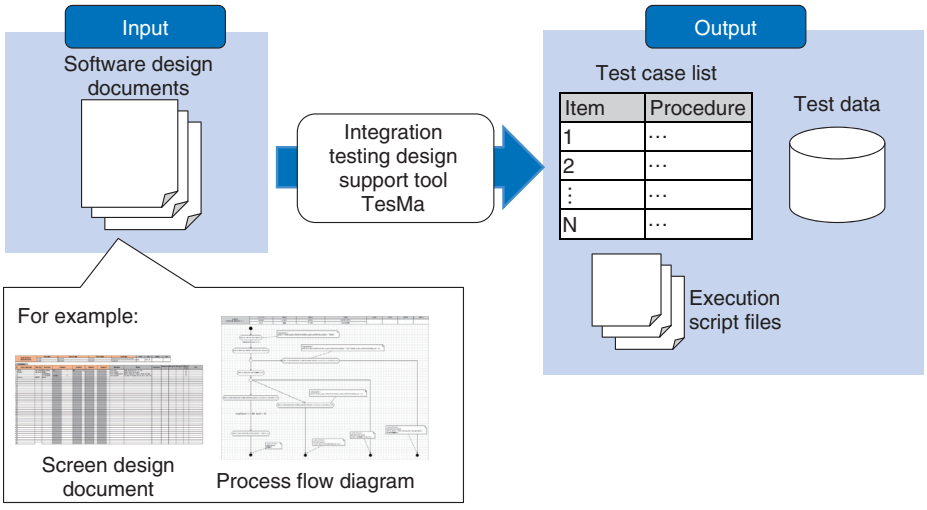


Fig. 5. Integration testing design support tool.

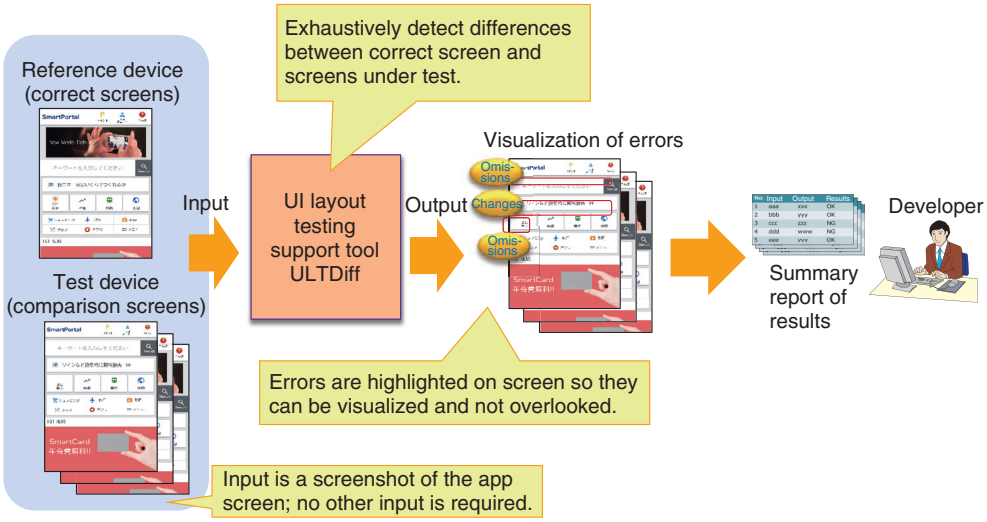


Fig. 6. UI layout test support tool.

through a comprehensive test design.

5. UI layout testing support tool: ULTDiff

Web application testing is done to ensure screens on various types of client devices—smartphones, tablets, and personal computers—display correctly for operating systems (OSs), browsers, and other applications. The test results must also be verified. For example, one might find that screens are displayed correctly on some devices, but buttons have been pushed off screen on other devices. There is an enormous range of terminals available, new model smartphones are constantly being introduced, and OSs are frequently updated. This makes it extremely cumbersome and time-consuming to visually inspect each and every application screen for errors when verifying test results.

The ULTDiff tool addresses these problems, as shown in **Fig. 6**, by automatically detecting missing or displaced screen elements such as buttons. This not only greatly reduces the amount of work required to

visually check for errors across the enormous range and variety of application screens but also catches errors that might otherwise be overlooked, and it improves the quality of applications. The ULTDiff tool has the following features:

(1) ULTDiff reduces the cost of manual detection and prevents omissions by exhaustively detecting the differences between a correct screen and screens under test.
(2) It assists people to effectively decide whether or not each detected difference is an error by highlighting differences on a screen.
(3) It can be applied to many kinds of applications and can be easily introduced to the development process because it only needs screen images as input and does not depend on specific implementation technology.

The combination of these features greatly reduces the amount of work involved in detecting errors and omissions when testing applications under development on a diverse range of client devices and when testing recently released applications on new model devices. Moreover, when revising or adding new functions to applications, ULTDiff can be used for regression testing to make sure the older programming still works with the new changes. ULTDiff significantly reduces the man-hours associated with each release and thus makes it possible to implement rapid release cycles.

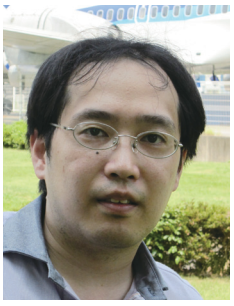
6. Future development

TesMa technology has already been adopted in over 100 software projects by NTT Group companies domestic and foreign, and the tool's ability to maintain excellent quality through comprehensive test design while cutting costs is becoming apparent at NTT development worksites. Meanwhile, ULTDiff has been made available to several NTT Group companies, and we continue to refine the tool based on feedback from the development sites, with plans for a general deployment in the near future.

In the future, we will be less reliant on massive design manuals supporting the waterfall development approach and will move toward test design support based on existing resources such as source code for a wide range of development processes. Without depending on a specific development process, we remain actively involved in research and development that helps all of the development worksites. Building on the QCD gains made so far, we are committed to steadily advancing software research and development in the years ahead.

References

[1] G. Suzuki, A. Kanamaru, T. Iwatsuka, J. Katada, S. Okada, S. Mochida, K. Natsukawa, K. Motohashi, T. Hishiki, T. Kaneko, K. Tanabe, H. Izumoto, M. Sakai, K. Yamashita, and Y. Iwaki, "Improving the Efficiency of Application Development Based on the Macchinetta Framework," NTT Technical Review, Vol. 15, No. 2, 2017. https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201702fa2.html
[2] H. Tanno, X. Zhang, K. Tabata, M. Oinuma, and K. Suguri, "Test Automation Technology to Reduce Development Costs and Maintain Software Quality," NTT Technical Review, Vol. 12, No. 1, 2014. https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201401fa3.html
[3] H. Tanno and X. Zhang "Automatic Test Data Generation Based on Domain Testing," IPSJ SIG Notes, 2014-SE-186, No. 6, pp. 1–8, 2014.
[4] H. Tanno and X. Zhang "Test Script Generation Based on Software Design Documents," IEICE Technical Report, Vol. 115, No. 154, pp. 103–110, 2015.



Haruto Tanno

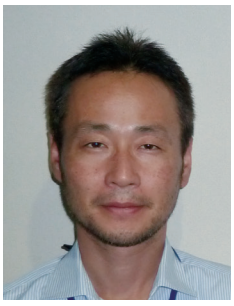
Researcher, Software Engineering Project, NTT Software Innovation Center.

He received a B.E. and M.E. in computer science from The University of Electro-Communications, Tokyo, in 2007 and 2009. He joined NTT in 2009. His interests include programming languages and software testing. He received the Super Creator Award from the Information-technology Promotion Agency in 2008, paper awards from the Information Processing Society of Japan (IPSJ) in 2009, 2013, and 2016, and the President's Award from NTT in 2016. He is a member of 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. His current research interests include software engineering. He is a member of IPSJ.



Katsuyuki Natsukawa

Senior Research Engineer, Supervisor, Software Engineering Project, NTT Software Innovation Center.

He received an M.E. from Nara Institute of Science and Technology in 1996. He joined NTT in 1996. His current research interests include software engineering.