2009 33rd Annual IEEE International Computer Software and Applications Conference

Implementing an Effective Test Automation Framework

Eun Ha Kim, Jong Chae Na, and Seok Moon Ryoo
Advanced Technology Lab
NHN Corporation
Seoul, Korea
{eunha.kim, monster, seokmoon.ryoo}@nhn.com

Abstract—**Testing automation tools enable developers and/or testers to easily automate the entire process of testing in software development. Nevertheless, adopting automated testing is not easy and unsuccessful due to a lack of key information and skills. In order to help solve such problems, we have designed a new framework to support a clear overview of the test design and/or plan for automating tests in distributed environments. Those that are new to testing do not need to delve into complex automation tools or test scripts. This framework allows a programmer or tester to graphically specify the granularity of execution control and aids communication between various stakeholders and software developers. It also enables them to perform the automated Continuous Integration environment. In this paper, we describe details on experiences and usage of the proposed framework.**

Keywords-Test Automation Framework, Automated Testing, Continuous Integration, Fit, FitNesse, STAF, STAX

I. INTRODUCTION

There is a paradigm shift in the software industry. It is being driven by business demands to create larger and more complex software products in less time. How is the paradigm shift changing the way software is produced? The paradigm shift may be characterized by two words: infrastructure and automation. So what does all of this have to do with the developer? What about consistent use of best coding practices by the team [1]?

In this paper, we propose our infrastructure and automation-based approach, together with the set of best practices. We then explain implementation details and a new test automation framework architecture that combines several control structures, which is easier to control the workflow of tests and test environments, as well as other reusable components to interface with external systems. We call this framework NTAF, which is an abbreviation for NHN Test Automation Framework. NTAF has gained acceptance in NHN and is now being presented outside NHN. This paper reports on applications of NTAF for supporting automated testing methods in practical products and details the benefits gained from NTAF used in various products.

II. BACKGROUND

This section introduces various frameworks and the analysis of the optimal framework for the application under test. This also includes the background of NTAF.

There are many frameworks that provide support for automated software testing such as Framework for Integrated Test (Fit), FitNesse, Software Testing Automation Framework (STAF), Selenium, StoryTestIQ (STIQ), Ranorex, Test Automation FX, and Concordion, ranging from decent to horrible and from free to very expensive.

Fit is an open source tool for automated customer tests. FitNesse [2] is a Wiki built on top of the Fit framework which is used for automating acceptance test cases. The basic idea of FitNesse is that tests can be written as tables, much like a spreadsheet. FitNesse works best in a pair environment, though, where one programmer and one business user can work together to define the business rules and write tests for them. FitNesse is most suitable for acceptance test tools to communicate between various stakeholders in the agile environment. FitNesse is excellent as a collaboration tool for communication with customers, but weak as a basis for testing in distributed environments. Yet it does not allow remote execution.

STAF [3] is an open source, multi-platform, multi-language framework designed around the idea of reusable components called services. STAF does allow distributed execution, meaning that tests can be executed from a STAF control machine which are sent to and executed locally on either the client or server machines. The results are returned back to the control machine for reporting. It is designed to be used as pluggable STAF Execution Engine (STAX) for automated test case distribution, execution, monitoring, and results analysis. However, writing a STAX job file in xml/python and understanding the workflow of tests through xml file are not easy.

Selenium and STIQ are software testing frameworks for web applications. Selenium tests can be written as HTML tables or coded in a number of popular programming languages and can be run directly in most modern web browsers. The STIQ tests are written using the FitNesse wiki, with Selenium commands and can be run directly in most modern web browsers [4].

Ranorex and Test Automation FX are Windows GUI test automation frameworks. Ranorex does not have a scripting language of its own and the user can use the functionalities of the programming languages like C#, VB.NET and IronPython. Test Automation FX supports for Visual Studio 2005 and Visual Studio 2008. It is currently only focused on testing Windows GUI and has only limited support for web testing [4].

Concordion is an open source tool for writing automated acceptance tests in Java. It is similar to the Fit Framework but more intuitive. However, it requires an HTML source document and runs using JUnit [4].

So we wondered if we could combine multiple testing tools. To accomplish this, we implemented a new framework called NTAF, built atop a well-known and widely-used tool, FitNesse. NTAF was designed to be used as an extensible framework through a powerful mix of various tools. The workflow of tests and test environments are graphically expressed as tables of input data and expected output data in distributed environments. For example, picking up new builds automatically, installing them on remote machines and executing tests remotely can be accomplished by sending commands via STAF to the applications and reporting the results back in FitNesse.

III. Controlling the Flow of Execution

NTAF has a simple but powerful syntax which controls the flow of execution by using keywords that redirect the path of execution when appropriate. Originally, Fit tables are executed in sequence. However, NTAF is designed to make it significantly easier to control the workflow of tests and test environments.

Tab. I shows control structures which enable to handle the logic and flow of the execution. They support distributed environments so that multiple servers with clients can be configured and tested at the same time. It is very effective in reducing duplicate codes and modules from test applications.

TABLE I. Control structures to handle the logic and flow of the execution

Fixture or Keyword	Action
StafFlowFixture	Controls the workflow of all tests and test environments.
StafCmdFixture	Runs a STAF command.
ITERATE	Iterates a list and runs each iteration in sequence as long as a specified Boolean condition is met.
LOOP	Runs tasks repeatedly as long as a specified Boolean condition is met.
BREAK	Jumps out of the closest enclosing loop or iterate.
CONTINUE	Jumps to the top of the closest enclosing loop or iterate.
PARALLEL	Runs tasks in parallel.
PARALLELITERATE	Runs tasks for each entry in a list in parallel.
IF/ELSEIF/ELSE	Selects a task to perform if a specified Boolean condition is met.
LOG	Logs a message and displays on the result page when an error occurs during program execution.
TIMER	Runs tasks for which time control is provided.

Fig. 1 shows an example of the STAX job for testing web application. The purpose is to run the HTTP service in a multi-thread environment. Fig. 2 is the same test workflow as Fig. 1. Fig. 2 is simpler and clearer to understand than Fig. 1. As shown in Fig. 2, tables are used by developers or testers in writing tests and configuring the workflow of tests. It is also available to customers, management or other stakeholders for running tests and checking the results. NTAF takes advantage of the powerful syntax to be able to develop better tests. Orchestration of complex workflows presented by tables can be easy to understand. It reduces the effort for creating workflows of test cases and increases the visibility of test workflow.

```
<stax>
<script>
  serviceUrl = 'http://www.htmlcodetutorial.com/cgi-bin/mycgi.pl?town='
  varList = [ 'Seoul', 'Busan', 'Dajeon']
 </script>
 <function name="Main">
  <testcase name="'httpTest'">
   <sequence>
    <paralleliterate var="cityName" in="varList">
     <sequence>
      <script>
       httpRequest = 'REQUEST METHOD GET URL %s%s' %
       (serviceUrl, cityName)
      </script>
      <stafcmd>
       <location>'local'</location>
       <service>'HTTP'</service>
       <request>httpRequest</request>
      </stafcmd>
      <if expr="RC != 0">
       <sequence>
        <tcstatus result="'fail'"/>
       </sequence>
       <else>
        <sequence>
         <script>
          if TAFResult['content'].find(cityName) != -1:
          isFind = 1
          else:
          isFind = 0
         </script>
         <if expr="isFind == 1">
          <sequence>
           <tcstatus result="'true'"/>
          </sequence>
          <else>
           <sequence>
            <tcstatus result="'false'"/>
           </sequence>
          </else>
         </if>
        </sequence>
       </else>
      </if>
     </sequence>
    </paralleliterate>
   </sequence>
  </testcase>
 </function>
</stax>
```

Figure 1. Example of the STAX job definition

StafFlowFixture

start_paralleliterate	{var: $cityName$}, {in: Seoul, Busan, Dajeon}

StafCmdFixture				
service	location	request	submit()	response()
http	local	request method get url http://www.htmlcodetutorial.com/cgi-bin/mycgi.pl?town=$cityName$	0	$cityName$

end_paralleliterate

Figure 2. NTAF syntax

IV. Building a CI System

We performed a case study to demonstrate the applicability of our framework in the practical product described in Section V, with the intention of convincing software developers and/or testers that the benefits mitigate the difficulties of manual testing. NTAF is available for performing CI by incorporating the automated build tools such Ant, Maven, Continuum, or CI servers directly. In this paper, we incorporated the Hudson into NTAF due to its usability, wide adoption, and robustness. It could be alternated with other CI servers or build tools that can poll for changes in the version control repository on a specified time interval. The overall architecture is depicted in Fig. 3.

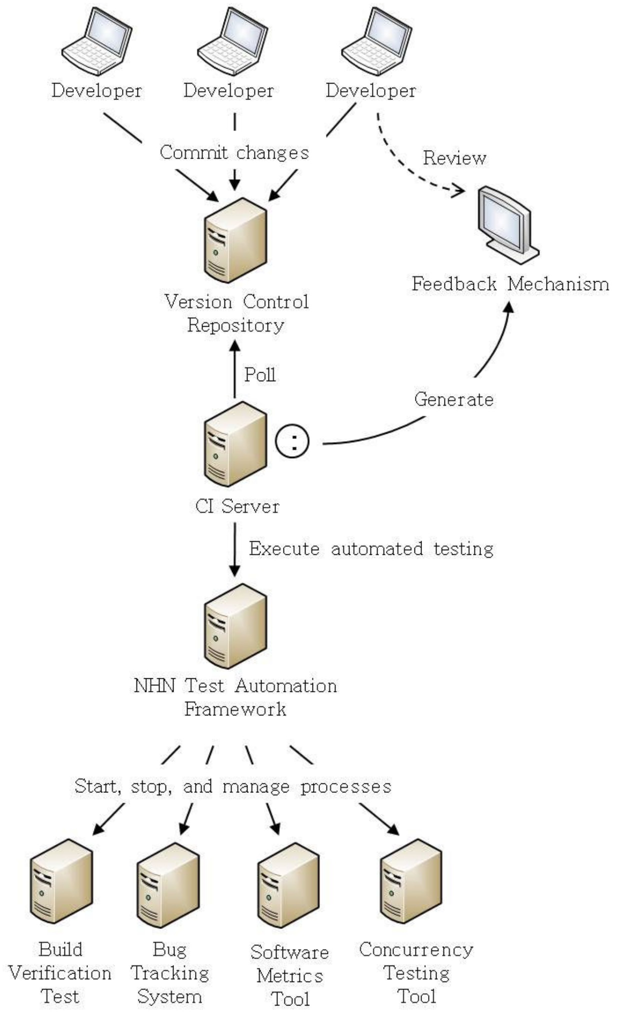


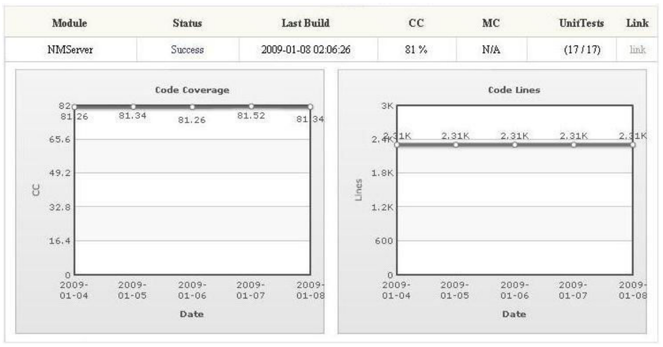
Figure 3. The components of a CI System

A CI scenario starts with the developer committing source code to the repository. The steps in a CI with NTAF scenario will typically look something like this [5]:

- First, a developer commits code to the version control repository. Meanwhile, the CI server on the integration build machine is polling this repository for changes.
- Soon after a commit occurs, the CI server detects that changes have occurred in the version control repository, so the CI server retrieves the latest copy of the code from the repository and then executes a build script, which integrates the software.
- The CI server executes the NTAF, which performs automated testing of the entire software package at each stage of the software development cycle. If necessary, the test is executed under the concurrency testing tool to find concurrent defects.
- The BVT generates a BVT report in HTML format and the metrics tool generates reports for automated statistical analysis to measure quality, examine practices and evaluate risks.
- When errors or exceptions arise in testing actions, NTAF registers them on the BTS automatically and the CI server generates feedback by e-mailing build results to specified project members.
- The CI server continues to poll for changes in the version control repository.

A. *Automated Builds*

A Build Verification Test (BVT) is a suite of tests run on each new build of a product to verify the overall quality of a build. BVT lets developers know right away if there is a serious problem with the build or an unstable build. The BVT service that we developed to interface with the BVT is invoked from the STAF service request through the NTAF to perform a test suite on the local machine or on another remote machine in the distributed environment. A build is considered a success if all the tests in the BVT have passed. If any build that fails the build verification test is rejected, testing continues on the previous build. The BVT generates an HTML report as shown in Fig. 4. This report shows the output on BVT with the build status, relevant code coverage and verification passing.

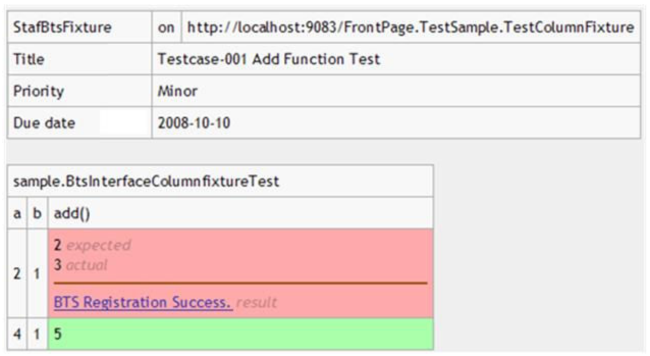


Module	Status	Last Build	CC	MC	UnitTests	Link
NMServer	Success	2009-01-08 02:06:26	81 %	N/A	(17 / 17)	link

Figure 4. BVT report

B. *Automated Bug Tracking*

STAF services are reusable components that provide all the capability in STAF [3]. Therefore, we developed our own service to interface with Bug Tracking System (BTS). After building a test suite, a results page with an unexpected value marked in red is displayed as shown in Fig. 5. When errors or exceptions arise in testing actions, the BTS service registers them on the BTS automatically so that BTS can help developers and testers keep track of reported software bugs in their work.



StafBtsFixture	on	http://localhost:9083/FrontPage.TestSample.TestColumnFixture
Title	Testcase-001 Add Function Test	
Priority	Minor	
Due date	2008-10-10	

sample.BtsInterfaceColumnfixtureTest		
a	b	add()
2	1	2 *expected* 3 *actual* BTS Registration Success. *result*
4	1	5

Figure 5. Automated bug report to the BTS

C. *Automated Testing*

By using NTAF, we can obtain a clear overview of the test design and configure dynamic test scenarios without writing scripts in xml/python or additional coding work. Test cases can easily be defined with a table. This project shows how to use NTAF to write a various types of tests and organize them into hierarchical test suites. It enables faster feedback and leads to quicker defect resolution early in the software development cycle. NTAF defines extensibility points, which are interfaces that allow integration of other plug-ins or legacy systems. Fig. 6 shows the test result trend in Hudson dashboard. Fig. 7 depicts the details of the test result.

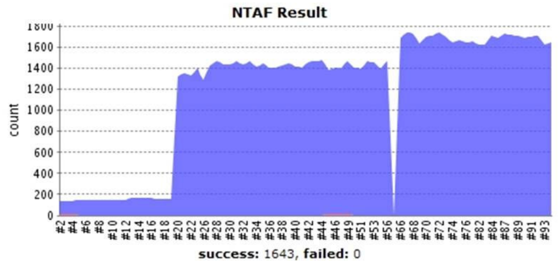
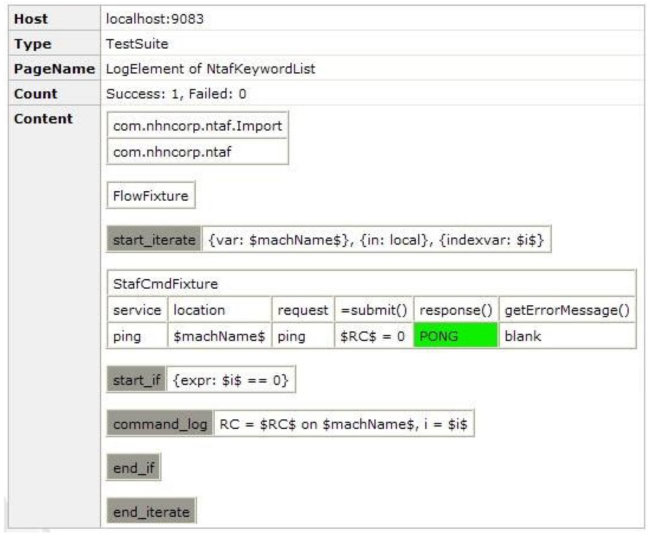


Figure 6. NTAF test result trend



Host	localhost:9083
Type	TestSuite
PageName	LogElement of NtafKeywordList
Count	Success: 1, Failed: 0
Content	com.nhncorp.ntaf.Import com.nhncorp.ntaf

FlowFixture

start_iterate {var: $machName$}, {in: local}, {indexvar: i}

StafCmdFixture					
service	location	request	=submit()	response()	getErrorMessage()
ping	$machName$	ping	RC = 0	PONG	blank

start_if {expr: i == 0}

command_log RC = RC on $machName$, i = i

end_if

end_iterate

Figure 7. An example of NTAF test cases

D. *Automated Metrics Measurement*

NHN Standard Indicator for Quality (N'SIQ) is a standard metric developed by NHN for measuring quality, examining practices and evaluating risks. It embraces many activities, all of which involve some degree of software measurement such as feature completion rate, unit test code coverage, cyclomatic code complexity, violation rate to coding standards, test case run rate, test pass rate, test code coverage, bug close rate, active bugs by priority, security vulnerability, and bug density. Therefore, we incorporated N'SIQ into NTAF for automated statistical analysis and reporting. Fig. 8 shows the cyclomatic code complexity trend of N'SIQ result in Hudson dashboard.

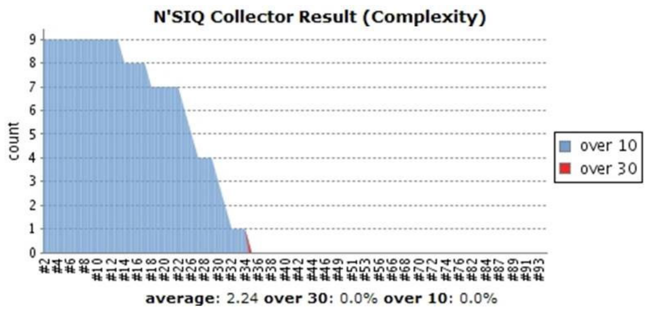


Figure 8. N'SIQ collector result trend

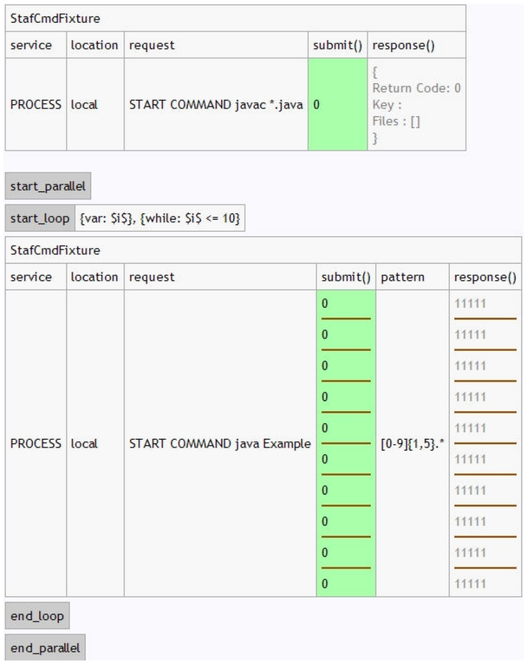
E. *Automated Concurrent Detection*

We started by finding a way to deal with concurrency issues in parallel and distributed software through the incorporation of a concurrent detection tool into NTAF. We used the example in ConTest with the specific bug in mind and expected to get every combination of five digit numbers composed of zeros and ones to be printed (with at least one 1 present) when run under ConTest. ConTest is an advanced testing solution from IBM, whose main use is to expose and eliminate concurrency-related bugs in parallel and distributed software [6].

In this simple example, five threads are created. Each processes a local variable initialized to 1,10,100,1000 and 10000 respectively. The global variable is initialized to 0, and then all the threads are started. When a thread is run, it adds the local variable to the global variable and terminates. The main thread waits for a long time and then prints the global variable [6]. We have run it 10 times by using NTAF keywords, such as PARALLEL and LOOP. When this program is executed it always prints the expected result of 11111 as output as shown in Fig. 9.

However, there is a bug hiding. In Fig. 10, the first step in using ConTest is to instrument the program. NTAF then uses the instrumented instead of the original application. After running the program 10 times, the program prints every combination of five digit numbers composed of zeros and ones to be printed (with at least one 1 present) when run under ConTest. The result shows the concurrency bug of the application which was not discovered in Fig. 9.

To support an effective method for finding concurrent defects in NTAF, we have started to work on integrating with tools for C/C++ such as Intel Thread Checker and Microsoft Chess for windows and Helgrind for Linux.



StafCmdFixture				
service	location	request	submit()	response()
PROCESS	local	START COMMAND javac *.java	0	{ Return Code: 0 Key : Files : [] }

start_parallel

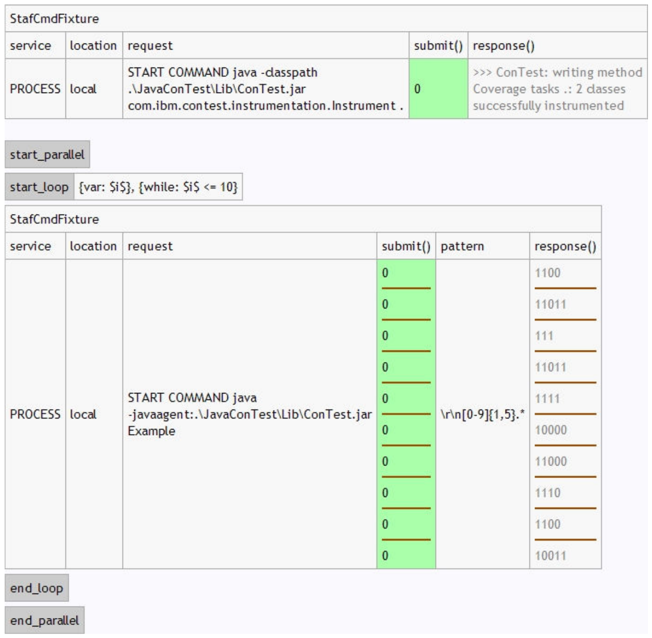
start_loop {var: i}, {while: i <= 10}

StafCmdFixture					
service	location	request	submit()	pattern	response()
PROCESS	local	START COMMAND java Example	0 0 0 0 0 0 0 0 0 0	[0-9]{1,5}.*	11111 11111 11111 11111 11111 11111 11111 11111 11111 11111

end_loop

end_parallel

Figure 9. The result of compilation and execution of processes with NTAF



StafCmdFixture				
service	location	request	submit()	response()
PROCESS	local	START COMMAND java -classpath .\JavaConTest\Lib\ConTest.jar com.ibm.contest.instrumentation.Instrument .	0	>>> ConTest: writing method Coverage tasks .: 2 classes successfully instrumented

start_parallel

start_loop {var: i}, {while: i <= 10}

StafCmdFixture					
service	location	request	submit()	pattern	response()
PROCESS	local	START COMMAND java -javaagent:.\JavaConTest\Lib\ConTest.jar Example	0 0 0 0 0 0 0 0 0 0	\r\n[0-9]{1,5}.*	1100 11011 111 11011 1111 10000 11000 1110 1100 10011

end_loop

end_parallel

Figure 10. The result of instrumentation and execution of processes under ConTest with NTAF

V. APPLICATIONS

We have been using NTAF for testing the practical products, such as database management tool, web application, network library, open API, enterprise service bus system, HTTP service, and XSS (Cross Site Scripting) filtering service at the NHN Corporation. Our case studies indicate that the use of NTAF can aid teams in developing a higher quality product.

We experienced that the use of NTAF can aid communication among various stakeholders, facilitate stress testing which is difficult or impractical to perform manually and support for a better understanding of the progress and code quality throughout the project lifecycle.

VI. CONCLUSION

In this paper, we describe how NTAF can be used to make tests easily. NTAF offers improved visibility and reporting, allowing quick communication among stakeholders and developers. It facilitates performance and stress testing, which is difficult or impractical to perform manually. Moreover, NTAF provides powerful compatibility and extensibility.

Although NTAF requires understanding on how to use control structures of NTAF, how to use wiki markup to make tests, and how to best express requirements in FitNesse, it is easy to learn and use.

Finally, we want to emphasize that NTAF is an open source framework. We recommend you to download NTAF and try it out. You can contact us if you would like to participate in the research process.

ACKNOWLEDGMENT

We thank to Jeongmin Kim for his insights and encouragement; and Hyejin Kim for providing language support when drafting this paper. We would also like to thank to the various development and QA teams in NHN for providing us with practical data.

REFERENCES

[1] Dorota Huizinga, and Adam Kolawa, Automated Defect Prevention : Best Practices in Software Management, Wiley-Interscience, 2007.

[2] Rick Mugridge, and Ward Cunningham, Fit for Developing Software : Framework for Integrated Tests, Prentice Hall, 2005.

[3] “Software Testing Automation Framework (STAF)”, http://staf.sourceforge.net.

[4] Wikipedia, http://en.wikipedia.org/wiki/Acceptance_testing

[5] Paul M. Duvall, Steve Matyas, and Andrew Glover, Continuous Integration, Addison-Wesley, Boston, 2007.

[6] Orit Edelstein, Eitan Farchi, Evgeny Goldin, Yarden Nir, Gil Ratsaby, and Shmuel Ur. Testing multi-threaded java programs. IBM System Journal Special Issue on Software Testing, 41(1), February 2002.