Automatic software validation process

Maciej Dorsz[1], Mariusz Wasielewski[2]

[1] Poznan University of Technology,
60-965 Poznań, Poland
Maciej.Dorsz@cs.put.poznan.pl
[2] Projekty Bankowe Polsoft Sp. z o.o,
60-965 Poznań, Poland
Mariusz.Wasielewski@pbpolsoft.com.pl

Abstract. This article presents the Automatic Software Validation tool (ASV), which is deployed in one of the Polish software companies. This system helps to automatically test web applications, create its simulations, which are helpful during end-user training, and then test those simulations. The tool was invented to speed the process of testing one of the company's applications working in more than 12 Polish financial institutions. The clients' system settings and database schemas are different, therefore while introducing a new system functionality it is not enough to test one system version, but repeat tests for all 12 different parameters settings. Manual testing is very time-consuming and expensive. Every night ASV tool, basing on CVS, ANT and HttpUnit, fully automatically prepares the current system version, deploys it twelve times on Tomcat server with different parameters settings, executes tests, creates application simulations, tests those simulations and sends a summary report.

1 Introduction

Rapid and almost aggressive software development, as can be noticed in the recent years, calls for radical testing effort [1]. Inadequate software testing costs the economy of United States about 59 billion dollars every year. It has been estimated that possible improvements in software testing infrastructure could reduce that cost at about 22 billions [12]. Models and standards related to software development such as CMMI, eXtreme Programming, ISO 9001:2000, RUP place great attention to careful validation of the final product [2,3,4,8]. This article presents the way of putting software testing infrastructure improvements into practice.

About two years ago one of the Polish software company applications was deployed in more than 12 financial institutions. In this article it will be named: AMLPortal (Anti Money Laundering Portal). Although application source codes are the same for all customers, unfortunately, all of the customers have got different parameters settings. Those parameters customizes presentation and business tires according to individual customer's requirements. Moreover, there are some differences in database schemas. The team developing this product prepared Ant script to generate a ready for deployment application [1]. AMLPortal is written in Java, therefore

Ant script simply generates .war file. Then, CruiseControl was installed to take care of storing in CVS repository only versions which can be compiled [5,6]. With time, the problem of software validation process appeared.

AMLPortal is used to search for amount, suspected and related banking transactions. One of its functionality is manual transaction adding to the AMLPortal database. The transaction form has about 45 different fields, such as: transaction number, date, owner data, beneficiary data, addresses, bank account numbers, remarks etc. Almost each of the customers uses unique form to add a bank transaction manually. The form can have additional fields, which may be used by one or some of the customers. Moreover, the clients uses different data validation. Therefore, not fulfilled *beneficiary address* for some clients is correct, for some it is shown as a warning, and for the others it is marked as an error. The example is presented in Figure 1.

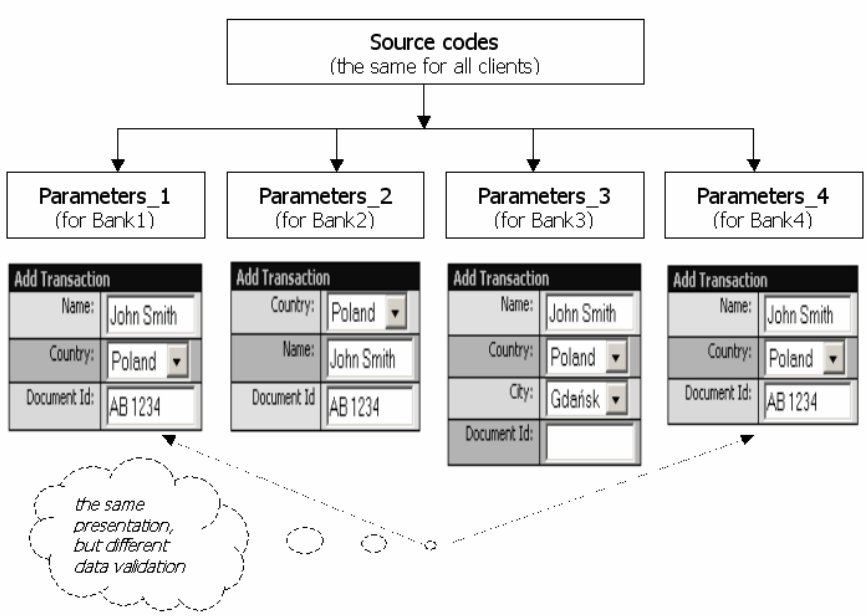

Fig. 1. The example of different presentations and data validations

The difficulty of testing the AMLPortal application will be shown on the example. Let's consider the case that a computer scientist changed a form for manual bank transactions adding. Because the system source codes, JSP pages, libraries, etc. are the same for all customers, this person introduced the change only once. However to carefully test it, one needs to test a new AMLPortal version 12 times, namely, for each set of the client's specific parameters settings. It is very time-consuming, expensive, and monotonous. Therefore, an application for automatic software validation was proposed.

2 Automatic Software Validation

The Automatic Software Validation tool (ASV tool) builds a system version, deploys it on the Tomcat server with client's specific parameters, tests it, deploys the same system version with parameters for next client, test it, etc. and finally send a report. In Figure 2 the diagram outlining the process of automatic software validation is presented.

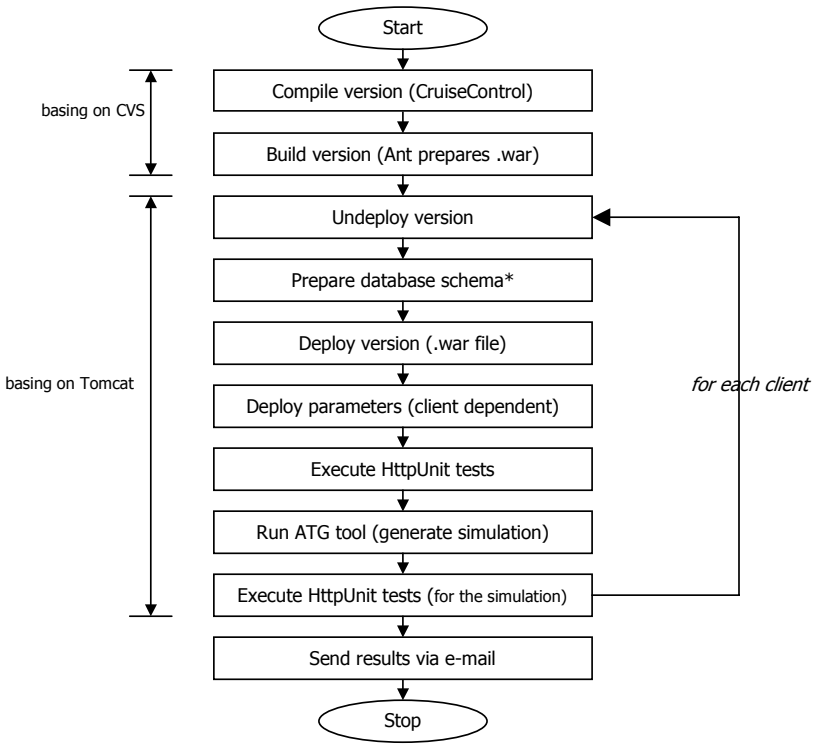

Fig. 2. The diagram outlining the process of automatic software validation

The diagram as well as an asterisk meaning are explained in detail in following sections.

2.1 Version compilation

CruiseControl periodically compiles the *head version* stored in CVS repository. If the compilation process fails, the application development team is obliged to repair the system version or rollback introduced changes. Therefore, in the end of the day, the *head version* can always be compiled.

2.2 Building the version

AML Portal is a web application written in Java. ASV tool uses Ant script to generate the current system version. Ant script, on the basis on the *head version* stored in CVS, prepares .war file.

2.3 Undeploying version

On the Tomcat server, the undeployment process is easy. It is enough, to *stop* the server, delete content of WEB-INF directory as well as the content of WORK directory.

2.4 Preparing database schema

This stage is marked in Figure 1 with an asterisk, because currently is not fully automatic. Application development team has a mirror of each client database schema, for example *bank1_head* database schema resembles the database schema for the *head version* of the application for client *bank1*. Therefore, there are 12 database schemas. A computer scientist willing to change the database schema is obliged to inform a person responsible for database to update all *head* database schemas.

Therefore ASV tool works only with correct database schemas. Moreover, the database schemas names are all time the same, for example *bank1_head, bank2_head* etc. ASV runs SQL commands to prepare a given schema for tests.

2.5 Deploying version

ASV tool unpacks .war file and copies it to Tomcat WEB-INF directory. If ASV worked with more sophisticated application servers, the deployment procedure would be more complex. However, regarding Tomcat server it is really simple.

2.6 Deploying parameters

Every client has individual system parameters. For example *bank1* may have parameter called *is_beneficiary_address_recquired* set to '1' that means it is required. Another one, may have it set to '2' what would mean that it will be marked as *warning.* The others may have it set with value designating 'not required'. For AML portal way of storing parameters is quite sophisticated, for the sake of clarity, it will be assumed that each client has a separate parameter file called: *bank1_AMLPortal.parameters, bank2_AMLPortal.parameters,* etc. In such files are all parameters describing system presentation and business login nuances, but also many other like database connection settings, mailer settings etc.

ASV tool simply copies the right parameters file to Tomcat WEB-INF/etc directory. Then ASV tool *starts* Tomcat server.

2.7 Executing HttpUnit tests

There is only one set of HTTP tests for AMLPortal application [7]. It means that automatic tests do not check client's specific functionality. They are general and focus on system functions which are used by all customers. Therefore, the option prepared and visible only for one client is not tested in this way. However, the main AMLPortal functionality concerns bank transactions management: inserting it into system, searching for amount, suspicious and related transactions, and exporting the founded transaction to an external institution, called: The General Inspector of Financial Information [10].

In order to make HttpUnit tests general the test which adds banking transaction has to add a bank transaction with filled in field *beneficiary address*. Then regardless of the parameter *is_beneficiary_address_recquired* value the bank transaction can be added properly.

2.8 Use ATG tool to generate AMLPortal simulation

AMLPortal was deployed in 12 institutions. It had meant many end-user training. In order to support that process Automatic Training Generation (ATG) tool was invented [4,5]. ATG on the basis on HttpUnit tests saves subsequent .html pages. HttpUnit uses *WebConversation* object to obtain connection with web page. Then it can *set* and *get* html form elements' values, clicks buttons and links. This tool bases on html protocol.

Then it changes their content by adding JavaScript. Finally, Automatic Training Generation tool prepares the simulation of a real application. The simulation is a set of .html pages powered with JavaScript. The end-user may "start" simulation and use it almost as a real system. Because the simulation is a set of static pages, the end-user does not need network connection, running database with an AMLPortal schema and application server.

Automatic Software Validation tool runs Automatic Training Generation tool to prepare AMLPortal simulation.

In case when generation simulations are not needed, processing concerning ATG should be excluded from application testing.

2.9 Executing HttpUnit tests for the simulation

Next, the same tests as were used for testing the real application are used to test generated simulation. Because the simulation visual side resembles the original application, and was created by saving .html files, the fields, links, and button names and their arrangement are the same. In order to test the static simulation pages, they are deployed on application server. Therefore in practice, HttpUnit is testing a web application, which, in fact, is a set of static pages.

2.10 Sending results

After repeating steps from 2.3 to 2.9 twelve times, ASV tool prepares a report and sends it via email. A report structure is shown in Figure 2.

```
-----start: 2006.02.03----
Bank1: passed
Bank2: passed
Bank3: failed
Bank4: passed
Bank5: failed
Bank6: passed
(...)
---------------
Bank3: Executing HTTPUnit tests : AddingTransactionTest
          <exceptions part>

---------------
Bank5: Deploying parameters:
          FileNotFoundException (file: Bank5.parameters)

-----end: 2006.02.03----
```

Fig. 2. The report structure

3 ASV in practice

The Automatic Software Validations tool can be used for the testing of one system version with only one parameters setting. However, it is really profitable for testing a versions with a few sets of parameters.

Table 1. Some of the ASV tool properties

Property name	Value	Comment
server_path	C:\tomcat	# Tomcat installation path
war_path	D:\AMLPortal\war	# generated AMLPortal .war path
properties_path	D:\AMLPortal\properties	# path to clients properties files
simulations_path	D:\AMLPortal\simulations	# path to generated simulations
start_time	02:00	# ASV starts at 2:00 a.m.
clients_list	bank1, bank2…	# list of clients
CVSROOT	:pserver:cod@10.5.5.10: /amlportal	# cvs repository path
(…)		

ASV tool is deployed in one of the Polish companies, whether it will be an Open Source application has not been decided yet. It would be not difficult to adapt ASV to another environments. To use it one needs CVS repository, Ant script to compile and

generate versions and HttpUnit tests. It is not necessary to use CruiseControl, also simulation generation with ATG tool may be used on demand. ASV tool properties are placed in *properties* file, some of them are shown in Table 1.

4 Summary

This article presents Automatic Software Validation tool, which allows one to automatically test the application *head version* with many different parameters settings. Basing on CVS repository, Ant and HttpUnit this tool can automatically prepare system version, then for each client deploy it, execute tests, create an application simulations and even tests those simulations. Finally, ASV sends a report.

The next development phase for ASV tool means the development of GUI side as well as integrating it with CruieControl reports. Automatic preparation of database schemas would be really helpful. Moreover, statistics about automatic testing and its results should be gathered.

References

1. Ant, http://ant.apache.org
2. Beck, K., Extreme Programming Explained. Embrace Change. Addison-Wesley, Boston, (2000)
3. CCTA, Managing Successful Projects with PRINCE 2, The Stationary Office, London (2002)
4. CMMI Product Team, Capability Maturity Product Integrations (CMMI), v1.1, Staged Representation, CMU/SEI-2002-TR-004, Software Engineering Institute, Pittsburgh PA, December (2001)1. Jefferies, R., eXtreme Testing: Why aggressive software development calls for radical testing effort, STQE Magazine, March/April (1999)
5. Concurrent Versions System, http://www.nongnu.org/cvs
6. CruiseControl, http://cruisecontrol.sourceforge.net/
7. HttpUnit, http://HttpUnit.sourceforge.net/
8. International Organization for Standardization, Quality Management Systems – Guidelines for performance improvements, ISO 9004:2000, ISO publication, December (2000)
9. Maciej Dorsz, Jerzy Nawrocki, Anna Demuth: ATG 2.0: the platform for automatic generation of training simulations, Software Engineering: Evolutions and Emergining Technologies, IOS Press, Krzysztof Zieliński, Tomasz Szmuc (ed.) (2005)
10. Ministry of Finance, Poland, http://www.mf.gov.pl
11. Rational Software Corporation, Using Rational Robot (2001)
12. RTI, National Institute of Standards and Technology, The Economic Impacts of Inadequate Infrastructure for Software Testing, Final Report, May (2002)