June 2010

testing experience

The Magazine for Professional Testers

printed in Germany

print version 8,00 €

free digital version

www.testingexperience.com

ISSN 1866-5705

Performance Testing

ONLINE TRAINING

English & German (Foundation)

ISTQB® Certified Tester Foundation Level

ISTQB® Certified Tester
Advanced Level Test Manager

ACCREDITED ISTQB TRAINING PROVIDER

ISTQB International Software Testing Qualifications Board

Díaz Hilterscheid

te testing experience

Editorial

Dear readers,

Load & Performance testing is one of the topics where I don't really have a clue. Well, didn't have!I think that Load & Performance is quite difficult for most people due to the comprehensive knowledge needed to be a good professional.
You will find here some nice articles about it. I hope you like it and enjoy reading.

I was at the Iqnite Conference in Düsseldorf recently, which we media sponsored.
It was good to see old and new faces and to discuss the magazine with some readers.
I'm really happy about the wonderful feedback that I received and thank you again for the congratulations.
We try to do our best.
We agreed that the last issue was extremely good, and there are no old or new competitors to be afraid of.

© Hans Schaefer

A few weeks ago, many of you were afraid about the volcano Eyjafjalajokull. Some well known professionals were stuck fast for some days without being able to go back home or attend conferences.
My friend Hans Schaeffer was in Iceland recently and made some great pictures of the volcano. Have a look on http://jalbum.net/browse/user/album/597403/

Amazing! Hans is not only a great professional and one of the best speakers I know, but also a person who loves the world and nature. I'm not sure whether he went to Iceland because he thought that the volcano was a steam locomotive... possible, because he is also a passionate steam locomotive driver!

On June 7-8 we will have the Testing & Finance in Bad Homburg. I hope to see you there. It looks like we will have a full house.

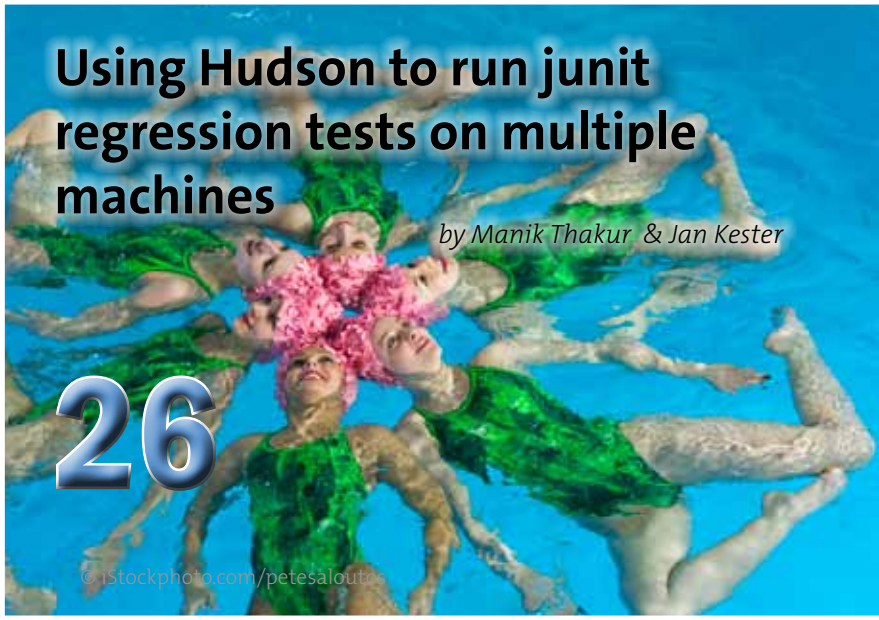
Last but not least, my son had his 7th birthday! I'm getting older and this is good! It is amazing to talk to and with him, and it's very difficult to explain to him what my company does! Childrens' minds work different. Nice.

When I turned seven, my father passed away. My father was almost my age. The same age difference between son and father. He was young like I'm now. I still miss him.

Enjoy reading!

José Díaz

Contents



Moving Globalization Test Quality upstream using Pseudo Translation Approach

by Anuj Magazine

With more and more software organizations realizing the importance of global markets, there has of late been immense focus on software globalization engineering. In order for the software product or application to be appropriately globalized, it should precisely involve software globalization in its design. Introducing software globalization in the design is usually not as straight-forward as it may seem. There are many factors including supporting character sets from the languages all over the world, making the installer multilingual, making applications locale-aware, actual translation of User Interface (UI) elements, taking care of many cultural considerations such as handling of date/time, numeric formats, currencies and a myriad of other things that change from one location to another. All in all, it's a complex activity and thus software globalization testing also assumes greater importance. Software globalization testing primarily comprises two different parts: internationalization testing and localization testing.

Internationalization is planning and implementing products and services so that they can easily be localized for specific languages and cultures. So for any product to be successfully localized in different languages, it is imperative that it has to be internationalized properly. Internationalization is abbreviated as I18n (where "18" represents the number of characters between the letters "I" and "n" in the word "Internationalization") .

Localization is the aspect of development and testing relating to the translation of software and its presentation to the end user. Localization is abbreviated as L10n.

Accordingly, the prime charter of I18n testing is to test the changes introduced in the product as a result of Internationalization activity. And similarly, L10n testing is done to test the changes introduced in the product as result of localizing the product.

With this background in mind, let me state the focus area of this article. This article primarily deals with one of the unique techniques called "Pseudo Translation" that helps find the globalization-specific bugs even before the actual translation of products has taken place. In doing so, it helps to move the quality upstream, reduce translation costs and, most importantly, helps the organization to achieve "Simship" (i.e. simultaneously shipping the product in all the supported languages).

Before we appreciate what pseudo translation is all about, I would like to explain some of the common issues found during globalization testing.

Some common bugs found in globalization testing:

Bug type	Bug description
Hard coding	Externalizing the strings from source code is an important I18n activity. If not done properly, it results in English language strings even in a product localized in any language, e.g. German etc.
Locale unawareness	Making the product locale-aware is again an I18n activity. Being locale-aware means that the product is able to recognize the underlying OS locale settings, such as language, date/time format, numeric/monetary formats etc. Locale- unaware products usually tend to display only one format relating to date/time, numeric etc., irrespective of the language run.
Character corruption - Question marks	Character corruption is usually signified by the incorrect display of localized characters, usually seen as question marks (???) on the UI. This is typically caused by incorrect Unicode to ANSI conversion.
Character corruption - Random characters	Another form of character corruption is related to the display of random high ANSI characters (⅓, †, ‰) on the UI. This is typically caused by wrong code pages being referred to.
Character corruption - Font display	Yet another form of character corruption is related to incorrect display of font. Presence of vertical bars, boxes and tildes (|, □ , ~) indicates that the selected font cannot display some of the characters.
UI sizing issues	An example of a UI sizing issue is: Let's say that after the product is localized into a non-English language, say Japanese, the UI gets distorted in a way that corresponding lines of text are overlapping each other, leaving the text not fully readable. These types of issues are usually caused by planning insufficient space in the UI for the localized text.
Text truncation	Text truncation is a type of issue which results in either the leading or trailing, horizontal or vertical parts of text being curtailed, resulting in the UI looking unprofessional.

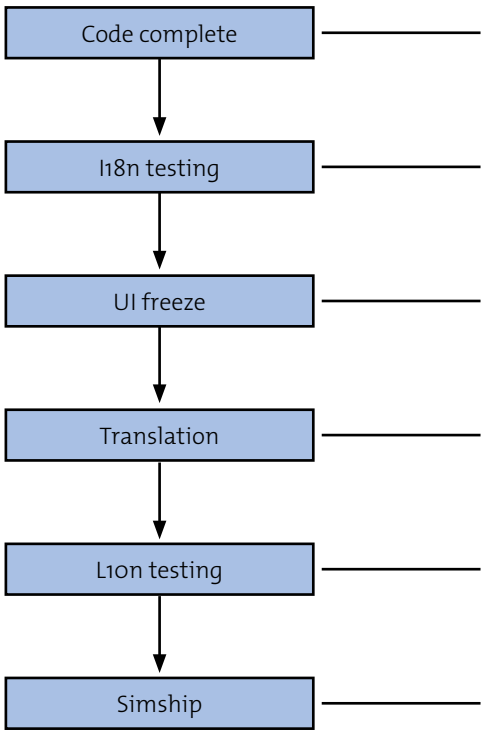
Bug type	Bug description
String concatenation	In the English version of a product, it might be entirely acceptable if one sees a combination of text such as "Welcome, John". In some countries the name is expected to be the first in the subject, e.g. in Japan. Thus, the usage of string concatenation functions in the code leaves it vulnerable.
Over-translation	Not everything present in the UI is a candidate for translation, e.g. product names, file/folder names etc. If by accident these are translated, it may cause issues.
Feature failures	The issues in this category are of quite serious nature. These would be cases where product features fail to function on localized operating systems, set-up or when using localized characters.

What is Pseudo Translation ?

In the simpler terms, pseudo translation is an approach, the essence of which lies in simulating the product translation much before the actual translation of the product has taken place. Now the question arises of what benefits would anyone reap by simulating the translation. Before even attempting to answer this question, let's take a look at a typical project life cycle involving a product in multiple languages.

A typical globalization project life cycle:

In this section, I would like to briefly update on project milestones that are there in a typical project life cycle involving a product to be released in multiple languages:



From the software globalization point of view, this indicates that all changes in the code that are needed to internationalize the software (like Unicode support, multilingual installer, locale awareness, externalization etc). have been completed.

I18n Testing is done after the "Code complete" stage. If the product is never localized, this type of testing is done using English (assuming English as base language) builds on the localized setup (OS/Browsers etc.). Typically bugs specific to I18n features (such as Unicode implementation, feature failures etc.) are found.

UI Freeze is a critical milestone for software globalization. Unless the English language UI is stable and not undergoing drastic changes, the actual translation should ideally not happen.

This phase follows UI freeze and is marked by the English resource files being sent over to language experts for translation and, once received, these are to be included as a part of the localized builds to be further tested.

L10n testing is done on the product after it has actually been translated. This phase is usually characterized by a huge surge in the number of bugs, which are typically related to text truncation, hard coding, concatenation, UI sizing, character corruption etc.

Simship means the simultaneous shipping the software product to all the targeted markets. In other words, the product is released for European, Japanese, Chinese etc. markets on the same day as for US, UK and other English speaking countries.

So, where does pseudo translation fit in the globalization software life cycle?

Pseudo translation by definition is simulating translation. So, this makes it obvious that pseudo translation will be done much before the actual translation takes place. Pseudo translation is done usually after the "globalization code complete" stage has been achieved, which should ideally coincide with the start of the I18n testing phase. Instead of taking the English builds for testing, the test engineers take the pseudo-translated builds. Before, I delve into how testing using pseudo-translated builds is done, let's look into how the pseudo translation is actually done and what the pseudo-translated product looks like.

How is pseudo translation done?

Simply put, to pseudo-translate a string element on the UI, the following is done -

1. Take a string element e.g. a label in the UI.
2. Append the string with leading characters (in localized language) and some trailing characters.
3. Replace some characters, especially vowels, in the string with localized characters.
4. Ensure that entire string expands by at least 30%.
5. Include starting and ending delimiters.

Let's take an example and follow the above steps-

1. Let's say the string in question is "**Pseudo Translate me**", and say this string needs to be pseudo- translated into German language.
2. As a first step, this string is appended with leading and trailing characters in German. So the original string would now become-**öüß Pseudo Translate meßüö**
3. Change the vowels in the string with localized characters-**öüß Pseüdö Tränsläte meßüö**.

4. Include the delimiters as leading and trailing square parenthesis "[", "]"-

   [öüß Pseüdö Tränsläte meßüö]

Usually the activity of Externalizing the strings from the source code is completed, before the pseudo-translation process is performed. Externalizing is an important activity that helps to separate the translatable elements from the source code into separate files called resource files. After the strings have been externalized into resource files, the resource files are sent to language experts for translation into the native language. These resource files also undergo pseudo translation using internal tools (to perform the above steps) or proprietary tools like Alchemy Catalyst etc.

What does a pseudo-translated screen look like ?

Below is an example -

Image source: http://www.globalvis.com/news/pseudo_after_noresizing.jpg

The essence of pseudo translation:

Before delving into the real benefits of pseudo translation, let me first explain what happens when the text is actually translated and shown on the UI. Whenever a text is translated into a non-English language, invariably the amount of space it occupies on the UI increases (as e.g. a German or Japanese character occupies more space than the English character), and the entire text expands horizontally and vertically. Based on past experience and facts, it is an accepted fact that incorporating 30% text expansion will help to solve most of the issues related to text expansion. Also, when the product gets localized, the majority of bugs are related to UI sizing, text truncation, hard-coded strings, character corruption, and few to over- translation, string concatenation etc. Most of these bugs are found "only" after the product is "actually" translated, which invariably happens towards the close-down in the product life cycle, i.e. in the L10n testing phase. This is where pseudo translation can add immense value, because it helps simulate the conditions related to all these bugs in earlier in the I18n testing phase. So by the time we reach L10n testing phase, most of the bugs have been found and flushed out#corrected?.

How does a test engineer test pseudo-translated builds?

- By its very nature, pseudo translation simulates translation. So when a tester inspects the pseudo translated screen, he/she can still read the actual English text. Looking at our previous example, the original text was "**Pseudo Translate me**". After pseudo translation, it looked like this -

  [öüß Pseüdö Tränsläte meßüö]

  So, by looking at the UI a test engineer can easily recognize the text highlighted in yellow, as it still resembles the original text.

- Think of a condition that either the leading or trailing delimiter is missing with some padded text, as in the string below -

  [öüß Pseüdö Tränsläte m

  Here, the trailing padding as well as some of the original text is missing. It is an indication that the appropriate space has not been designed in the UI for possible string expansion, and this is probably a text truncation.

- Another situation would be that after the string has been pseudo-translated, it appears in the UI as follows -

  [??? Pse??d?? Tränsl??te me???]

  Such a string is an example of character corrpution and is indicative that the rendering of strings has not happened as expected.

- Suppose that after pseudo translation has taken place, the resultant text still shows up as the original string, i.e. **Pseudo Translate me**. This is an indication of a hard-coded string, i.e. the string was not externalized to resource files and is still embedded in the source code.

- Another instance would for example be a product name (product names are usually not translated). If it is shown as a pseudo-translated string, it is indicative of over-translation and hence should be taken out of the resource files.

The beauty which is shown in the above examples is that pseudo translation helps uncover most of the common bugs related to globalization, and it does so even before a single penny is spent on the actual translation of the UI. This results in huge cost savings for the organization.

Limitations of pseudo translation:

Pseudo translation, as you may appreciate by now, is quite a valuable approach for helping to find globalization-specific bugs early in the life cycle. However, the real utility of this approach is generally for the products that are being translated for the first time, or are generally undergoing extensive UI changes which require a lot of translation. Suppose that you used Pseudo translation for version 4.0 of your products and reaped a lot of benefits and that you are now planning version 4.5 testing, you may realize that the UI is going to be largely the same and that there may also not be extensive I18n code changes. In such a situation, pseudo translation may add very little or no value. So in some cases pseudo translation would not be considered, as most of the translation is already in place and won't change across different versions.

Epilog:

Moving quality upstream for software products has been a buzzword for quite a while. The pseudo translation technique, when applied correctly, certainly goes a long way in improving quality. There are some very strong benefits in helping to unveil bugs, such as text truncation, character corruption, hard coding etc., which traditionally have been found only after the base product was translated. Finding these bugs after the products has been translated is a costly affair, mainly because translation is usually an outsourced activity and the cost of outsourcing increases with each additional change request. With pseudo translation in place, most of these bugs are simulated and found early on. Apart from cost savings, one of the other tangible benefits for the organization is to achieve "simship".

Biography

Anuj Magazine is a software testing practitioner currently working at Citrix R&D India Pvt. Ltd. He has been a regular speaker at conferences, such as QAI, STEPin forum and has spoken on several software testing topics. He has written articles in www.stickyminds.com (TheSmartTechie Magazine) and writes frequently in his blog- http://anujmagazine.blogspot.com . He is an MS in Quality Management and holds CSTE and CSQA certifications.

Mobile Testing - A Toolkit for Advanced Testing - The Techie, The Simple and The Obvious

by Dario Uriel Estrin & Bernard Shai Lelchuk

The field of mobile[1] testing poses unique challenges to the advanced software tester given its diversity and fast change pace.

It is a world of many devices, ever changing operating systems, APIs, software components and screen resolutions as well as evolving customer demands, device designs and mobile network carriers.

These characteristics of the Mobile software world aggravate the regular challenges faced by software testers. Testing has to be performed in a small window of time - between the tight and much abused deadlines of the coders and the short planned time to market of the marketing department. In order to cope with the high levels of uncertainty and to perform our work with high certainty (covering the important things) and deliver reproducible results, we must establish a stable testing platform with which to navigate through the stormy waters of new mobile platforms, application releases and tight schedules

While testing world-leading mobile applications in our work places, our teams have developed over the years (since 2000) various techniques which allow us to cope with short release timelines, while only minimally compromising the continuous quality improvement for which the our products are known. It is important to remember that ‚testing technology' isn't just about the hardware/software technology used, but also about the techniques of the testing, i.e. the methodology and the way things are done.

In this article we aim to present a range of useful techniques (though not a comprehensive one) we have gathered over the years in our SQA teams. We hope to offer a simple toolkit that will provide a jumpstart to making the life of mobile testers easier and help focus on the important things - the "bugs' hideouts" - which lay behind the platforms' built-in impediments.

The Techie - Pure technique tools

Simulating real-world scenarios by means of configurable data manipulation.

The communication protocols connecting clients with servers are routed to a proxy mediator in order to simulate various server responses to the mobile device.

A growing portion of mobile applications aims to provide information to customers. Two kinds of information are usually provided: static information hard-coded into the client or online updated information. [2]The backend systems provide updated information upon request. This information changes as a function of various parameters – reality-related and user- related. Testers can set the user-related parameters, check the reality- related ones, and see whether the resulting information/application behavior is correct. However, such methodology impedes them from testing all possible results as reality is reality, i.e. chaotic and unpredictable. For instance, testing an instant messaging mobile client will require testing interactions with other users, server responses, network problems, etc. The best way we came up with to cover the system-centric scenarios was 'controlling the world', i.e. simulating server responses according to real- world input/triggers/responses. Simulating a server's response for a given request (response which is for itself a function of responses provided by another system) allows us to isolate the mobile client behavior from real- world input and to focus on inter-system testing.

In our case the response to the mobile application was sent via a comma- delimited data stream from one of the servers. The testing team learned how the stream looks like and behaves. Later, we built a testing server environment which generates responses according to variables set by the tester (this could be done by manually manipulating a text file with values or getting an HTML page to pick values from and which creates the file for the tester). This resulted in a valid comma-delimited response that was fed to the mobile client to be tested. This allowed us full control of the data that in its turn triggered the various logical paths of the application. Being able to easily simulate any possible scenario helped discovering flaws in the system, which may not have been discovered, if we had to count on the uncontrolled real-world data that is the input to the servers and application.

Technically, we routed the device to a 'server' that was getting the status from our testing environment:

1. We configured the device for data connections via a PC machine or used a device simulator on a PC machine.[3] This allowed us to control the routing path of data networking.
2. We manipulated the address of the servers which the client was looking for and routed it to the testing environment described above. This was done in the Windows ‚hosts' file (usually located in the ‚C:\WINDOWS\system32\drivers\etc'

1 Phones, PDAs and Smartphones

2 Static information on client or server consists of two types of data. One which is totally static and never changes (e.g. the number of players in a soccer team), and the other which is rarely updated by an external source (e.g. the number of teams in the soccer league).

3 In which case the connections are obviously routed via the PC too.

folder). Thus, every time an application tried to communicate with a remote server, it was actually directed to the test environment in which we could control the answers it was getting.

3. On the test server we created an index.php file, which redirected all communications to the path where the output of our 'real-world creation machine' was. After client data was redirected to our testing environment, it passed through a set of PHP filters & parsers (see sample below), which separated data parameters for manipulation – and which returned answers according to the tester-configured 'real- world' answers.

index.php Redirection Sample:

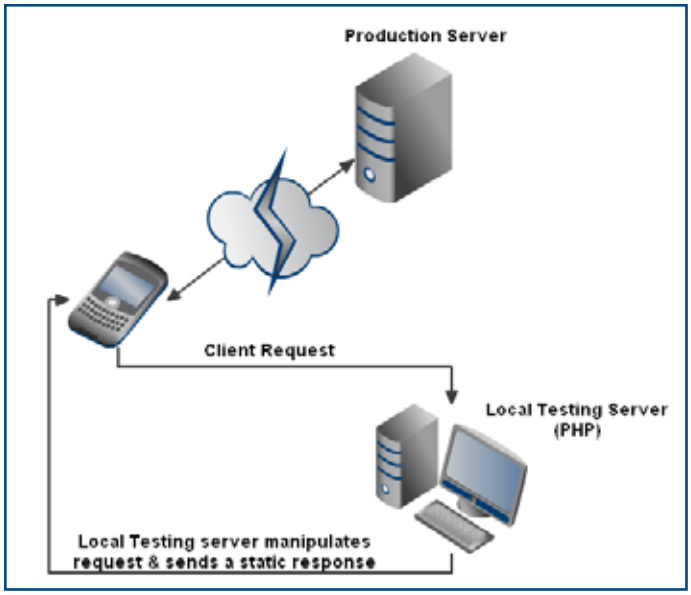
```
<?php
    header( ,Location: /location/data/ResponseGe-
    nerator.php' ) ;
?>
```

The index.php routed the device to the test environment for data manipulation and returning of the tester-controlled 'real-world' mimicking data.

Using the above data-routing scheme and 'real-world machine', allowed the testing team to configure the test environment based on testing needs and enabled to cover a wider range of 'real-world' situations. This increased both the test coverage & product maturity.[4]

4 Thanks to Mr. Uri Mor for his contribution to this section of the article, for the PHP code example and for his professional partnership.

```
#Handle submitted forms:
if(isset($_POST["baseString"])){
$string=$_POST["baseString"];
$i=0;
while(isset($_POST["value$i"])){
$string.=$_POST["value$i"];
if(isset($_POST["value".($i+1)]))
$string.="|";
$i++;
}
$string.='~';

$handler = fopen("data\DataQuery.$Content$Date.$Number","w");
fwrite($handler,$string);
fclose($handler);

#if the original Response was updated
elseif(isset($_POST["original"])){
$handler = fopen($_POST["orgFile"],"w");
fwrite($handler,$_POST["original"]);
fclose($handler);

$message = "The fields were updated with the original response";
}

else{
$orgFile='originalResponse.txt';
$testerName="default";
$message = "";
}

#Dislpay the forms:

### Short Format
echo "<H2>Short Format</H2>";
$lines = file('queryBody.txt');
$i=0;
echo "<FORM action=# method='POST'>";
echo "<INPUT type='hidden' name='baseString' value=$newString>";
foreach ($lines as $line){
if(!in_array($i,$shortArray)){
echo '<INPUT name=value'.$i.' type="hidden" value="'.$originalBody[$i].'">'."
\n";

}
$i++;
}
echo "<BR><INPUT type='submit' Value='Save'>";
echo "</FORM>";

### Long format
echo "<H2>Long Format</H2>";
$lines = file('queryBody.txt');
$i=0;
echo "<FORM action=# method='POST'>";

echo "</FORM>";
```



Code example:

- End-to-end functionality via data integrity testing.

There are two major cases in which we would like to trace the information flow through its whole life cycle.

In the first testing phases of a new product, we would like to confirm the data integrity in order to confirm that the basic product infrastructures work before moving forward to testing the product at a higher level. Later on, during the testing, when encountering unexplained problems, we would also like to trace the data from its inception (e.g. input by user) to the processor (e.g. server) and back to the client. In both cases, we learned to appreciate the usefulness of various "sniffing" and data interception tools.

In order to confirm that the basic product's infrastructure works correctly, we used a simple, yet straight-forward sniffing tool (e.g. 'WireShark') [5]. We verified the data integrity on the server, while it was being passed to the client, on the client and vice versa. The same was performed for authentication requests and their respective responses passed between the clients and the servers.

At a later stage, when encountering a problem, we monitored all the incoming/outgoing client/server requests/responses, in order to trace the complete information flow.

The testing was divided into small functionality & data validity chunks:

- Initiate a client request
  - Client sends to server initial values for authentication such as product identifiers and authentication data.
- Verify server's initial response
  - Server validates client's initial authentication request & returns respective response:
    - Authentication failure: Error code & message returns to client & connection ends.
    - Authentication success: Positive response (e.g. OK) returns to client & connection continues.
- Verify client additional requests
  - Client now sends additional requests to the server - either together or one at a time
- Verify server's final response
  - Server returns a response to the client per each sent request.

Certain modules of the systems communicated via XML for easier cross- platform coding. Remember we are talking about the mobile world, where there is a plethora of different platforms. This leads us to a deeper level of data validation tests, which include thorough XML testing of client/server responses upon various requests & data changes.

As data changes can originate either from a client or a server, root-cause tracing for problems is done by intercepting the communication between them. The intercepted data is later analyzed by the testing team at the data integrity level as well as the XML validity. Mobile applications create a lot of data traffic, which presents an additional challenge – the tester must know the application well enough in order to know where to look for the problem and not spend precious time trying to sort out were exactly in the data stream he should be looking. On the other hand, when a new feature or communication-producing feature is added, the best thing to ensure a bullet-proof infrastructure is to intrusively and thoroughly test the added data exchange and not only a small chunk of it.

For more advanced data sniffing and interception, we used tools such as BurpSuite.[6] Its set of tools gave us extended flexibility, such as advanced data scanning and filtering, interception, inspection and modification capabilities of all the traffic passing in both directions from client to server and vice versa. These tools proved to be useful as well in security testing, such as penetration, SQL injection and various other vulnerability attacks.

The 'Simple' – Some simpler tools

As the iPhone rocked the market, *mobile web* applications finally became part of the mainstream mobile applications market. In our quest for tools that will support our functionally testing of mobile web applications, we ran into Firefox's extensions. A combination of these enabled us to simulate mobile client browsers, allow faster testing of mobile web applications and easier debugging on a PC, while saving a substantial amount of time and effort.

Using FireFox add-ons, such as Live HTTP Headers, XHTML Mobile Profile & Modify Headers, we were able to simulate mobile client browsers of any type (especially clients we do not have access to). Testing using these tools proved to be time efficient, bugs were discovered at early stages and fixed on-the-go, thus test coverage was greater than we would have achieved with regular testing.

To improve the development process, we've introduced these tools to the Development, Graphics Designers and Web Masters teams which espoused it. Thus we gained a shift in product quality and maturity due to preliminary testing (or unit testing) by the teams using these productive tools. By doing so, we managed to release products earlier than scheduled with higher maturity and coverage levels.

One of the biggest pains of mobile testers is the number of deliverables to be tested according to various platforms, sales channels, customizations, etc. Externalizing relevant parameters allows focusing on testing this externalization module once and then later on being able to focus on the core functionalities of the application without having to worry about the multiple release packages. Examples of such externalized parameters could be versions identifiers, paths of graphic images or even names to be embedded into the applications.

The externalized parameters are placed in a file nearby the installation file. The latter takes the relevant information on the fly, thus eventually customizing the application to be installed. Once this mechanism is proof-tested, it spares the need to test multiple installers. Moreover, these parameters can at any time be manually changed and tested by non R&D personnel (e.g. sales and production). On many mobile platforms, the installation-reboot-test-uninstall-reboot process is known to be extremely time consuming, which emphasizes the advantage of this technique.

5 http://www.wireshark.org

6 http://www.portswigger.net

Feierabend! Seine Last- und Performance-tests laufen mit TAPE automatisch weiter

Am nächsten Morgen beginnt er mit der Analyse

MEHR Zeit zum Testen. Mit TAPE – Testing Automation for Performance Engineering – lassen sich Last- und Performancetests effektiver als bisher durchführen. Zentral und ganzheitlich. Kostbare Zeit ist aber nicht alles, was Sie an MEHR mit TAPE gewinnen:

- MEHR Toolintegration
- MEHR Überwachung der Testaktivitäten
- MEHR automatisierte Testschritte auf verteilten Testrechnern
- MEHR Analyse der Performance-Anforderungen

Wenn Sie MEHR über TAPE wissen möchten: www.tape-loadtest.com. Gerne informieren wir Sie auch persönlich.

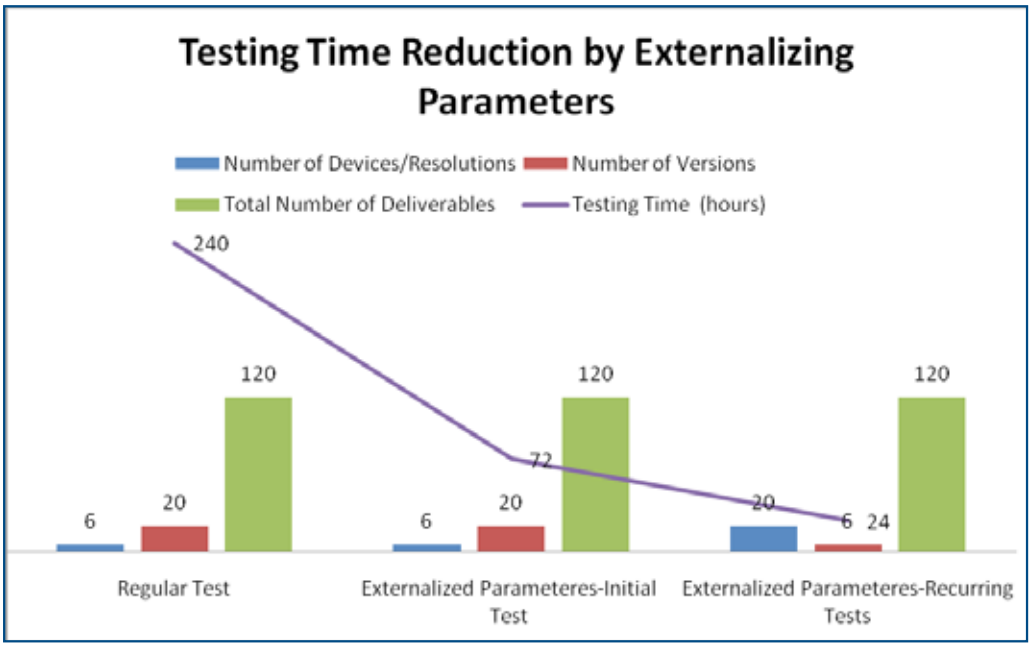
Kontakt: Daniela Fichera · Kapuzinerstr. 7a · 80337 München · Tel: +49(0)89 589399-0 · daniela.fichera@c1-setcon.de · www.c1-setcon.de

To illustrate this, let's take a look at the table below. A company has 6 devices/resolutions and each sanity test is estimated to take 2 hours. Now, the company currently works with 20 different versions. If you multiply the number of devices by the number of versions, this gives a total of 120 deliverables which in turn translates to 240 hours of testing!

However, by externalizing the versions' parameters, so that these can be thoroughly tested once, we reduced the initial testing duration to ~72 hours.

Now, after verifying that this new process works correctly and with no failures, future tests will be estimated at only ~24 hours – which amounts to 10% of the regular testing time. This significantly lower percentage is mainly due to the ease of testing the externalized parameters and the need to sanity-test only a few of the deliverables for validity in the next versions, rather than testing all deliverables over and over again.

Deliverables	Regular Test	Externalized Parameters-Initial Test	Externalized Parameters-Recurring Tests
Number of Devices/Resolutions	6	6	20
Number of Versions	20	20	6
Total Number of Deliverables	120	120	120
Testing Time (hours)	**240**	**72**	**24**



Testing big chunks of new content. To test new content, it is always faster to use desktop PC simulators and regular web browser to test content included in mobile apps and mobile web apps vis-à-vis databases. The content doesn't change and while we do not care about UI and on device functionality, we can definitely trust the PC tools for pure data testing, e.g. calculations done by a mobile/mobile web calculator.

The 'Obvious' – Can't do without methodology

Every combination of application and mobile platform has its own special common problem. A mobile testing team should map the known and recurring issues and create threshold test cases for them. These should be run on every version delivered by the developers. Embedding these test cases will result in virtually no major issues with devices' compatibility and flawlessly passing criteria tests (AppStore testing, Symbian Signed, etc). It will also put more focus on testing new functionality and delivering high quality instead of wallowing in the same bugs.

Things you should look out for:

- Known issues with your application per platform
- Known issues on the tested platform (application independent)
- Critical issues in regards to platform style guides and device interoperability

Do not re-invent the wheel. Use manufacturers/OS test criteria as a base for creating a threshold acceptance test that is briefly run on the application before any major testing cycle. Combine this with the previous point (your application-specific issues) and you have a winning approach..

Test texts and graphics only in a second round after they were approved by stakeholders. Mobile applications often change graphic layout and strings (because of localization or marketing needs). The need to test on various devices and resolutions significantly slows down the testing phase and subsequently the quality of mobile testing. In order to reduce the number of iterations, and thus saving time and improving quality, the relevant 'owners' should approve each change before testing. This also helps bringing external functions into the testing process and makes them partners in the testing teams' quest for quality, deliverability and time to market.

The previous point leads us to the section required when approaching an application for testing graphics, strings and localizations. Test planning and execution should be separated from logic and functionality testing. Only after logic and functionality have passed the initial set of tests, should the focus move towards the visual aspects of the application. Otherwise, we run the risk of finding basic functional flaws at the last moment. As 'visual' testing requires a longer testing time and usually results in higher modification rates, it should be tested following these two preconditions:

1. Specifically approved by stakeholders (see previous point)
2. Logic and functionality have reached an acceptable level of maturity.

Our experience shows that failing to adhere to these rules results in quality and timelines being sacrificed on the altar of testing mere graphic aspects.

Another good practice for localization testing, which proved to decrease our testing times, is the 'Dual-Language'/'Dual-Resolution' testing. Our experience shows that it allowed for early identification of root problems which is critical when testing multiple languages towards a release.

1. Pick the two most 'problematic' languages for your application. These are commonly the one with longest strings and the one more technically problematic (e.g. double-byte, diacritic punctuation or middle eastern).
2. Run the test cases on these languages on two identical de-

vices/resolutions.

3. Repeat the test cases for the 'longest' language on different resolutions.

   A great tool: a multi-resolution device that can flip between two resolutions, which allows covering 2 test cases in a single installation.

4. After the first phase described in steps 1-3, you can move on to test each language on two resolutions at the same time. As strings trimming behaves differently on various resolutions, placing 2 identical applications and languages with different resolutions results in an immediate and precise string comparison, which enables pinpointing incorrect text handling on the fly, thus reducing testing time.

5. One last tip on this issue. In the last phase, after tackling most of the problems, you still have to review all the languages on all the resolutions. In such case we recommend having one device on each hand, each with a different language and passing through the various screens one by one. We found this kind of 'multi-tasking' to save almost 50% on testing time.

Last but not least - everyone in the company is a user. The testing team has an obvious inherent interest in having everyone in the company using devices as users, thus gaining general knowledge on devices as well as occasional bug reports. The testing team should be the evangelist in the ongoing development of platform-savvy team members by continuous daily usage of devices and encouraging them to report bugs. Recompensing team members with a little shot of single malt whisky for each bug reported can do magic!

Good Mobile Hunting!

Biography

Dario Uriel Estrin, Vice President of Operations, WorldMate Inc.
Dario started his career as a programmer in 1994 and was initiated to SQA as a testing automation programmer. He later led testing teams. In WorldMate, he was responsible for various departments including SQA, support, IT and corporate sales. As a veteran who worked in various startup companies, Dario brings a wide and integrative view. He strongly believes in the additional value that interdisciplinary testing teams hold and that a strong testing team is a pivot player in any successful product, integrating business, customer and technological views into one.
Dario is Certified Scrum Master. He is currently taking an MBA and holds a B.A. in Political Science. He was a university assistant teacher and led a research team which focused on IT systems in public institutions.
http://il.linkedin.com/in/darioestrin

Bernard Shai Lelchuk
Bernard is a Senior QA Engineer at WorldMate Inc, holds a Bachelor in Recording Arts, is a certified System Administrator, a certified Audio Engineer & a passionate musician. His diverse experience in the IT & music production domains were his key to the testing world, which was followed by many years of beta testing music applications. This know-how allowed him to achieve wide recognition of his testing abilities, subsequently leading testing teams on various platforms and projects.
Bernard joined WorldMate in 2006 when he took his first formal position in the software testing industry. During his career, Bernard mentored many novice/veteran testers. He was responsible for most of the company's product testing and release processes, thus obtaining a wealth of knowledge in testing tools, automation, techniques and methodologies.
Bernard published several articles on web & mobile testing. In his free time he is a frequent guest blogger, leading tester, content coach, community forum moderator & contributor at uTest (the largest on-line professional testing community).
http://bernardlelchuk.wordpress.com/

The Test Data Challenge for Database-Driven Applications

by Klaus Haller

It was the summer of 2008. Testing suddenly emerged as *the* topic in my professional life. I became responsible for the database back-end of a credit rating application. Banks use such applications for estimating whether companies pay back their loans. This impacts whether they get a loan and for which interest rate. So I was interested in how to test "my" application. But the application was different to the ones you read about in testing literature. The application was built on a database (a database-driven application, or, short, a DBAP). The DBAP output of an action not only depends on your input, but also on the history of the DBAP's usage. The history manifests in rows and data stored in the tables. And this history influenced the present and future behavior of "my" application. What sounds like a philosophical question is of high practical relevance. Four questions sum up what I had to answer myself: What do I need DBAP test data for? When is a DBAP "correct"? What is a DBAP test case? And, finally, which test data is "good"?

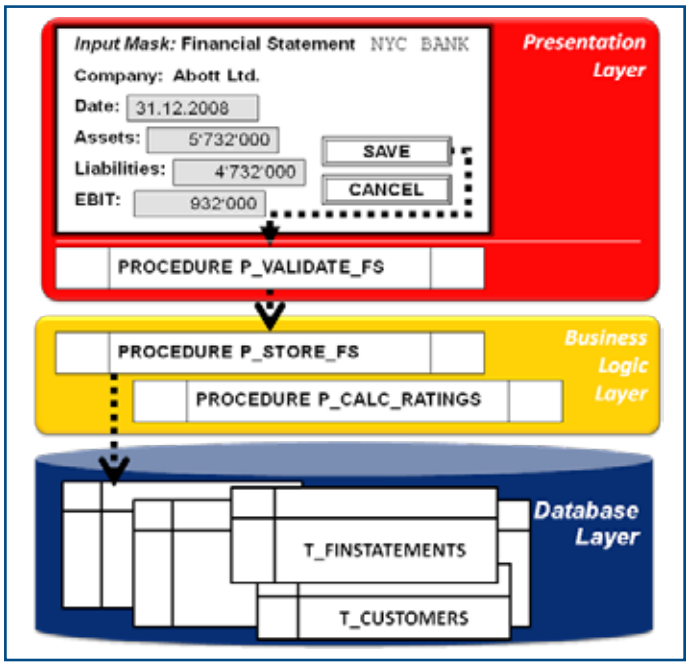


Figure 1: Sample Credit Rating Application

These questions are the roadmap for this article. A simplified credit rating application (Figure 1) will serve as an example. It is a three-tier application. The database layer stores the data persistency in tables. Table T_CUSTOMERS contains the information about the customer companies. Table T_FINSTATEMENTS stores financial statements for the various companies. The middle layer contains the business logic. It has two procedures. Procedure P_STORE_FS stores a financial statement in the database. Procedure P_CALC_RATINGS calculates the rating for all customers. On top, there is the presentation layer. It provides the GUI input form for manual input of financial statements. Also, it comprises the procedure P_VALIDATE_FS. The procedure checks whether the financial statement inserted into the GUI is "sensible", e.g. whether the sum of all assets and liabilities are equal.

DBAP Test Data Cube

Conventional tests know only one kind of data: data used as input parameters for the procedure to be invoked[1]. It is the idea of the DBAP test data cube (Figure 2) to visualize why and what kind of test data testers need. The cube has three dimensions: test trigger, test stage, and data purpose. The first dimension is the **test trigger**. It represents why we test. This issue is discussed intensively in literature. So we point out only five main reasons. It can be a completely *new application* that has never been tested before. When our credit rating application comes to the market the first time, the trigger is "new application". The subsequent releases trigger tests for reason two: a *new release* of an existing application. We test the new functionality and do regression tests for the existing functionality. Thus, if we add rating functionality, we need data for testing the new functionality as well as data for regression testing the existing functionality. A new application or a new release are two test triggers for software vendors. Customers buying software have other reasons for testing: parameterization and satellite system interaction. *Satellite system interaction* reflects that most of today's applications run in complex application landscapes. Processes span many applications. Various applications have to interact. Thus, we test the interaction of an application with its satellite systems. The fourth test trigger is *parameterization*. Standard software such as SAP or Avaloq allows adapting the software and the workflows according to specific customers' needs. One might parameterize e.g. that three persons have to check the rating for loans over ten millions. Whereas the customer can trust the standard software, the customer has to test whether its parameterization works as intended. Finally, the underlying *infrastructure* can trigger regression tests, e.g. when operating systems or compilers change.

The second dimension of the DBAP test data cube represents the

1 „Invoking a procedure" is a terminology typical for unit tests. However, it is meant in an inclusive way, i.e. this term also includes GUI based actions with input and output values inserted or presented via the GUI.

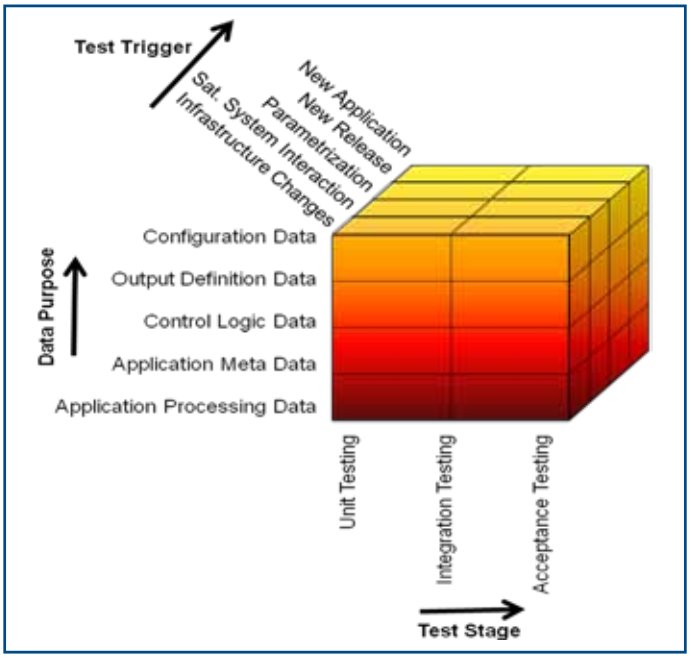


Figure 2: DBAP Test Data Usage Cube

test stages. Test stages are the different steps in testing addressing different needs and can be done by different units of the company. Typical stages are unit tests, integration tests, and acceptance tests. The exact steps and their names often depend on the process of the company.

The last dimension is the **data purpose**. It is the role the data plays in the DBAP. The actual roles might depend on the concrete architecture. We have identified the following data purporses as especially important in our applications: *Application processing data* is data everybody is aware of at first glance. Customers and their financial statements are examples in the context of our credit rating application. *Application meta data* is data which is more or less stable, but influences the "normal" data processing. An example is tables with credit pricing data. The data determines the risk-adjusted interest rate based on the credit score of a company. If the company has a scoring of "medium", the interest rate might be 9.2%. If the score is "excellent", the interest rate might be 3.7%. *Control logic data* influences the execution of processes and workflows. An example would be the ten million limit for loans. Loans over ten million demand three persons to check the rating. *Output definition data* defines the design and appearance of reports and customer output. The bank name "NYC BANK" or a bank logo are examples. Finally, *configuration data* deals with the IT infrastructure the application is deployed to. Examples are the configuration of interfaces, e.g. to SWIFT.

DBAP Correctness

Testing aims at finding as many bugs as possible as early as possible in the development life-cycle. The DBAP shall be or shall become "correct". But correctness in the context of DBAPs has various meanings. A data architect, a database administrator, a software developer, and a tester might focus on completely different aspects. To point this out, we present three possible DBAP correctness concepts (Figure 2): schema correctness, conformity correctness, and application correctness.

Schema correctness focuses on the database schema the DBAP uses for storing its data (green). Schema correctness understands correctness as having (a) a specification that reflects the real world and (b) an implementation reflecting the specification and the real world. Our credit rating application stores financial statements in the database schema. Schema correctness means in this context: First, there is one table (or more) for storing the financial statements. Second, the table has attributes for all the information provided by financial statements. Third, the financial statements must refer to the companies they belong to.

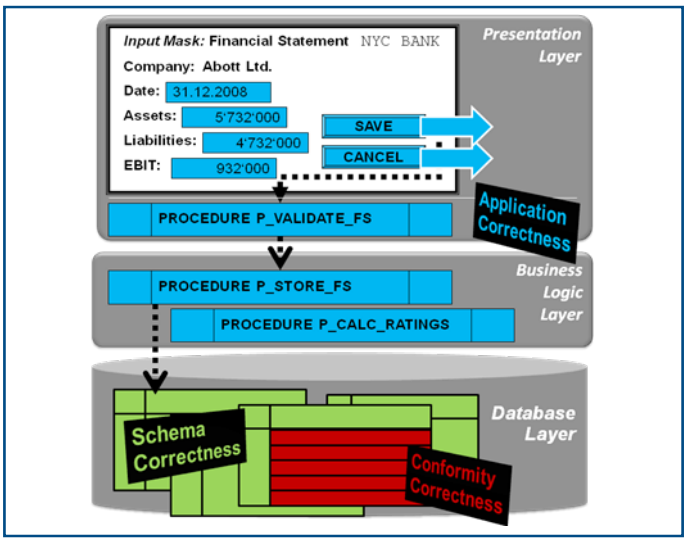


Figure 3: DBAP Correctness Criteria

Conformity correctness (brown) focuses on constraints or dependencies which are not part of the schema. The dependencies between balance sheet positions and profit-and-loss accounts are a good example. They are too complex to be reflected by database constraints. In our example, there are also no constraints in our database enforcing that the sum of all assets and all liabilities are equal. The data (and the DBAP) is only conformity-correct if it reflects also these non-schema-enforced constraints. Conformity correctness is similar to the concept of assertions in programming languages such as Java. Assertions do not improve the quality by looking at the result of actions, but by ensuring that the preconditions are as they have to be. Whereas these two correctness criteria focus only on the database, the third criterion, **application correctness**, looks at the complete DBAP behavior as observed e.g. via the GUI. However, it makes sense not to concentrate only on GUI interactions. Also batch processes such as the rating calculation procedure P_CALC_RATINGS are relevant. Application correctness is the most intuitive DBAP correctness criterion. Thus, we rely on it for discussing DBAP test cases.

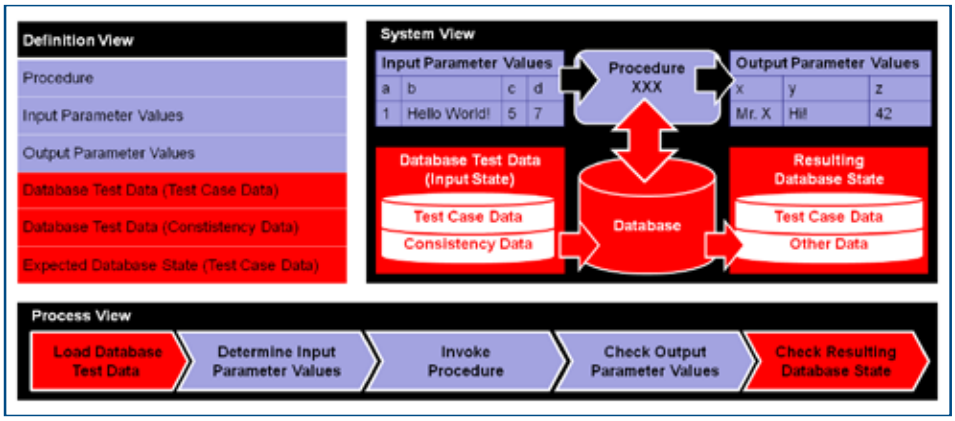


Figure 4: Understanding DBAP Test Cases (conventional test case: blue, DBAP extension: red)

Test Cases

Literature defines test cases based on three elements: the procedure to be invoked and the input and output parameter values. This model is suitable for stateless applications such as a calculator. A calculator returns *always* "7" after pressing "2", "+", and "5."

We need to adopt the test case concept for DBAPs, because they are not stateless. Procedure P_CALC_RATINGS needs financial statements to operate on. Procedure P_STORE_FINSTATEMENT needs a customer who a new financial statement refers to. So we need an initial database state. Also, these procedures show that we cannot rely only on the GUI output for deciding whether a procedure works correctly. We have to check the database whether a new financial statement was added to the database (P_STORE_FINSTATEMENT), or whether the rating calculations are correct (P_CALC_RATINGS). So a DBAP test case consists of five elements: input parameter values, an input database state, a procedure to be invoked, output parameter values, and a resulting database state.

This model was first presented by Willmor and Embury. We extended it for practical usage for complex systems by distinguishing two parts of the input state. ERP systems or core-banking-systems have hundreds or thousands of tables. One table might be relevant for our test case, but we need the other hundreds or thousands to be filled such that we can perform our test case. Thus, we divide our input state into two parts, test case data and consistency data. We illustrate this with procedure P_CALC_RATINGS. Here, we want to test whether the rating function works correctly, e.g. whether a bankrupt company gets rating "0". So we need test case data in table T_FINSTATEMENTS. This test data must contain a financial statement of a bankrupt company. However, we can add such a financial statement if we can link it to a customer in table T_CUSTOMERS. Thus, the customer would be consistency data. After execution of the procedure, we might have to check whether the rating is as expected. Thus, we look at data in the database. Again, there are two kinds of data. There is data we are interested in (test case data), e.g. the rating information in T_FINSTATEMENTS. We can ignore all other tables in this particular case, because it is not in the focus of this test case.

Figure 4 compares a conventional test case and a DBAP test case. *Blue* reflects what a conventional test case definition contains, the involved system components, and which actions (input parameter selection, invocation, checking the outcome) have to be done. The needed extensions for a DBAP test case are shown in red. These are the input and output states, the database, loading the database before the test case execution, and, potentially, checking the resulting state.

Quality

All roads lead to Rome. And in projects many ways lead to DBAP test data. One can design them by analyzing the specification. One might use commercial data generation tools. The decision often depends (besides on costs) on the test data quality. If we want to compare the quality of different DBAP test data, we need a notion of quality. In other words: We have to understand what DBAP test quality means and how it can differ.

T_FINSTATEMENTS			T_CUSTOMERS	
OWNER_ID	SUM_AS	SUM_LI	ID	NAME
NUMBER	NUMBER	NUMBER	NUMBER	VARCHAR2(100)
67	120000	12000	55	ALICE CORP.
CONSTRAINTS: PRIMARY KEY(OWNER_ID); FOREIGN KEY (OWNER_ID) REFERENCES T_CUSTOMERS(ID); SUM_AS NOT NULL; SUM_LI NOT NULL; CHECK(SUM_AS>0); CHECK(SUM_LI>0);			67	BETTY LTD.

Figure 5: Sample Tables with Constraints

Therefore, we rely on the concept of *test data compliance levels*[2]. The compliance levels (Figure 5) are like a stair with four steps. It requires effort to get to the next step, but you gain quality. The lowest level is **type compliance**. Type compliance considers the data type of the columns. Table T_FINSTATEMENTS has three columns: one stores the sum of all assets, one the sum of all liabilities; an ID column refers to the customer ID in table T_CUSTOMER. The reference refers to the company that the financial statement belongs to. Type compliance demands that we insert only rows for which all attributes have the right type.

We take a look at the following three INSERT statements for table T_FINSTATEMENTS (Figure 5):

```
(1) INSERT T_ FINSTATEMENTS(OWNER_ID, SUM_AS, SUM_LI)
      VALUES('ALICE CORP.', 'ONE MILLON', 'NO INFORMATION');
(2) INSERT T_ FINSTATEMENTS(OWNER_ID, SUM_AS, SUM_LI)
      VALUES (55, -50'000,  NULL);
(3) INSERT T_ FINSTATEMENTS(OWNER_ID, SUM_AS, SUM_LI)
      VALUES (32, 23'000, 20'000);
```

Statement (1) does not reflect that the attributes must have the data type NUMBER. It is not type compliant. Statements (2) and (3) are type-compliant. However, statement (2) does not make sense. It does not reflect the schema constraints. A NULL value is not allowed for attribute SUM_LI. Also, there is no customer with ID 55 in table T_CUSTOMERS. Next, the check constraints demand that the values for the sum of all assets and liabilities are positive. The problem from a testing perspective is that all rows not complying with a constraint are rejected by the database. So if we prepare 50 type-compliant rows, we do not know whether 50, 45, or 0 rows make it into the database. However, statement (3) reflects this requirement, as does statement (4). Thus, we use the term **schema compliance** for statements (3) and (4). The advantage compared to only type-compliant data is the guarantee that all rows are loaded into the database.

```
(4) INSERT T_ FINSTATEMENTS(
              OWNER_ID, SUM_AS, SUM_LI)
      VALUES (32, 20'000, 20'000);
```

We can achieve the two previous compliance levels "type compliance" and "schema compliance" relying only on information of the database catalogue[3]. The two highest compliance levels need more information. From an application point of view, the sum of all assets and liabilities is always equal. SUM_AS and SUM_LI must be equal. This is not reflected by the database schema. In the case of GUI input, procedure P_VALIDATE_FS ensures this. Otherwise, the procedure rejects the GUI input. So we have dependencies between attributes, which are enforced by the application and not reflected by constraints. Such dependencies can also exist between tables, e.g. one table with financial statements and a table with profit and loss information. The problem with dependencies not reflected by schema constraints is that there might be data that has been inserted in the database which does not reflect these dependencies. Thus, the DBAP might be in a state that was not specified. The consequence can be unexpected behavior of the application. If errors emerge for unspecified circumstances, they are false positives[4]. Such circumstances would never appear under normal usage. So our third level is **applica-**

2 *K. Haller: White-Box Testing for Database-driven Applications: A Requirements Analysis, Second International Workshop on Testing Database Systems, Providence, RI, 29.6.2009*

3 *The database catalogue are dedicated tables in a database which store all information about the content of the database: users, tables, constrains, views, access rights etc.*

4 *False positive means that testers seem to have found a failure. After an analysis by the testers or software developer the failure turns out not be a "failure". Certainly, they are costly. If they appear too often, one risks that software engineers might stop taking "failures" seriously. They might assume all failures to be false positives and stop analyzing potential failures in a sensible way.*

Database Constraints restrict which data can be stored in the tables of a database. There are the following constraints: *Unique constraints* state that this attribute must be different for all rows of a table. ID columns are a good example. *Primary key* constraints allow identifying a row. They can span more than one attribute. *Foreign keys* refer to primary keys of a different table. They ensure that there is a fitting value in the different table, e.g. a financial statement refers always to an existing (!) customer ID in the customers table. *Not null constraints* demand that there is a value provided for this attribute, e.g. that all financial statement have a sum of assets or a sum of liabilities. *Check constraints* allows formulating nearly arbitrary conditions.

tion compliance. Application compliance means that the DBAP input state could be the result of "normal" GUI input and data processing. Statement (4) is a perfect example. Now no false positives can appear. But still, test data can be better. Let us assume a test case shall test the credit rating functionality for large corporations with assets and liabilities of more than 1'000'000. We are interested whether procedure P_CALC_RATINGS considers specific risks of large companies. This test case requires a financial statement with "higher" values such as statement (5).

```
(5) INSERT T_ FINSTATEMENTS(OWNER_ID, SUM_AS, SUM_LI)
      VALUES (32, 1'500'000, 1'500'000);
```

The difference between statements (4) and (5) is that the latter allows us to test the execution path (or test case) we are interested in: the credit rating for large corporations. If the DBAP test data is suitable for a test case, it is **path-compliant** (the term refers to the path coverage criterion). Path compliance bases always on a test case and a specific execution path of the application. This is the highest level we can achieve. However, it also makes clear that different test cases might need different test data sets. Figure 5 compiles the information about all four compliance levels.

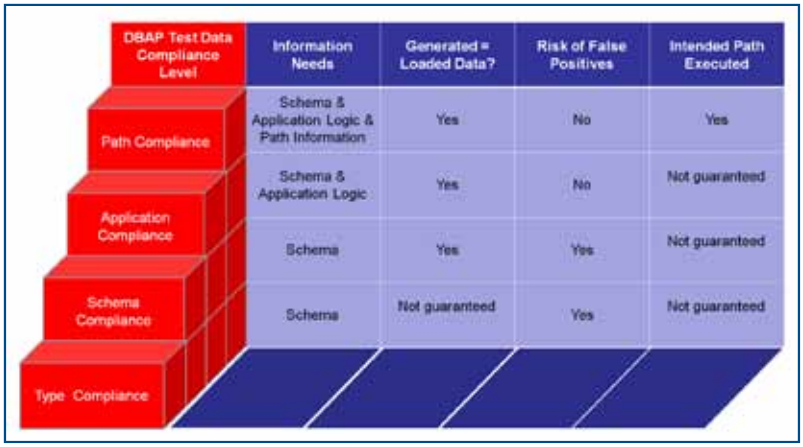


Figure 6: Overview Compliance Levels

In this article, we explained the typical problems of testing database-driven applications. Instead of providing simple answers to complex problems, we concentrated on explaining the different challenges for projects. Our goal is to foster discussions within projects to find the best solutions for their particular problems. Thus, we explained the most important concepts: the test data cube, correctness, test cases, and quality for DBAPs.

Biography

Klaus Haller is a Senior Consultant with COMIT, a subsidiary of Swisscom IT Servies. COMIT focuses on projects and application management services for the financial industries. Klaus' interest is the architecture and management of information systems. His experiences span from process engineering, architecture and implementation of data migration solutions for core-banking systems to the responsibility for the database backend of a credit rating application. Klaus studied computer science at the Technical University of Kaiserslautern, Germany. He received his Ph.D. in databases from the Swiss Federal Institute of Technology (ETH) in Zurich, Switzerland. He welcomes your feedback at klaus.haller@comit.ch

Load Testing Web Applications – Do it on the DOM Level!

by Ronny Vogel

HTTP-level tools record HTTP requests on the HTTP protocol level. They usually provide functionality for basic parsing of HTTP responses and building of new HTTP requests. Such tools may also offer parsing and extraction of HTML forms for easier access to form parameters, automatic replacement of session IDs by placeholders, automatic cookie handling, functions to parse and construct URLs, automatic image URL detection and image loading, and so on. Extraction of data and validation are often done with the help of string operations and regular expressions, operating on plain HTTP response data. Even if HTTP-level tools address many load testing needs, writing load test scripts using them can be difficult.

Challenges with HTTP-Level Scripting

Challenge 1: Continual Application Changes

Many of you probably know this situation: A load test needs to be prepared and executed during the final stage of software development. There is a certain time pressure, because the go-live date of the application is in the near future. Unfortunately, there are still some ongoing changes in the application, because software development and web page design are not completed yet.

Your only chance is to start scripting soon, but you find yourself struggling to keep up with application changes and adjusting the scripts. Some typical cases are described below.

- The protocol changes for a subset of URLs, for example, from HTTP to HTTPS. This could happen because a server certificate becomes available and the registration and checkout process of a web shop, as well as the corresponding static image URLs, are switched to HTTPS.
- The URL path changes due to renamed or added path components.
- The URL query string changes by renaming, adding or removing URL parameters.
- The host name changes for a subset of URLs. For example, additional host names may be introduced for a new group of servers that delivers static images or for the separation of content management URLs and application server URLs that deliver dynamically generated pages.
- HTML form parameter names or values are changed or form parameters are added or removed.
- Frames are introduced or the frame structure is changed.
- JavaScript code is changed, which leads to new or different HTTP requests, to different AJAX calls, or to a new DOM (Document Object Model) structure.

In most of these cases, HTTP-level load test scripts need to be adjusted. There is even a high risk that testers do not notice certain application changes, and although the scripts do not report any errors, they do not behave like the real application. This may have side effects that are hard to track down.

Challenge 2: Dynamic Data

Even if the application under test is stable and does not undergo further changes, there can be serious scripting challenges due to dynamic form data. This means that form field names and values can change with each request. One motivation to use such mechanisms is to prevent the web browser from recalling filled-in form values when the same form is loaded again. Instead of „creditcard_number", for example, the form field might have a generated name like „cc_number_9827387189479843", where the numeric part is changed every time the page is requested. Modern web applications also use dynamic form fields for protection against cross-site scripting attacks or to carry security-related tokens.

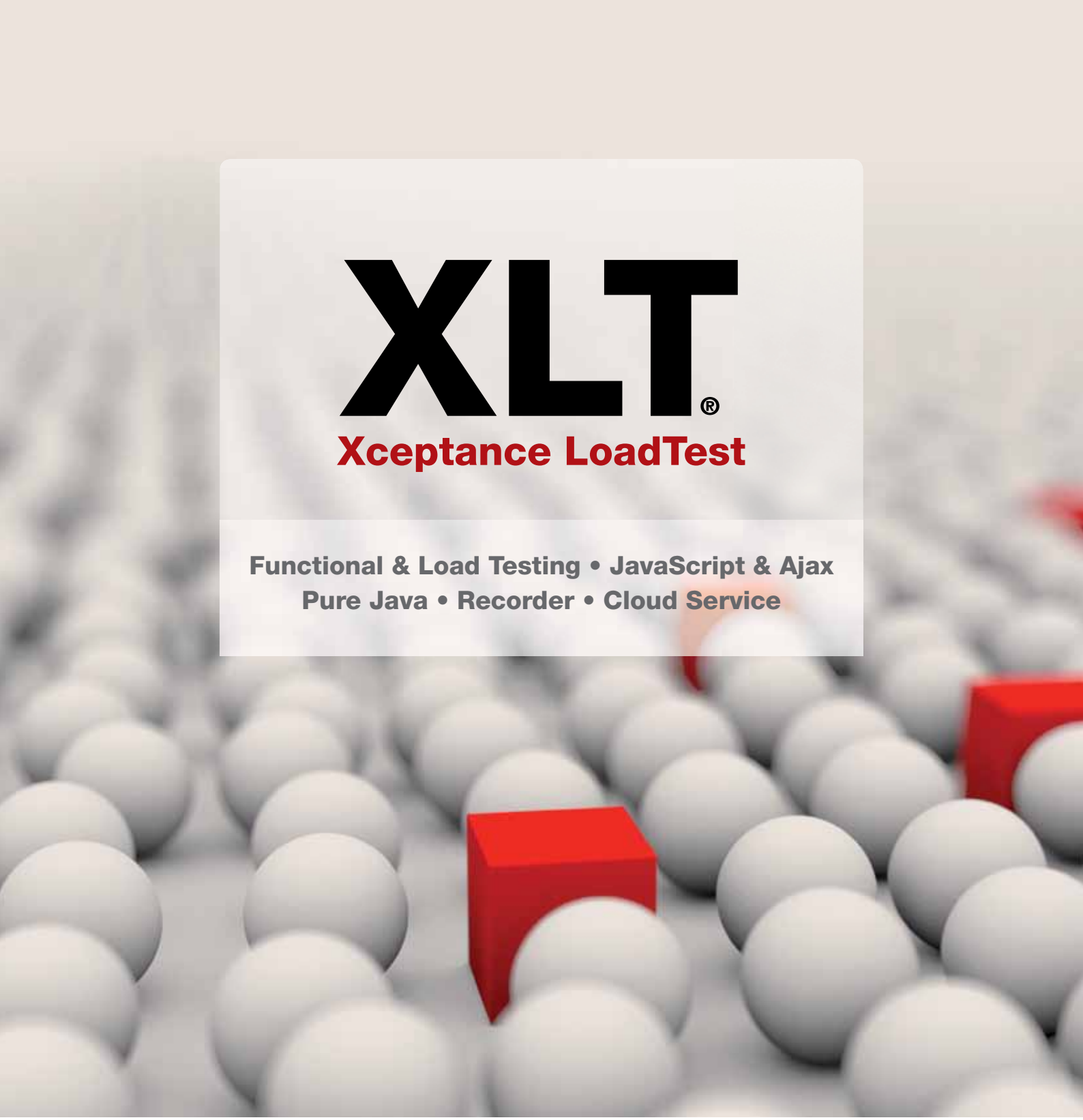
Another problem can be data that is dynamically changing, because it is maintained and updated as part of the daily business. If, for example, the application under test is an online store that uses search-engine-friendly URLs containing catalog and product names, these URLs can change quite often. Even worse, sometimes the URLs contain search-friendly catalog and product names, while embedded HTML form fields use internal IDs, so that there is no longer an obvious relation between them.

Session IDs in URLs or in form fields may also need special handling in HTTP-level scripts. The use of placeholders for session IDs is well supported by most load test tools. However, special script code might be needed, if the application not only passes these IDs in an unmodified form, but also uses client-side operations on them or inserts them into other form values.

To handle the above-mentioned cases, HTTP-level scripts need manually coded, and thus unfortunately also error-prone, logic.

Challenge 3: Modeling Client-Side Activity

In modern web applications, JavaScript is often used to assemble URLs, to process data, or to trigger requests. The resulting requests may also be recorded by HTTP-level tools, but if their



Xceptance
Software Technologies

The Software Testing Experts.
www.xceptance.com

For more information please visit our website at www.xceptance.com/xlt

URLs or form data change dynamically, the logic that builds them needs to be reproduced in the test scripts.

Besides this, it can be necessary to model periodic AJAX calls, for example to automatically refresh the content of a ticker that shows the latest news or stock quotes. For a realistic load simulation, this also needs to be simulated by the load test scripts.

Challenge 4: Client-Side Web Browser Behavior

For correct and realistic load simulations, the load test tool needs to implement further web browser features. Here are a few examples:

- caching
- CSS handling
- HTTP redirect handling
- parallel and configurable image loading
- cookie handling

Many of these features are supported by load test tools, even if the tools act on the HTTP level, but not necessarily all of them are supported adequately. If, for example, the simulated think time between requests of a certain test case is varied, a low-level test script might always load the cacheable content in the same way – either it was recorded with an empty cache and the requests are fired, or the requests were not recorded and will never be issued.

DOM-Level Scripting

What is the difference between HTTP-level scripting tools and DOM-level scripting tools? The basic distinction between the levels is the degree to which the client application is simulated during the load test. This also affects the following characteristics:

- Representation of data: DOM-level tools use a DOM tree instead of simple data structures.
- Scripting API: The scripting API of DOM-level tools works on DOM elements instead of strings.
- Amount and quality of recorded or hard-coded data: There is much less URL and form data stored with the scripts. Most of this data is handled dynamically.

DOM-level tools add another layer of functionality on top. Besides the handling of HTTP, these tools also parse the contained HTML and CSS responses to build a DOM tree from this information, similar to a real web browser. The higher-level API enables the script creator to access elements in the DOM tree using XPath expressions, or to perform actions or validations on certain DOM elements. Some tools even incorporate a JavaScript engine that is able to execute JavaScript code during the load test.

Advantages

DOM-level scripting has a number of advantages:

- Load test scripts become much more stable against changes in the web application. Instead of storing hard-coded URLs or data, they operate dynamically on DOM elements like "the first URL below the element xyz" or "hit the button with id=xyz". This is especially important as long as application development is still ongoing. As a consequence, you can start scripting earlier.
- Scripting is easier and faster, in particular if advanced script functionality is desired.
- Validation of result pages is also easier on the DOM level compared to low-level mechanisms like regular expressions. For example, checking a certain HTML structure or the content of an element, like "the third element in the list below the second H2" can be easily achieved by using an appropriate XPath to address the desired element.
- Application changes like changed form parameter names normally do not break the scripts, if the form parameters are not touched by the script. But, if such a change does break the script because the script uses the parameter explicitly, the error is immediately visible since accessing the DOM tree element will fail. The same is true for almost all kinds of application changes described above. Results are more reliable, because there are fewer hidden differences between the scripts and the real application.
- CSS is applied. Assume there is a CSS change such that a formerly visible UI element that can submit a URL becomes invisible now. A low-level script would not notice this change. It would still fire the old request and might also get a valid response from the server, in which case the mismatch between the script and the changed application could easily remain unnoticed. In contrast, a DOM-level script that tries to use this UI element would run into an error that is immediately visible to the tester.
- If the tool supports it, JavaScript can be executed. This avoids complicated and error-prone re-modeling of JavaScript behavior in the test scripts. JavaScript support has become more and more important in recent years with the evolution of Web 2.0/AJAX applications.

Disadvantages

There is one disadvantage of DOM-level scripting. The additional functionality needs more CPU and main memory, for instance to create and handle the DOM tree. Resource usage increases even more if JavaScript support is activated.

Detailed numbers vary considerably with the specific application and structure of the load test scripts. Therefore, the following numbers should be treated with caution. Table 1 shows a rough comparison, derived from different load testing projects for large-scale web applications. The simulated client think times between a received response and the next request were relatively short. Otherwise, significantly more users might have been simulated per CPU.

Scripting Level	Virtual Users per CPU
HTTP Level	100..200
DOM Level	10..30
DOM Level + JavaScript execution	2..10

If you evaluate these numbers, please keep in mind that machines are becoming ever more powerful and that there are many flexible and easy-to-use on-demand cloud computing services today, so that resource usage should not prevent DOM-level scripting.

Conclusion

Avoid hard-coded or recorded URLs, parameter names and parameter values as far as possible. Handle everything dynamically. This is what we have learned. One good solution to achieve this is to do your scripting on the DOM level, not on the HTTP level. If working on the DOM level and/or JavaScript execution are not possible for some reason, you always have to make compromises and accept a number of disadvantages.

We have created and executed web-application load tests for many years now, in a considerable number of different projects. Since 2005, we have mainly used our own tool, Xceptance LoadTest (XLT), which is capable of working on different levels and supports fine-grained control over options like JavaScript execution. In our experience, the advantages of working on the DOM level, in many cases even with JavaScript execution enabled, generally by far outweigh the disadvantages. Working on the DOM level makes scripting much easier and faster, the scripts handle many of the dynamically changing data automatically, and the scripts become much more stable against typical changes in the web application.

Biography

Ronny Vogel is technical manager and co-founder of Xceptance. His main areas of expertise are test management, functional testing and load testing of web applications in the field of e-commerce and telecommunications. He holds a Masters degree (Dipl.-Inf.) in computer science from the Chemnitz University of Technology and has 16 years of experience in the field of software testing. Xceptance is a provider of consulting services and tools in the area of software quality assurance, with headquarters in Jena, Germany and a branch office in Cambridge, Massachusetts, USA.

Where are the non-functional requirements?

Column

by Erik van Veenendaal

Although this issue of Testing Experience focuses on performance testing specifically, I would like to look at non-functionals and their state-of-the-practice from a broader perspective. Let me start with a quote:

"…. The increasing intensive use of software products in society as a whole, but especially in business environments, has made the IT user aware of the possibilities of software products. Users of software products have pushed quality to the forefront of their wishes. They can no longer be satisfied with software products that 'only' meet the defined functional requirements, that are delivered in time and at reasonable costs. Users of software products ask for clear, objective and quantifiable software quality."

This quote from Dr. ir. Jos Trienekens (associate professor at the Eindhoven University of Technology) dates back to the early nineties (over 15 years ago!). What he basically stated is that quality is not only determined by the functionality of the product, but largely also by its non-functional characteristics, such as the reliability, usability or portability. I believe that in principle we all agree to this statement. However, what, if anything, has happened in the last 15 years regarding the non-functionals in our real-life projects?

Requirements perspective

One cannot test quality into a product. Quality starts with clear and unambiguous requirements. Most of the requirements specifications that I have seen, have very limited focus on non-functional requirements. Just on the last page they state "it shall be usable". Good luck …. !! Maybe not so strange, if we consider that for instance the IREB Foundation syllabus for requirements engineering certification pays little attention on how to specify measurable non-functional requirements. And the well-known standard ISO 9126 was to be expanded to also define how to write requirements based on the non-functional quality characteristic terminology defined by the standard. But so far there have been no results.

Engineering perspective

Of course translating non-functional requirements into design and code is a major challenge. What engineering measures need to be taken to enhance the portability or reliability of a product? There is a whole community built around reliability engineering, and also for other non-functionals, such as maintainability, there are supporting techniques, such as code metrics and coding standards. In fact, I believe there is a lot available to support engineers in doing a good job provided they are given the time by management to do so. They tend to focus on more technically oriented non-functionals though. However, why should they worry, if no or only a few vague non-functional requirements are defined?

Testing perspective

How many testers actually spend more than 5% of their time testing non-functionals? How many testers know (which of course does not mean they can apply!) more than one non-functional technique for each non-functional characteristic? [On the other hand: Sit down with your test team and try to identify three different test design techniques for reliability, maintainability and usability. Would you succeed?] It is a good thing that non-functional characteristics are included in the ISTQB Advanced Level syllabus, especially within the Technical Test Analyst module, and that security has been identified as an ISTQB Expert Level topic. Testers need much more knowledge and skills on this. Test managers should understand how to define non-functional acceptance criteria, perform a product risk analysis for non-functionals, etc. Some, e.g. within TMMi, state that one first has to master functional testing before starting to build knowledge and skills on non-functional testing. If a calculation is wrong, who cares about the performance and usability of the calculation function? But again, why should we worry to build those skills if there are no or only few vague non-functional requirements defined?

Business perspective

In the end we (requirements analysts, software engineers and testers) are all just providing a service to the business. If they don't care, why should we? The business should also take their responsibility in asking for non-functional requirements. One of the problems regarding non-functionals is that they are often too technical to discuss with the business. It is just not on their patch (or so they think). Perhaps it is our responsibility to educate the business representatives and explain the importance to them in a business language. In this context, new development methodologies, such as Agile/SCRUM, do not help either. Is it not great as a user to get new functionality delivered fast, e.g. every 4 weeks? However, how much non-functional engineering and testing can you do in a four-week sprint?

Food for thought

So what's next? I certainly do not have the answer, but I do believe it is a shared responsibility and each discipline will have to play their part. We are probably still in a software crisis, and only focusing on functionality being delivered faster does not always help. It sometimes pays off to deliver less functionality, but to deliver it inside more reliable software. It should not be about making business tomorrow, but also about the years to come. Ask the automotive industry; they may have recently learned this lesson. Software quality, specifically non-functional quality, is still largely underestimated, and there is a lot of work to do!

What are you doing on non-functional requirements, engineering and testing in your projects.....?

Erik van Veenendaal is a leading international consultant and trainer, and recognized expert is the area of software testing and quality management. He is the director of Improve Quality Services BV. At EuroStar 1999, 2002 and 2005, he was awarded the best tutorial presentation. In 2007 he received the European Testing Excellence Award for his contribution to the testing profession over the years. He has been working as a test manager and consultant in software quality for almost 20 years.
He has written numerous papers and a number of books, including "The Testing Practitioner", "ISTQB Foundations of Software Testing" and "Testing according to TMap". Erik is also a former part-time senior lecturer at the Eindhoven University of Technology, the vice-president of the International Software Testing Qualifications Board and the vice chair of the TMMi Foundation.

Ranorex®

Automate Your UI Testing with Ranorex

Object-based Capture & Replay Editor

- Maintainable recordings via the actions table editor
- Integration of Ranorex repositories

Automated Testing of Web & Windows Applications

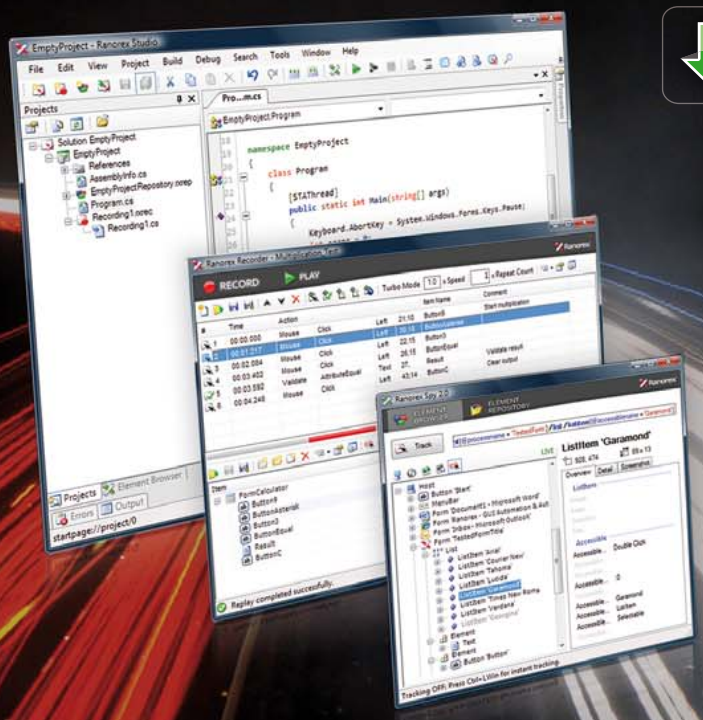
- Winforms / C# / VB.NET
- WPF / Silverlight / Win32 / MFC
- Flash / Flex / Web 2.0 / AJAX / /

Maintainable UI Object Repositories

- Easy to maintain all types of UI objects
- Separate test automation from UI identification

Write tests in C#, VB.NET and IronPython

- Automatic and flexible code generation in C#, VB.NET and IronPython
- All-on-one test environment with code editor, code completion and debugging



Get your 30-day Trial
www.ranorex.com

VISIT US!
29 Nov - 02 Dec
Copenhagen/Denmark
EuroSTAR 2010

Using Hudson to run junit regression tests on multiple machines

by Manik Thakur & Jan Kester

In this article we describe how Hudson, a continuous integration server, helped us to achieve a next level of test automation, where we start tests on several machines from one central console.

Test execution before using Hudson

In our department we have a big collection of unit test cases. For each build that we test, we run these test cases on multiple environments: multiple application servers, multiple databases and multiple configurations. We have a set of Virtual Machines (VMs), all with their own specific configuration, and on each individual VM execute the same set of tests. Although we used ANT to automate our execution steps on a single machine, we still spent a lot of time connecting up all machines:

- copying the new build to a central network location
- on each machine: updating unit tests from the SVN repository
- on each machine: starting ANT to install the new build, configuring the unit client environment, deploying software on the application server
- on each machine: starting the application server, setting up the initial database, running tests, loading test results into the database
- after execution has finished, generating a report with all test results.

Most of these steps are the same for each VM, and need to be repeated for each new build. So getting this automated promised a lot of potential time saving.

Introducing Hudson

Next comes Hudson, a continuous integration tool written in Java, It supports SCM tools including CVS, Subversion, Git and Clearcase and can execute Apache Ant and Apache Maven based projects, as well as arbitrary shell scripts and Windows batch commands. Continuous integration servers are especially strong in distributing tasks over multiple machines. There is one central console that dispatches tasks to all nodes, and thus binds jobs together. In our company we also have experience with Teamcity and Bamboo, which are both licensed. We then came across Hudson, which is an open-source solution. The good thing about Hudson is that it has a very active community, and many additional plug-ins have been released, extending it from purely being a build tool for Java projects. New builds come out every week. We have set up Hudson on two different networks, and so far we are very content. Below we will describe one of our set-ups, which runs in any servlet container.

Installation

On the Hudson website, http://Hudson-ci.org/, you can download the latest war, and either start this with Java -jar, or drop it in your favourite servlet container. When you run it from the command line, you can add some simple arguments to add basic security:

```
java -jar Hudson.war -argumentsRealm.passwd.
admin=donttell \
    --argumentsRealm.roles.admin=admin --http-
    Port=8001 --prefix=/Hudson
```

Now you can open a browser at http://localhost:8001/Hudson/, and log in with admin/donttell.

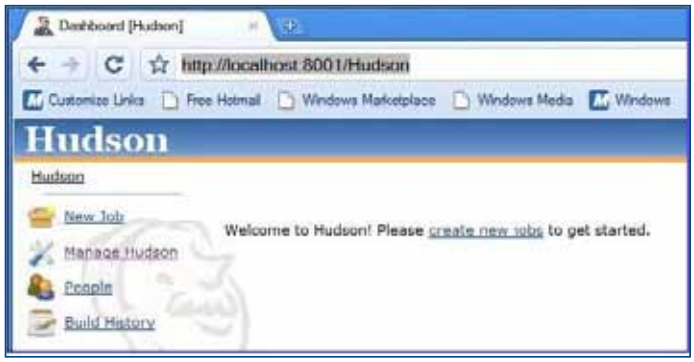


Image 1: Welcome Page of Hudson

Alternatively, Hudson can be installed as a Windows service. Under Manage Hudson, there is a link which will configure Hudson as a Windows service under the installed directory provided.

Configuration

Under manage Hudson -> Configure System, we configured global settings and paths, which were commonly used by all the nodes (VMs). On the main page you can change settings for:

- Smtp mail server: Of course, you will want this, as sending mail is essential for a continuous integration server.
- Security settings: We only used simple security to have one single admin.
- Default behavior for source code repositories.

- Number of parallel jobs you allow for this machine.
- Global variables valid for all jobs on all nodes.

Apart from the main page, the configuration menu also allows you to:

- Install additional plug-ins.
- Define your nodes.

Definition of nodes

In order to dispatch jobs to other machines, you will need to define these as nodes. The server that Hudson runs on, i.e. the console, is called the master, and all the nodes that Hudson connects to are called slaves. In Hudson, a slave can either be set up to run any job, or to run specific jobs only. In our test set-up, we have used the latter type. We connected to both Windows and Linux machines.

Build Executor Status		
#	Master	
1	Idle	
2	Idle	
147.249.65.1		
1	Idle	
2	Idle	
147.249.65.3		
1	Idle	
2	Idle	

Image2: Master and slaves

Connectivity to Windows nodes

Hudson offers several protocols for master-slave connection. For Windows slaves, we used the Windows-service option, which runs the slave jars as a Windows service on the remote machine. In the configuration page, Hudson requires IP, username, password and remote root directory, and on save will automatically try to connect to the remote machine and push the slave jars to the remote PC. There is no need to log in on the remote PC! We faced a few problems due to required .net version on the remote machine, but after updates got it all working. We did change one setting of the service though: instead of logging in as a local user, we told the service to connect as a domain user, otherwise Hudson's remote jobs would not have access to network resources.

Connectivity to Linux nodes

In order to connect to Linux nodes, we used the ssh protocol. With IP, username, password and remote root directory, Hudson could connect to the remote machine, pushed the slave jars to the remote root directory, and started the slave. The only thing to pay attention to is that Hudson does not read the .bashrc file, so it does not know any environment variables or path extensions defined here. We therefore had to make sure that we had a Java executable on the standard PATH. A symbolic link to /usr/local/bin worked fine here. The missing environment variables we could push from Hudson.

Jobs, an example

With all slaves in place, we could now create the jobs. But before going into this, it is maybe better to explain an example job: HelloWorld.

```
#!/bin/bash
echo "Hello World , $NAME"
```

We deployed this script under directory /home/tester, on one of the slaves. We then created a new job, and in this new job bound it to our slave and added a build step to execute this script. Inside the job, we tied this job to a specific node and added a parameter that a user needs to fill in. This parameter will be passed on to the job as environment variable.

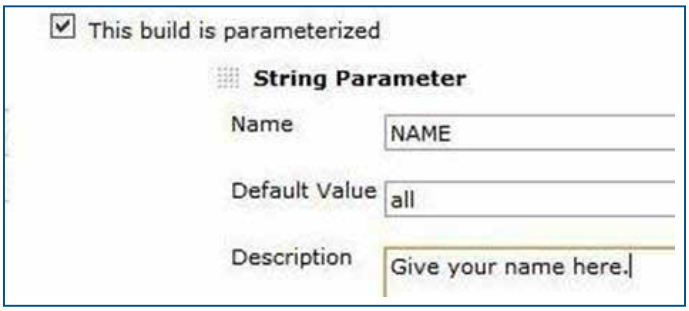
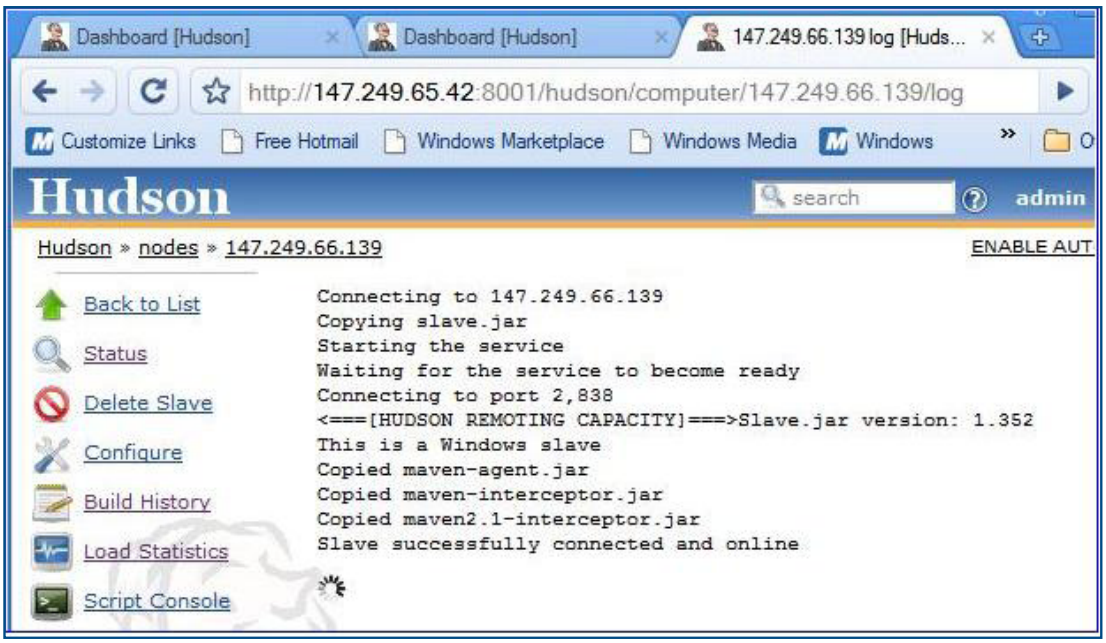


Image 4: Build parameterization

We then added an "Execute shell" build step that executes the command /home/tester/helloworld.sh. When we execute this job, we need to fill in NAME, and then the job gets started.



Why choose E-EXams?

More Flexibility in Scheduling:

- choose your **favourite test center** around the corner
- decide to take the exam at the **time** most convenient for you

Convenient System Assistance throughout the Exam:

- "**flagging**": you can mark questions and go back to these flagged ones at the end of the exam
- "**incomplete**": you can review the incompletely answered questions at the end of the exam
- "**incongruence**": If you try to check too many replies for one question the system will notify you

Immediate Notification: passed or failed

***IREB**® Certified Professional for Requirements Engineering (English, German) / **ISTQB**® Certified Tester Foundation Level (English, German, Spanish) / **ISEB**® Intermediate Certificate in Software Testing (German) / **ISTQB**® Certified Tester Advanced Level – Test Manager (English, German, Spanish) / **ISTQB**® Certified Tester Advanced Level – Test Analyst (English) / **ISTQB**® Certified Tester Advanced Level – Technical Test Analyst (English) / **ISSECO**® Certified Professional for Secure Software Engineering (English)*

www.isqi.org | certification@isqi.org

CERTIFYING PEOPLE

iSQI®
International Software Quality Institute

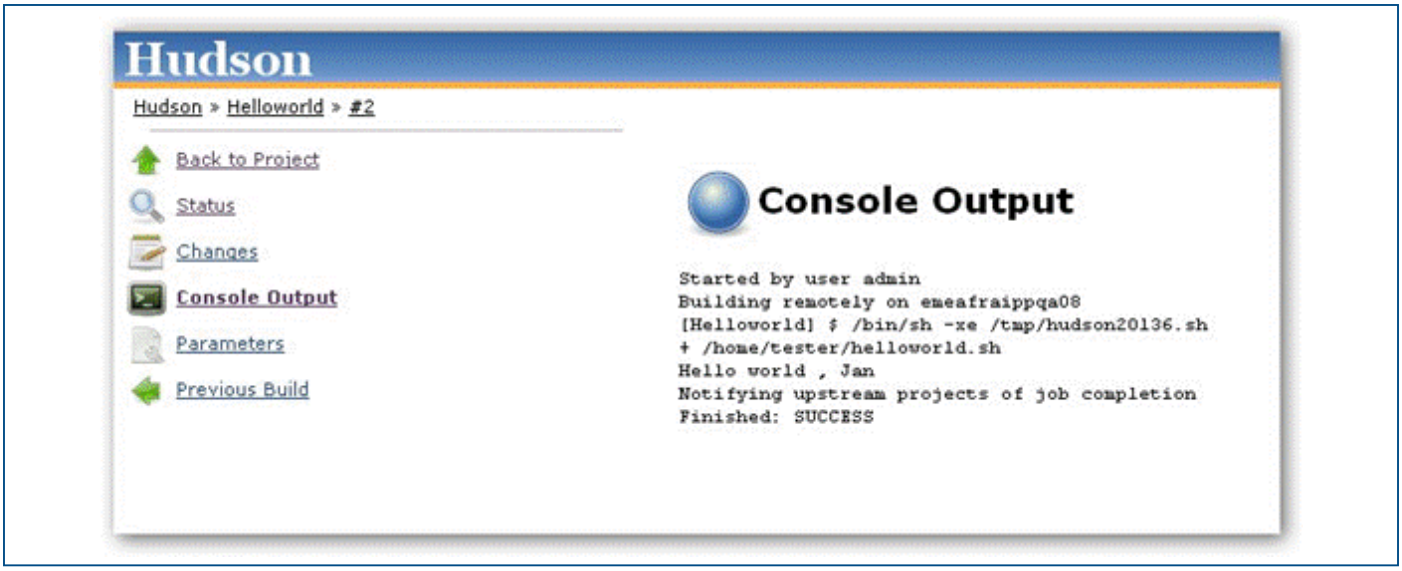


Image5: Output of job "HelloWorld"

Jobs for our tests

In our Hudson set-up, we defined a set of jobs to automatically execute our tests. These jobs need to be executed on each single machine, representing a single environment.

The jobs do the following:

- update_svn: revert and update from SVN
- init_server: clean and restart application server
- Install: install software, configure and deploy
- Runtest: setup, run tests, load results into database
- Reg_report_trunk: create report from database

Image6: Overview of jobs

update_svn

This job updates the sources of our test software. All update_svn jobs get started by one common job - updatesvn_all_trunk, i.e. the updatesvn_all_trunk job starts jobs update_svn_65_1, update_svn_65_3, etc. The updatesvn_all_trunk job does not do anything on its own; it is only responsible to start the children jobs. The job itself gets started via a remote trigger.

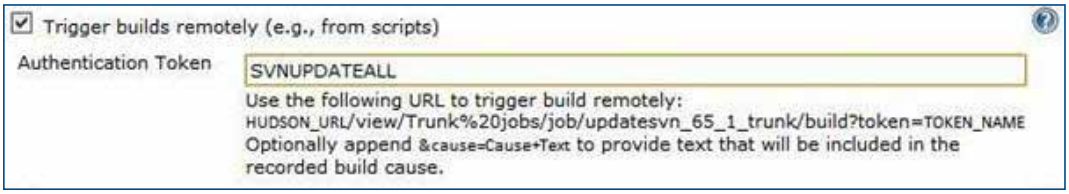


Image7: Remote trigger

With this remote trigger, the job can be started by calling the URL HUDSON_URL/view/Trunk%20jobs/job/updatesvn_all_trunk/build?token=TOKEN_NAME.

So any other process, ANT script or other continuous integration server can trigger the first job on this Hudson server. In our case, it is the software build server that starts the Hudson test server.

The update_svn job uses an ANT script to clean, revert local changes and get updates from SVN. We already had these ANT SVN targets in place, and therefore did not use Hudson's internal SVN integration. Our SVN jobs start ant target svn-all, with parameters svn.user=$SVNUSER and svn.password=$SVNPWD. These $-values represent variables, and will also be available on the node as environment variables. Variables can also be set during execution (as parameterized builds), on the level of a job, on the level of a node, or as a global parameter. As our svn.user and password are the same on all nodes; we used a global parameters for this, which you can define under main configure menu, and they are therefore valid for all nodes and all jobs.

Alternatively, Hudson also provides SVN integration. It requires that SVN executable is available on the remote node, and authentication is working on that node. Inside the job you will then need to define the SVN URL. The first time Hudson will check out under the remote directory / <job name>, and once checked out, it will do updates on this directory. When you have multiple jobs on the same node, Hudson would create separate SVN working directories for each job. To avoid this, you can use symbolic links to have one single physical directory only.

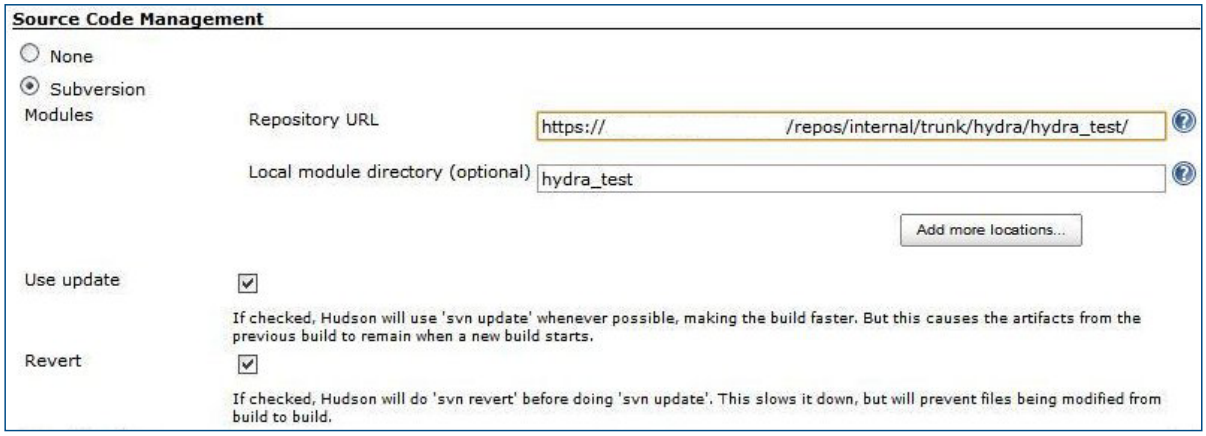


Image8: SVN repositories

init_server

The init server job helps to clean up and restart the application server (websphere, weblogic, jboss, tomcat). It was not easy to get this one stable, as the ANT job that starts the server on the remote node got killed once the parent Hudson job returned to server. The solution here is to add an environment variable BUILD_ID=dontKillMe to this job, so that child jobs stays alive.

The logic that defines which server needs to be started, and where the server is located, is actually not visible here, but hidden in our test framework. We just call the ANT target clean-start-server, and the rest will be handled locally on the node. In future, we want to integrate Cargo into our ANT task, as it allows for easy server start and deployments.

Install

The install job installs the software, compiles the updated test software, configures the client-server set-up and deploys the server part on the application server. All these tasks have been integrated into a single ANT job "ant testenvir", so the Hudson job just needs to call this ANT target, with some additional environment specific ANT properties.

Runtest

Our runtest jobs do the real test execution. We use separate build steps in one single job:

- On Windows we call a Windows batch script to map network drives
- We call ANT target "setup-minimal-audittrail" to set up our database
- We call ANT target "runsuite" to start our test
- We call ANT target "load-all" to load the results of the unit tests into a database. We have some Java jobs that parse the unit .xml files, store them into a database schema to facilitate good comparison and reporting afterwards.

The ANT jobs above all require their own specific environment settings, which we supply as a parameter in the ANT properties. Each node will get its own environment setting.

Report

The Hudson job "report_reg_trunk" has a script execution build step, that starts a Python script. The script reads test results from the database, both for the current and for the previous version, and writes output to the console. The job also has an additional email configuration that creates a mail with the console output and sends it to the QA department.

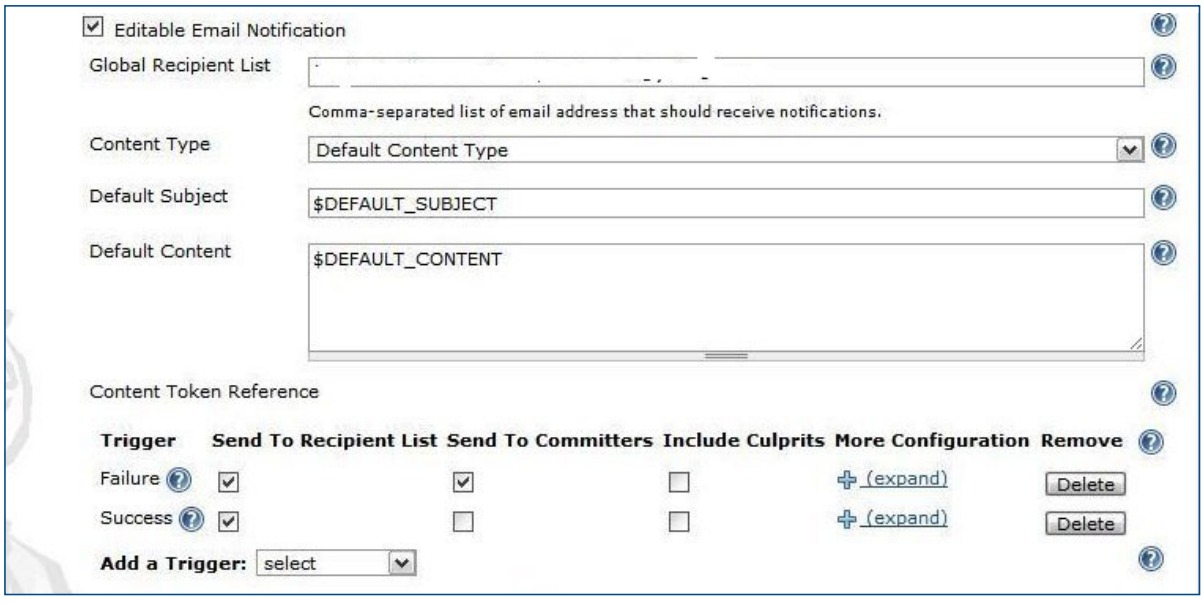


Image9: Generate report

Test report

The result of the testing is sent by mail. The report shows failures and errors per test suite, for each individual environment. It also compares the results of this build against the results of the previous build. Analysis of the report still takes time, but the comparison helps.

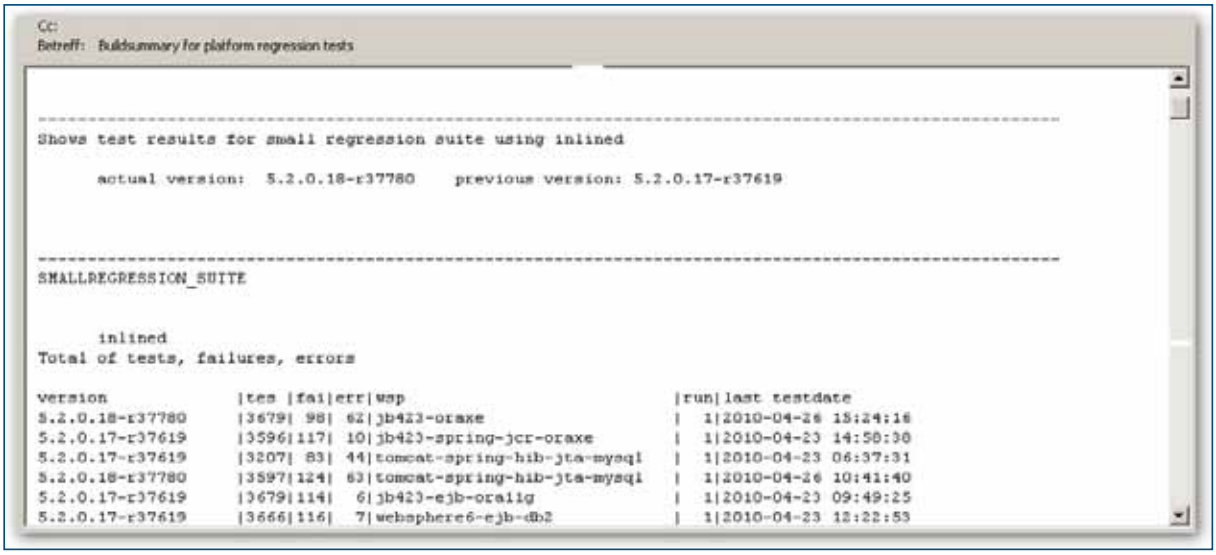


Image10: Test Report

Summary

Our tests now get started automatically, and reports arrive per mail. We have saved lots of time, and also became less dependent on the availability of the build engineer. Of course, we still need this expertise, as even automated tests can break, and then need to be repaired. The nice thing with the chain of jobs is that even when they are broken, you can see where it broke, fix it, and continue running from where the job stopped.

Nice to haves

Apart from our test automation, we added a few nice utility jobs to our Hudson setup. On our Windows and Linux test machines, we added scripts that create zipped backups, and scripts that check resources of the OS. We configured Hudson jobs for these scripts, and start those via a cron trigger. In that way we have automatic backups, and get daily health overviews by mail.

Biography

Manik Thakur is QA Engineer at SunGard Technology Services, Pune (India). He holds an MCA degree and has 5yrs of experience in developing, white box testing and deployment configuration. His special interest is in R&D.

Jan Kester is Director of QA at Sungard Infinity, Frankfurt. He has been active# since 1993, and has worked for SunGard since 2005 (formerly Carnot). His interest is in test automation in order to gain efficiency and quality in software QA processes. His knowledge involves Java, application servers, databases and OS.

Load and Performance Testing for Software Applications

by Hansjörg Senn

Performance might not be the first thing you think of when using a software application, but at the latest when software response time taxes your patience you will think about it. This article shows you the motivation for performance testing and points out the topics you have to take into account. Besides the performance objectives, the use of adequate tools and the appropriate strategy and procedures are other important aspects. If you are interested in these topics, please read on.

1. The purpose of performance tests

Do you remember your last performance problem? I remember my last performance problem well, and I also remember that fortunately the performance bottleneck was discovered in the testing and not in operational phase, which simplified the situation at the time. For most software development projects, performance is a crucial success factor. Project management should plan for performance even in the initial design and development phases, because with the first production run performance will be on the agenda at the latest. Most web applications have a good performance as long as only a few users are active. But what happens if a large number (e.g. thousands) of users work with the application simultaneously? At this point „Load & Performance Testing (L&P)" becomes a major issue. By simulating a large number of web users, we can generate load and network traffic on the system, and the application as well as various servers are driven to their performance limits.

There are several reasons to perform L&P tests:

- There is no 100% certainty that multi-user software makes all its functions and methods available to all users, and is therefore multi-user capable. But with performance testing it is possible to find a high percentage of the faults which were not found by individual manual testing, independent of whether unit tests or system tests were performed.
- Performance testing allows a realistic check of the configuration of interfaces to other systems or subsystems and computers. In addition, performance tests can also detect bottlenecks in host systems..

Performance tests essentially concern the following topics:

- Consumption of resources
- Response times
- Large load / number of users

Especially people from operations are mainly interested in the system resource consumption. They want to know what to expect with the introduction of an application to the production system. The adequacy of servers and instances is of interest. On the other hand, application users are interested in response times. With a large load created by many users, there will be an increase in response times and resource consumption.

2. New challenges

Application development used to be done using machine-oriented programming languages. This way the programmer had a more precise view of the target platform, on which the application code was carried out. For this reason, he could see certain performance-related risks more clearly and also earlier than today. New methodologies and technologies promote the use of distributed applications. This offers high flexibility for the interaction between components and for the composition of services to construct application systems. On the other hand, the new technologies are increasingly abstract and complex, which leads to an almost infinite number of possible combinations. This increases the risk of hidden performance problems.

3. Test types

Contrary to the various functional test procedures for testing the functional and technical correctness of software code (e.g. during unit testing, integration testing, regression testing), load and performance testing belongs to the category of non-functional test procedures. The technical literature [1] generally differentiates between the following types of non-functional tests:

- Performance test
- Load test
- Volume test / mass test
- Stress test

Of course, it strongly depends on the test objectives, which of the non-functional test types are to be accomplished. With the above-mentioned test types we will be able to detect faults in the following areas:

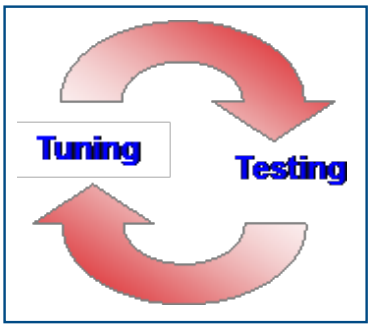
- Memory problems
- Performance issues
- Concurrency problems
- Excessive use of system resources
- Lack of storage space

From now on, we will summarize these kinds of tests under the generic terms „load and performance tests" or simply „performance tests".

4. Lifecycle of performance tests

An important basis of performance testing and the tuning of a system is the lifecycle of testing.

1. Each monitor which is installed and each logging mechanism which is used to measure the performance of a system has an effect on the system itself, whether intended or not. Therefore each monitor is also an interference factor. The aim should be to keep the impact of the monitors on the system as small as possible without any negative effects on the quality of measurement results.
2. Every test run is different. Even with a complete conformity of all configuration files and settings, we will receive different test results from one test run to another. Therefore the aim is to keep the deviations as small as possible.
3. After each tuning and after each parameter change, a new test run is required. Furthermore, the results must be analyzed and the system optimized and/or repaired after each test. If more than one parameter setting is modified at a time, we run the risk of not being able to understand why the system or some modules have become slower or faster. In the worst case, the effects of parameter changes cancel each other out and system behavior remains unchanged.
4. This cycle (testing, tuning, testing,...) is a never-ending story and could be continued indefinitely without reaching the maximum system performance. Thus it is important to find a trade-off, i.e. a good ratio between effort and yield.



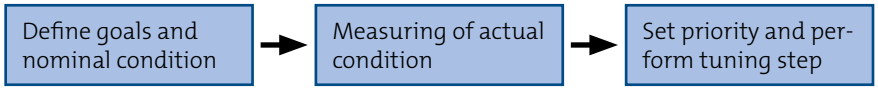
5. Procedure of load and performance testing

Test procedures are an essential part of quality assurance in the software development process. This applies to both the functional and the non-functional requirements of a system. Load and performance tests are an essential part of the non-functional test requirements of a system.

A very important point of planning is therefore to...

Define your objectives before taking any action!

In practice this rule often is violated, especially if the project is under time pressure. Therefore the following procedure is recommended:



5.1 Definition of goals and specification of metrics (benchmarks)

It is important to define the benchmarks before performing the tests. The benchmarks are the basis for professional optimization and for the selection of the appropriate test techniques. The benchmark should fit the application's requirements, i.e. a complex intranet application should not have to deal with e.g. 10,000 concurrent users. The following metrics are useful:

- Maximum number of users
- Maximum number of simultaneously logged-in users
- Average user think-time per page
- Average of max. response time per page
- Number of transactions per time unit (throughput)

In addition, at least the following conditions should be met:

- Test data based on realistic data
- Reproducible baseline, i.e. a recoverable baseline for every test run (regression), so that we can guarantee the comparability of different test runs
- Realistic test environment to obtain test results that are relevant.

Performance requirements must be defined during the specification phase and not only during the acceptance test phase.

Performance requirements should be checked for consistency and validity. As a guideline, we recommend using the so-called SMART criteria (Mannion and Keepence 1995, [2]). SMART is an acronym and means:

Specific

Are the requirements formulated in a clear and consistent way, so that they can be understood by the contractor?

Measurable

Are there measurement procedures to identify and analyze the appropriate performance data?

Attainable

The requirements can be achieved in theory and are not impossible to physically implement.

Relevant

Is it possible to meet the demands in practice, or are they contrary to basic conditions (time, budget, hardware, etc.)?

Traceable

Does the implemented system meet the requirements? This means that the performance management is able to monitor and control the achievement of performance requirements with suitable test methods.

5.2 Test data

Test data is a very important topic, which is often neglected. Professional performance testing should be done with realistic data volumes, i.e. volumes which are comparable to production data volumes. In the majority of cases it is not possible to get test data from the production system, either because the application is newly developed or due to the protection of data privacy. In such cases, the mass of test data has to be generated synthetically. In practice, such mass data is generated on the basis of a functionally correct data subset (e.g. with SQL-scripts). The following graph illustrates this approach.

limited places

testing experience

Knowledge Transfer

Hands-on Agile Acceptance Testing and Specification by Example

3-day tutorial with

Gojko Adzic

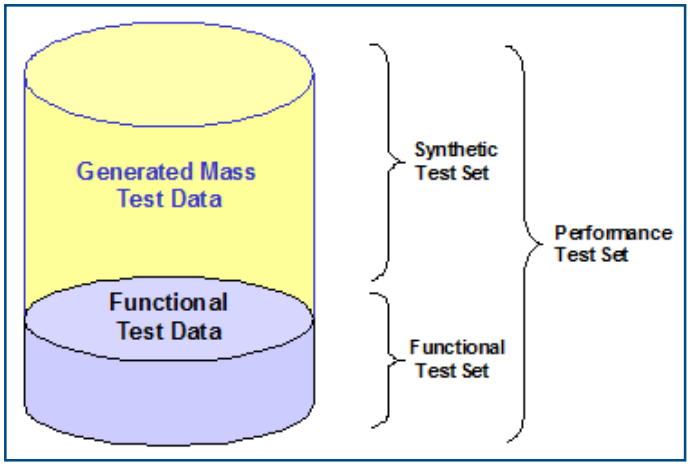
July 7-9, 2010 in Berlin, Germany

This three day workshop immerses the participants into a project driven by Specification by Example and Agile Acceptance Testing. Through facilitated exercises and discussion, you will learn how to bridge the communication gap between stakeholders and implementation teams, build quality into software from the start, design, develop and deliver systems fit for purpose.

This workshop is aimed at testers, business analysts and developers. It combines an introduction to Specification by Example and Agile Acceptance Testing, a set of exercises to help you get started with FitNesse - the most popular tool for agile acceptance testing - and a full day of working on realistic domain examples taken from your recent projects or a future phases of projects. This ensures that you gain real-world experience, enabling you to kick-starting internal adoption of these practices in your team.

www.testingexperience.com/knowledge_transfer.html

1250,- €



Realistic data volumes are the basis for meaningful performance tests, because database-related performance bottlenecks can often only be detected with a mass data load. The setup of appropriate test databases can be tool-supported (e.g. with Data Factory from Quest).

5.3 Reporting and presentation of test results

For reporting and presentation of L&P test results, it is important to know in advance how and which data has to be logged. There is no general procedure for data analysis and data collection, because this is highly dependent on the information needs of the measurement plan. In any case, the following data should be part of every test reporting:

- Statistics on response times measured on the client side compared to system load.
- Statistics on throughput as a function of the system load.
- Number of faults occurring during test
- Utilization of processors and disk volumes in the test environment.
- Network utilization and network latency.

With each performance test you will obtain large data volumes for analysis. Therefore it is strongly recommended to use tools for data handling, because we will often not find the relevant information in a single data set but in the relationships between large amounts of data. Well-established technical tools for analyzing and processing of data and handling data correlation apply statistical methods, clustering techniques for data aggregation and techniques for data filtering. The results can be saved as MS-Excel spreadsheets. The reports must contain both the actual test results and all relevant system parameters. Of course, graphical reports are useful and essential to get a quick outline on the test results.

5.4 Organizational aspects

To ensure a smooth flow of the performance test, some organizational measures have to be taken. Sometimes it is necessary to perform L&P-tests outside of normal working hours. Then it has to be ensured that access to workplaces and server rooms is guaranteed. The people involved must be on site or must at least be reachable. These kinds of issue are best organized with the help of checklists.

A very important point regarding the team organization is the coordination with all parties involved. These are usually:

- Developers
- External / internal testers
- System engineers
- Project manager

Often, performance tests are only meaningful if they are tested in production environments. In this case we must bear in mind that, inevitably, other systems can be influenced, e.g. by network load, database access, etc.

For the implementation and monitoring of Load & Performance testing, qualified specialized staff must be available. This is also because of the special tools which are often used.

5.5 Test tools

For load and performance tests, appropriate tools are essential. These are primarily tools for performance monitoring, for load generation and possibly for data analysis, visualization and report generation. In recent years the market of performance testing tools has changed significantly. Some tool providers no longer exist or were bought by a competitor, while some have established themselves as newcomers. The choice of adequate test tools is a very important issue. It would therefore be useful to test the tools in a pilot project before rolling out in your organization.

6. Evaluation of L&P test tools

New methodologies and technologies promote the use of distributed applications. On the one hand, this provides high flexibility for the interaction between components and for the composition of services for application systems, but it also results in high complexity of the systems. Therefore it is a challenging task to evaluate the suitable tool for L&P tests. In [1] (Florian Seyfert, "Testwerkzeuge für Last- & Performancetests") we will find an evaluation method based on a cost-benefit analysis ("Nutzwertanalyse", C. Zangemeister [3]), which is a useful method to identify adequate tools.

6.1 Setup requirements catalogue

The requirements catalogue consists of two parts:

- Pre-selection criteria (ko-criteria)
- Mandatory criteria

6.1.1 Pre-selection criteria (ko-criteria)

In a first step, the selection of the possible test tools is restricted by pre-selection criteria (ko-criteria). These are as follows:

ko-1: The tool must be suitable for load and performance testing

ko-2: The tool must be available in the local market

ko-3: A free demo version must be available

ko-4: The product must be up-to-date (not older than 5 years)

There are a number of tools listed in a study by IDC[1] (International Data Corporation, 2006). This list is the basis for pre-selection.

6.1.2 Mandatory criteria and weightings

The mandatory criteria were determined by consulting reference books and by a survey among experts. The weightings are as follows:

5 = extremely important

4 = very important

3 = important

2 = less important

1 = unimportant

The mandatory criteria were divided into two parts, a theoretical and a practical part. The theoretical part rates the provider's information, mainly based on white papers or other product information. The practical part is an assessment of the practical tests runs with the different tools.

1 International Data Corporation (IDC), with U.S. headquarters in Framingham, is an international consulting and event supplier in the fields of IT and telecommunications industry with offices in about 40 countries.

Altogether the criteria are grouped into 6 themes, the first two (A, B) represent the theoretical part, the others (C, D, E, F) belong to the practical part:

A: Provider information	(weighting: 4)
B: Product information	(weighting: 4)
C: Scripting	(weighting: 5)
D: Scenario design	(weighting: 5)
E: Scenario process flow	(weighting: 4)
F: Scenario analysis	(weighting: 5)

6.2 Test tools and market study

An IDC study (see [1], Florian Seyfert, "Testwerkzeuge für Last- & Performancetests") did a global market study of current L&P-test tools for the period from 2005 to 2010. The study was based on market figures from 2004 to 2005 and analyses sales trends of manufacturers and the growth of the market. At least two of the following requirements must be fulfilled:

- Annual sales in 2005 greater than 1 million U.S. Dollar
- Market share above 1% ...
- ... or growth in sales greater than 15%

Based on the above-mentioned requirements, the manufacturers list is as follows:

- HP / Mercury
- IBM
- Compuware Corp.
- Empirix
- Segue Software (since 2006 Borland)
- Borland Software Corp.
- RadView Software

Finally, only tools that pass the ko-criteria are included in the shortlist, which is as follows:

- Loadrunner (HP / Mercury)
- e-Load (Empirix)
- SilkPerformer (Borland)
- Rational Performer Tester (IBM)

The open-source testing tool OpenSTA was also considered in the evaluation, but sometimes it was difficult to apply certain criteria to the tool.

6.3 Test conditions

For each tool a test script was implemented to perform the practical test. This is the basis for rating criteria C to F:

C: Scripting

D: Scenario design

E: Scenario process flow

F: Scenario analysis

Please note that „practical test" does not mean a real performance test in this context, but test for evaluation of L&P-test tools. Therefore it is extremely important that the test conditions for each tool are the same, i.e. identical test environment and test scenarios. A PC was used as the test environment and the test scenario was a web application for flight reservations, which was launched on a local web server. Six different users logged-in one after the other (ramp-up) to book different flights. The measurement focus was on throughput (RPS, requests or hits per second) and the response times of transactions. The user think-times were from 3 to 18 sec, depending on the web site on which the test was done. Each scenario was repeated for each tool under the same conditions in order to identify possible discrepancies. In addition, the test scenario included the simulation of a severe failure to verify how each tool handles this case.

6.4 Evaluation result

Referring to [1] the evaluation result, the test winner was Loadrunner. However, if you are not prepared to pay the relatively high license costs, e-Load and SilkPerformer may be an alternative.

The ranking list is:

1. Loadrunner
2. e-Load
3. SilkPerformer
4. Rational Performer Tester
5. OpenSTA

When used by highly qualified experts, OpenSTA may also be an alternative, particularly in terms of cost. Please note, however that its functionality is limited in contrast to the winners.

Basically for each project we should define the performance objectives in detail to decide which tool is suited best. The present evaluation was done in 2007. Therefore, it is always advisable to check the current version of any tool concerning its suitability for a project. However, future versions of the tools will not offer new features, but will be enhanced in terms of stability and design.

7. Conclusion

L&P testing is a non-functional test type which should not be underestimated. L&P testing should be taken into consideration early in the project, as early as in the design phase. New methodologies and technologies provide increasing opportunities, but also more complexity for application systems. This fact increases the risk of hidden performance problems. L&P-testing is no one-man-show. In fact, the cooperation of a professional team of experts is necessary. The definition of the objectives is as important as a structured approach. In this context, we would like to point out the so-called SMART criteria (Mannion and Keepence 1995, [2]). L&P-testing is a cycle of "Testing - Tuning - Testing...", and it is very important to change only one parameter at a time to see what the result of this action is. Moreover, there are two other issues that are also significant: these are „test data" and "data analysis / reporting". There is no L&P testing without tool support. Therefore this paper gives a rough overview of well-established tools available in the market and an evaluation result. In any case, it is essential to verify whether a tool is suitable for a specific application and for the associated performance objectives.

8. Bibliography

[1] Florian Seyfert, "Testwerkzeuge für Last- & Performancetests", VDM Verlag Dr. Müller,

January 2008, ISBN: 978-3-8364-5559-6

[2] Mannion and Keepence, SMART criteria, 1995

http://www.win.tue.nl/~wstomv/edu/2ip30/references/smart-requirements.pdf

[3] C. Zangemeister, "Nutzwertanalyse in der Systemtechnik", Wittemannsche Buchhandlung, 1976

Biography

Hansjörg Senn holds the Diploma in Electrical Engineering from the University (Fridericiana) of Karlsruhe (Germany). Since 1996, he has been working for COMIT AG (Zürich) as an expert in the area of software testing (Test and Project Manager). He focuses on banking software.

Test Effort Estimation

by Jeetendra Kumar Swain

Testing is carried out primarily for unearthing all defects in the system and to prevent a defective product reaching the customers. Secondarily, testing is also carried out to convince the customer that the product conforms to and fulfills the specifications and provides the functionalities that were agreed.

Software testing is recognized as a very important activity in software development. Of late, the importance of independent testing – that is testing by persons not involved in the development of software – is increasingly being recognized, so much so, that software companies specialized only in software testing are established and are doing good business. Also significant is the fact that - as the complexity and size of software have increased significantly – so have the complexity of testing as well as the types of testing. As a result, there is a paradigm shift in the estimation of testing effort from being taken as a percentage of development effort, to an independent estimation of size and effort.

First - Definition of test estimation

Test estimation is the estimation of the testing size, testing effort, testing cost and testing schedule for a specified software testing project in a specified environment using defined methods, tools and techniques.

1. **Estimation** – as above
2. **Testing size** – the amount (quantity) of testing that needs to be carried out. Sometimes this may not be estimated, especially in embedded testing (i.e. testing that is embedded in the software development activity itself) and in cases where it is not necessary
3. **Testing effort** – the amount of effort in either person days or person hours necessary for conducting the tests
4. **Testing cost** – the expenses necessary for testing, including the cost of human effort
5. **Testing schedule** – the duration in calendar days or months that is necessary for conducting the tests

Now, having understood the definition of test estimation, we are ready for looking at the different approaches to test estimation.

Approaches to test effort estimation

The following approaches are available for test effort estimation:

1. Analogy based estimation
2. Software size based estimation
3. Test case enumeration based estimation
4. Task (activity) based estimation
5. Testing size based estimation

Software size based estimation

By the time a testing project is in its initiation phase, the software size is available. We now adopt the software size as the testing project size, and then assign a productivity figure (rate of achievement) for the software size to arrive at the required effort to execute the testing project.

Let us say that it takes 2 person hours to test a software size of one function point. Based on this norm, we can therefore determine the amount of effort required for the testing project based on the size the of software to be tested.

Suppose that the size of software to be tested is 1000 function points, then, based on the effort of 2 person hours per function point, we would require 2000 person hours for testing the software size of 1000 function points.

However, such norms for converting software size to effort are not available from any standards body, and they must therefore be developed and maintained within the organization using historical data, and adhering rigorously to a process. We have to derive this norm for all the software size measures used in the organization as well as maintain them.

Merits of software size based estimation

The merits of this technique are:

1. Very Simple to learn and use
2. Very fast – it takes very little time to arrive at the effort estimate
3. If the organization derives and maintains these norms using the right process, the results of effort estimation using this technique could be surprisingly accurate.

Drawbacks of software size based estimation

1. Too simplistic – not auditable
2. As the testing size is not available, productivity cannot be derived. However, testing productivity can be derived against the software size.
3. The amount of testing differs depending on the type of application, even though the size may be same. Stand-alone

Testing iT

Hunting Bugs...Opening Business

Testing IT is a consultancy firm focusing on Software Testing Practices and Quality Assurance.

Testing IT Consulting

"...There is always a better way of doing things, and we will find it..."

Testing Processes Creation and Innovation, Equipping Testing Teams, Tool Integration, Mentorship, Assessments and Audits.

Testing IT University

"...Education and Experience is simply the soul of a Tester..."

ISTQB® Certification, other courses based on our Knowledge Base (TIBoK©), In-class and e-Learning Training... A different but effective way of training.

Testing IT Units

"...TEAM = Together Everyone Achieves More..."

To Be Responsible for the Testing Process Within the Life-Cycle of the Development.

Hunting Bugs... Opening Business

Information:

+52 55 5566-3535

http://www.testingit.com.mx

info@testingit.com.mx

mexico@hastqb.org

Paseo de la Reforma 107, int.102,
Col. Tabacalera, México, D.F., 06030

software applications and web based software applications need different amounts of testing, even though their size may be the same. Hence the norm per software size may not be applicable in all cases.

4. Organizations have to keep meticulous records and employ full-time specialists to derive norms and maintain them. The timesheets need to be tailored to capture suitable data to derive these norms accurately. Data collection has to be rigorous

	<---------- Person Hours per unit size ---------->					
Type of Application	**Function Points**	**Use Case Points**	**Object Points**	**FPA Mark II**	**Feature Points**	**SSU**
Stand-alone	0.25	0.5	0.25	0.5	0.25	0.5
Client Server	0.3	0.65	0.3	0.65	0.3	0.65
3-tier Application	0.5	0.9	0.5	0.9	0.5	0.9
4-tier Application	0.75	1.1	0.75	1.1	0.75	1.1

To use this technique, we need to maintain a table like the sample table shown in the organization,. Please note that the values indicated in the table are by no means validated – they are indicative only.

Test case enumeration based estimation

The following steps describe this technique –

1. Enumerate the test cases – list all test cases
2. Estimate the testing effort required for each test case consistently (using either person hours or person days)
3. Use best case, normal case and worst case scenarios for estimating the effort needed for each test case
4. Compute the expected effort for each case using beta distribution

		<---------- Effort in PH ---------->			
Test Case ID	**Test Case Description**	**Best Case**	**Worst Case**	**Normal Case**	**Expected**
US1	**Setup test environment**				
US1.1	Check test environment	1	2	1.5	1.500
US2	Install screen recorder	0.75	1.5	1	1.042
US1.2	Ensure defect reporting mechanism	1.25	3	2	2.042
UI1	**Login screen on IE**				
UI1.1	Correct login	0.05	0.2	0.1	0.108
UI2	Wrong ID and correct password	0.07	0.2	0.1	0.112
UI1.2	Correct ID and wrong password	0.07	0.2	0.1	0.112
UI3	Forgot password functionality	0.15	0.3	0.2	0.208
UF2	**Login screen on Firefox**				
UF2.1	Correct login	0.05	0.2	0.1	0.108
UF3	Wrong ID and correct password	0.07	0.2	0.1	0.112
UF2.2	Correct ID and wrong password	0.07	0.2	0.1	0.112
UF4	Forgot password functionality	0.15	0.3	0.2	0.208
	Total effort estimate	3.680	8.300	5.500	5.663

Best Case + Worst Case + (4 * Normal Case) / 6

Sum up the following times–

1. **Expected times** to get the expected effort estimate for the project
2. **Best-Case times** to obtain the best-case effort estimate
3. **Worst-Case** times to obtain the worst-case effort estimate
4. **Normal-Case** times to obtain the normal-case effort estimate

The table below is an example of test effort estimation using test case enumeration (PH = person hours).

Merits of test case enumeration based estimation

The merits of this technique are:

1. Auditable estimate – the estimate has sufficient detail so that another peer can review the estimate and ensure that the estimate is comprehensive and as accurate as possible
2. Fairly accurate – accuracy is ensured as all test cases are enumerated three times to arrive at the expected effort
3. Progress monitoring is facilitated by marking the test cases that have been completed, and the percentage of completed test cases can be easily be computed

- Facilitates giving a range of values for the estimates to aid in the decision-making process. The project can be executed with a minimum effort of person hours, or a maximum effort of person hours, or with an expected effort of person hours. These different estimates allow the decision makers to set negotiation margins in their quotes.

Drawbacks of test case enumeration based estimation

1. No testing size – hence productivity cannot be derived
2. All test cases and attendant overheads need to be enumerated, which means that it will take time to complete the estimation.

Task (activity) based estimation

This method looks at the project from the standpoint of tasks to be performed in executing the project. Any project is executed in phases. Phases in a testing project could be

1. Project initiation
2. Project planning
3. Test planning
4. Test case design
5. Set up test environment
6. Conduct testing
   a. Integration testing
   b. System testing
   c. Load testing
   d. etc.
7. Log and report test results

Berlin, Germany

IT Law
Contract Law

German
English
Spanish
French

www.kanzlei-hilterscheid.de
info@kanzlei-hilterscheid.de

k a n z l e i h i l t e r s c h e i d

8. Regression testing
9. Prepare test report
10. Project closure

Of course, the phases may differ from project to project and from organization to organization. Also, each of these phases could be further broken down into tasks. Here is an example for the project initiation and project planning phases –

Phase 1 – Project initiation

1. Study the scope of testing and obtain clarifications, if necessary
2. Identify the project (test) manager
3. Retrieve data of past similar projects and make it part of the project dossier
4. Prepare PIN (Project Initiation Note) and obtain approval
5. Conduct a project kick-off meeting and hand over project dossier to project (test) manager

Phase 2 – Project planning

1. Prepare effort estimates
2. Determine resource requirements
3. Raise resource request forms
4. Prepare project management plan
5. Prepare configuration management plan
6. Prepare quality assurance plan
7. Arrange peer review of project plans
8. Obtain approval for project plans
9. Baseline project plans

We can break down each of the phases into their constituent tasks. Now using these tasks, we can carry out the test effort estimation.

The following are the steps in task based effort estimation –

1. Assign durations for each of the tasks consistently (using either person hours or person days)
2. Use three time estimates – best case, worst case and normal case for each of the tasks
3. Compute the expected time using the formula

   [Best Case + Worst Case + (4* Normal Case)]/6
4. Make adjustments for project complexity, familiarity with the platform, skill of the developers, usage of tools
5. Sum up the total effort estimate of the project
6. Use the Delphi technique to validate the estimate, if in doubt or if felt necessary

The table below gives an example of task based effort estimation for testing projects

			<---------- Effort in PH ---------->			
Task ID	**Phase**	**Task**	**Best Case**	**Worst Case**	**Normal Case**	**Expected**
1	Test planning	Study specifications	2	5	3	3.167
2	Test planning	Determine types of tests to be executed	0.5	1	0.65	0.683
3	Test planning	Determine test environment	0.5	1	0.65	0.792
4	Test planning	Estimate testing project size, effort, cost & schedule	2	4	3	3.500
5	Test planning	Determine team size	0.5	1.25	0.75	0.917
6	Test planning	Review of estimation	1	2	1.5	1.750
7	Test planning	Approval of estimation	0.5	2	0.75	1.042
8	Design test cases	Design test cases for Module 1	5	8	6	7.167
9	Design test cases	Design test cases for Module 2	6	8	7	8.333
10	Design test cases	Design test cases for Module 3	4	6	5	5.833
11	Conduct tests	Conduct tests for Module 1	15	18	16	18.833
12	Conduct tests	Conduct tests for Module 2	16	19	17	20.000
13	Conduct tests	Conduct tests for Module 3	14	16	15	17.500
14	Defect report	Defect report for Module 1	1	2	1.5	1.750
		Total effort estimate	68.000	94.250	77.800	91.267

1. **Expected times** to get the expected effort estimate for the project
2. **Best-Case** times to obtain the best-case effort estimate
3. **Worst-Case** times to obtain the worst-case effort estimate
4. **Normal-Case** times to obtain the normal-case effort estimate

Merits of task based effort estimation for testing projects

The merits of this technique are:

1. This most closely reflects the way projects are executed
2. This technique takes into consideration all the activities that are performed and gives effort estimates as accurately as possible
3. It has adequate detail which makes it amenable for reviewing and auditing and postmortem analysis by comparing with the actual values
4. Estimate is simple and easy to prepare
5. Makes project progress monitoring easy by marking the tasks that have been completed, and the percentage of completion can be easily computed
6. Also suitable for use in analogy based test effort estimation.

Drawbacks of task based effort estimation for testing projects

The testing size is not computed; therefore the testing productivity cannot be arrived at.

Issues in sizing the testing projects

When we attempt to specify a unit of measure, there must be a clear definition of what is included in it. Secondly, there must be a means to measure the size. Even though they do not need to be identical, there must be some uniformity in the practices of testing, namely in the preparation of test plans, the comprehensiveness of test cases, the types of testing carried out and the

Agile
TESTING DAYS

Agile Testing Days 2010

4–7 October 2010, Berlin

The **Agile Testing Days** is the European conference for the worldwide professionals involved in the agile world.

We chose Berlin, one of the best-connected capitals in Europe to host this event due to the affordability/service ratio for hotels and flights. Berlin also offers a lot of fabulous sights to visit in your spare time.

Please have a look at the program at **www.agiletestingdays.com** and enjoy the conference!

Agile TESTING DAY

Berlin, Germany

October 4

Tutorials


Lisa Crispin


Janet Gregory


Elisabeth Hendrickson


Stuart Reid


Isabel Evans


Linda Rising


Michael Bolton


Jennitta Andrea


Anko Tijman


Eric Jimmink


Pekka Klärck


Juha Rantanen


Janne Härkönen

October 5

Conference

Day 1	Track 1	Track 2	Track 3	Track 4	Track 5 – Vendor Track
08:00	Registration				
09:25	Opening				
09:30	Keynote – Lisa Crispin				
10:30	Incremental Scenario Testing: Beyond Exploratory Testing *Matthias Ratert*	Experiences on test planning practices in Agile mode of operation *Eveliina Vuolli*	Testability and Agility – "Get your quality for nothing and your checks for free" *Mike Scott*	Testing Web Applications in practice using Robot Framework, Selenium and Hudson *Thomas Jaspers*	Continuous Deployment and Agile Testing *Alexander Grosse*
11:30	Break				
11:50	Test Driven Migration of Complex Systems *Dr. Martin Wagner and Thomas Scherm*	Making GUI Testing Agile and Productive *Geoffrey Bache*	Real World Test Driven Development *Emanuele DelBono*	Acceptance Test Driven Development using Robot Framework *Pekka Klärck*	Talk 5.2
12:50	Lunch				
14:20	Keynote – Linda Rising				
15:20	Test Center and agile testers – a contradiction? *Dr. Erhardt Wunderlich*	Reinvigorate Your Project Retrospectives *Jennitta Andrea*	Error-driven development *Alexandra Imrie*	Flexible testing environments in the cloud *Jonas Hermansson*	Talk 5.3
16:20	Break				
16:40	Solving the puzzles of agile testing *Matthew Steer*	Structures kill testing creativity *Rob Lambert*	Integration Test in agile development *Anne Kramer*	Waterfall to an Agile Testing Culture *Ray Arell*	Talk 5.4
17:40	Keynote – Elisabeth Hendrickson				
19:00	Chill Out/Event				

Agile TESTING DAY
Berlin, Germany

October 6

Conference

Day 2	Track 1	Track 2	Track 3	Track 4	Track 5 – Vendor Track
08:00	Registration				
09:25	Opening				
09:30	Keynote – Michael Bolton				
10:30	Top 10 reasons why teams fail with ATDD, and how to avoid them *Gojko Adzic*	Agile hits Formal – Development meets Testing *Matthias Zieger*	A lucky shot at agile? *Zeger Van Hese*	The Leaner Tester: Providing true Quality Assurance *Hemal Kuntawala*	Talk 5.1
11:30	Break				
11:50	TDD vs BDD: from developers to customers *Alessandro Melchiori*	How can the use of the People Capability Maturity Model® (People CMM®) improve your agile test process? *Cecile Davis*	Mitigating Agile Testing Pitfalls *Anko Tijman*	Maximizing Feedback – On the importance of Feedback in an Agile Life Cycle *Lior Friedman*	Talk 5.2
12:50	Lunch				
14:20	Keynote – Stuart Reid				
15:20	Alternative paths for self-education in software testing *Cirilo Wortel*	Hitting a Moving Target – Fixing Quality on Unfixed Scope *David Evans*	Fully Integrated Efficiency Testing in Agile Environment *Ralph van Roosmalen*	The Flexible Art of Managing a Multi-layered Agile team *Shoubhik Sanyal*	Talk 5.3
16:20	Break				
16:40	Mastering Start-up Chaos: Implementing Agile Testing on the Fly *Christiane Philipps*	Building Agile testing culture *Emily Bache and Fredrik Wendt*	Implementing collective test ownership *Eric Jimminik*	Alternative paths for self-education in software testing *Markus Gärtner*	Talk 5.4
17:40	Keynote – Janet Gregory				
19:00	Chill Out/Event				

October 7

Open Space

Open Space hosted by **Brett L. Schuchert**.

Day 3	Track
08:00	Registration
09:25	Opening
09:30	Keynote – Isabel Evans
10:30	*Open Space*
11:30	Break
11:50	*Open Space*
12:50	Lunch
14:20	Keynote – Jennitta Andrea
15:20	*Open Space*
17:20	Keynote – Tom Gilb
18:20	Closing Session

Exhibitors

Díaz Hilterscheid

testing experience

NOKIA OVI

iSQI International Software Quality Institute

Agile Record

Please fax this form to +49 (0)30 74 76 28 99
or go to http://www.agiletestingdays.com/onlinereg.html.

Agile TESTING DAY
Berlin, Germany

Participant

Company: _____
First Name: _____
Last Name: _____
Street: _____
Post Code: _____
City, State: _____
Country: _____
Phone/Fax: _____
E-mail: _____

Billing Address (if different from participant)

Company: _____
First Name: _____
Last Name: _____
Street: _____
Post Code: _____
City, State: _____
Country: _____
Phone/Fax: _____
E-mail: _____

Remarks/Code: _____

Tutorial (4 October 2010)

- ☐ Making Test Automation Work on Agile Teams *by Lisa Crispin*
- ☐ A Rapid Introduction to Rapid Software Testing *by Michael Bolton*
- ☐ A Tester's Guide to Navigating an Iteration *by Janet Gregory*
- ☐ Patterns for Improved Customer Interaction/Influence Strategies for Practitioners *by Linda Rising*
- ☐ The Foundations of Agile Software Development *by Jennitta Andrea*
- ☐ Testing2.0 – agile testing in practice *by Anko Tijman & Eric Jimmink*
- ☐ Managing Testing in Agile Projects *by Stuart Reid & Isabel Evans*
- ☐ Agile Transitions *by Elisabeth Hendrickson*
- ☐ Executable Requirements in Practice *by Pekka Klärck, Juha Rantanen & Janne Härkönen*

750,– EUR
(plus VAT)

Early Bird – register until June 30, 2010 to get the discount
625,– EUR
(plus VAT)

Conference (5–7 October 2010)

☐ 1 day (first day)	550,– EUR	☐ 3 days	1.350,– EUR
☐ 1 day (second day)	550,– EUR	☐ 2 days	1.100,– EUR
☐ 1 day (third day)	350,– EUR		

1.350,– EUR
(plus VAT)

Early Bird – register until June 30, 2010 to get the discount

1 day (first day)	*440,– EUR*	*3 days*	*1.080,– EUR*
1 day (second day)	*440,– EUR*	*2 days*	*880,– EUR*
1 day (third day)	*250,– EUR*		

1.080,– EUR
(plus VAT)

Included in the package: The participation at the exhibition, the social event and the catering during the event.

Notice of Cancellation
No fee is charged for cancellation up to 60 days prior to the event. Up to 15 days prior to the event a payment of 50 % of the course fee becomes due and after this a payment of 100 % of the course fee becomes due. An alternative participant can be designated at any time at no extra cost.

Settlement Date
Payment becomes due no later than at the start of the event.

Liability
Except in the event of premeditation or gross negligence, the course holders and Díaz & Hilterscheid GmbH reject any liability either for themselves or for those they employ. This particularly includes any damage which may occur as a result of computer viruses.

Applicable Law and Place of Jurisdiction
Berlin is considered to be the place of jurisdiction for exercising German law in all disputes arising from enrolling for or participating in events by Díaz & Hilterscheid GmbH.

Date _____ Signature, Company Stamp

availability of empirical data to normalize various situations to a common measure.

The following aspects must be taken into consideration –

1. **Type of application** – Stand-alone, client-server, web-based
2. **Type of testing** – White Box or Black Box
3. **Stage of testing** – unit, integration, system tests
4. **Purpose of testing** – Reliability, client acceptance, ensuring functionality
5. **Test case coverage** – how much is covered by test cases and how much is left to the intuition of the tester
6. **Definition of the granularity test case** – If one input field is tested with five input values, does this count as one test case or five test cases?
7. The **size of the test cases** at the unit, integration and system test levels vary. We need a normalization factor to bring them all to a common size
8. The impact of the **usage of tools** and the effort needed to program them
9. **Environmental factors** – the tester experience and knowledge, complexity of the application, resources (time and budget) allowed for testing, existence of a clean testing environment etc. and the impact of these factors on testing.
10. Existence of the practice of **code walkthrough** before testing software in the organization.

The literature and practices I have seen so far do not suggest that all these aspects are well considered and covered in defining the size measure for testing effort.

The question is: Is a size measure necessary to estimate testing effort? No, it is not. The testing effort can be estimated using other techniques as mentioned above.

However, size measure is important so that comparisons can be made between projects; and it is important to assess how reasonable the effort estimates are. It also facilitates computation of testing productivity (or rate of achievement).

Who needs test size estimation?

1. Testing organizations, whose mainline of business it is to test software of others and certify the products. Their objective is to ensure that the product meets the customer specifications and expectations. This set would carry out –
   a. Mainly Black Box testing
   b. Functional testing
   c. System testing
   d. Negative testing
   e. Regression testing
2. Customers who have entrusted their software development to a vendor. Their objective is to ensure that they are getting what they are paying for.

Sizing of a testing project

The term "test points" is catchy and perhaps the unit of measure for the estimation of testing size and effort. This term is being used by many and is popular for sizing software testing projects. Test points can be extracted from the software size estimates.

Test Point is a size measure for measuring the size of a software testing project and a Test Point is equivalent to a normalized test case. In this context, a test case has one input and one corresponding output.

It is common knowledge that test cases differ widely in terms of complexity and the activities necessary to execute them. Therefore, the test cases need to be normalized – just the way function points are normalized to one common measure using weighting factors. However, there are no uniformly agreed measures of normalizing the test cases to a common size. Also, the relation between other software size measures like function points or use case points etc. is not clear? Would it be fair to say that one adjusted function point results in one normalized test point? – Again, no universal agreement. Perhaps we may say that one adjusted function point results in 1,or 1.2, or 1.3, etc.

There are many types of testing performed on software. Is there a standard that says that these are the tests that should be included in a testing projects? I am afraid, there is no agreement on this either. Generally, although not necessarily always, a testing project would include integration testing, system testing and acceptance testing - all of which use the black-box testing technique.

Reality, however could be different.

The variety in applications, on which testing depends, is significantly large. And the method for normalization between various application types is not commonly agreed.

The types of testing carried out varies from project to project. There are no uniformly agreed types of testing to be carried out on any given project.

There is barely enough research and empirical data that would allow accurate guidelines to be drawn, as the profession of testing itself is very nascent#.

However, we may estimate test points converting the size estimate using a set of conversion factors into test points and adjust the test point size using various weights.

Weights

The following weights could be considered:

1. Application weight
2. Programming language weight
3. Weights for each type of testing, namely -
   a. Unit testing
   b. Integration testing
   c. System testing
   d. Acceptance testing (positive testing)
   e. Load testing
   f. Parallel testing
   g. Stress testing
   h. End-to-end testing
   i. Functional testing (negative testing)
   j. etc.

All weights are project-specific.

A test point has a weight equal to 1 when the combined weights of three tests, namely integration testing, system testing and acceptance testing is equal to 1. Or in other words: The sum of weights of these three test types cannot be more than 1 (or less than 1).

When other tests are added to the project, their weights may be assigned and added to test point weight.

We need to compile weight data for all these tests and maintain them in-house by comparing the estimated values with actual values at the end of every testing project after conducting a rigo-

Probador Certificado Nivel Básico

Tester profesional de Software

Formación para el Probador Certificado - Nivel Básico
de acuerdo al programa de estudios del ISTQBâ

República Argentina

Docente: Sergio Emanuel Cusmai

Co - Founder and QA Manager en QAustral S.A.
Gte. Gral. de Nimbuzz Argentina S.A.
Docente en diplomatura de Testing de Software de UTN - 2009.
Titular y Creador de la Diplomatura en Testing de Software de la UES XXI - 2007 y 2008.
(Primer diplomatura de testing avalada por el ministerio de Educación de Argentina).
Team Leader en Lastminute.com de Reino Unido en 2004/2006.
Premio a la mejor performance en Lastminute.com 2004.
Foundation Certificate in Software Testing by BCS - ISTQB. London – UK.
Nasper - Harvard Business school. Delhi – India.

Casa Central: + 54 0351 4817951 - **Tel Bs As Argentina:** 011 53685847 - **Tel Londres UK:** 00442033183814
Tel San Francisco. EEUU: 001 14157041515 - **E-mail: info@qaustral.com**

rous causal analysis in each case.

Testing tool usage is expected to reduce the effort, even though there are views that tools would not reduce the effort for the first iteration of testing. Well perhaps, but it really depends on the tool itself. Therefore the weight for tool usage also may be assigned suitably based on the tool itself and the project in hand. A weight of 1 for tools usage indicates that the tool would not have any impact on the effort required for testing. A weight of more than 1 indicates that the tool increases the testing effort and a weight of less than 1 indicates that the tool would reduce the testing effort.

If we include unit testing in the proposed tests for the project, we need to assign another weight for the programming language used for developing the code for the project. Here we mean independent unit testing carried out by a person who did not write the code in the first place. The reasons for this additional weight are –

1. Unit testing is white box testing – that is based on the code
2. The development environments for different languages varies, and so the amount of effort required for the testing project will differ

The following are the steps in computing the testing project size in test points. We are using the software size as the basic input to this model.

1. Use an existing software development size
2. Convert the software size into unadjusted test points (UTP) using a conversion factor which is based on the application type
3. Compute a composite weight factor (CWF)
   a. Sum up all individual weights of selected tests
   b. Multiply this by the application weight
   c. Multiply this by the language weight if unit testing is selected
   d. Multiply this by the tools weight if tool usage is selected
4. The unadjusted test points are multiplied by the CWF to obtain the testing size in test points size
5. The productivity factor indicates the amount of time for a test engineer to complete the testing of one test point
6. Testing effort in person hours is computed by multiplying the test point size by the productivity factor.

The table below illustrates the test points estimation:

Sl. No	Aspect	Test Points
1	Product size in FP	2500
2	Conversion factor (TP per FP)	4.5
3	Unadjusted test points	11250
4	Application weight for client-server application	1.1
5	Composite weight factor (CWF)	1.375
6	Adjusted test points	15468.75
7	Productivity factor in person hours / TP	0.2
8	Test effort in person hours	3093.75

The next table gives the various test weights used in computing the CWF in the above table:

	Test Weights	
1	Functional test	0.35
2	System test	0.3
3	Acceptance test	0.2
4	Virus-free test	0.2
5	Spy-ware-free test	0.2
6	Total weight	1.25

Merits of test point estimation

The merits of this technique are:

1. Size is estimated – this makes it amenable to productivity computation, comparison and benchmarking
2. Size makes it useful in analogy based estimation

Drawbacks of test point estimation

1. There is no universally accepted definition of what a test point is
2. Perhaps not as simple as other test effort estimation methods
3. No universally accepted or benchmarked data available on various weights used in the method. These have to be developed and maintained adhering to rigorous methodology and record keeping. This puts overheads on the organization
4. Timesheets have to be oriented for deriving the required data

Well, nothing that is good ever comes free or easily. This is also the case for test point estimation for sizing testing projects. One needs to spend effort and time to set the norms, as perhaps in any other case.

Final words about test effort estimation

This is obviously an area where further work is necessary. However, there are methods that make it possible to estimate the effort required for executing testing projects. Test points are slowly emerging for sizing software testing projects.

It is suggested that testing project is scheduled, just the way software development projects are scheduled, resources allocated and the schedule is re-worked with resource constraints, and only then the schedule and effort are committed.

It is also suggested that the presentation of the test effort estimate also be subjected to the format suggested for software development project estimates, in order to ensure that all aspects of estimation are communicated to the decision makers.

Biography

I am Jeetendra Swain and have 8+ years of experience in software testing in different domains. I have work experience as Test Lead in planning, executing and delivering software testing projects at HCL Technologies Limited. I am presently located at Sydney, working for a leading banking and finance account.
I completed my post graduation in Computer Sciences and graduation as B.Sc.
I am very friendly and sporty person, who enjoys the company of a team. Reading books and listening to music are my two favorite hobbies.

Certified Tester Training

in

Spain

Argentina

Mexico

Chile

info@certified-tester.es

An approach for reuse and customization of performance test scripts for web applications

by José Carréra & Uirá Kulesza

In this article we describe the results achieved by the research developed during my master's degree in software engineering, and will present an approach for reuse and customization of performance test scripts for web applications. Our study was developed with the main purpose of providing an alternate way to traditional software performance testing, currently based on manual test script code implementation using available automation tools.

We focused on performance due to its extreme importance as a non-functional requirement, especially for web applications, where underestimating it may cause severe consequences and financial loss to users/clients of the application.

Our work,which is based on software product-line concepts and practices, defines an approach composed of six steps, which is supported by a product derivation tool. It allows the identification of variabilities found among the different test scripts that were collected or implemented, and the establishment of models and templates that make it possible to derive new instances or test scripts focused on different scenarios for web applications. During the development of our approach, a case study was performed, which was composed of the analysis of test scripts developed for four different web applications with the intent of validating the proposed approach. In this article we will describe part of this case study, which will show you how our approach can be applied.

1. The defined approach

First we will describe the approach developed during our work, as an alternative to the problem of finding a way to implement test scripts to validate performance requirements of web applications.

After presenting our approach, we will describe how it was possible to validate the appoach by applying it along with a product derivation tool entitled GenArch, for web applications performance validation environment. We will describe how each step was conducted and the results observed at the end of the case study.

The proposed approach makes it possible to reuse test scripts for web applications, offering an alternative path for creating test scripts. We observed that by structuring and documenting test scripts used in previous projects, and analyzing them, specific test scripts could be generated (completely or partially), thus providing greater agility.

Our approach is structured in six steps that range from performance test type definition, along with creating and preparing test scripts to be reused, and finally supporting performance test scripts derivation and customization for specific applications. We describe where each of these steps can and must be checked, since we are not describing a sequential series of activities that must be done only once.

Below, each step of our approach is presented from a practical point of view to make it easier to understand and apply, and also to indicate where future modifications or adaptations might be needed for a different scenario.

Steps 1 to 4 shall preferably be executed by a domain engineer, who frequently will be the one who has greater knowledge regarding application's domain. Together with a test and a requirement engineer, the domain engineer may correctly analyze the test scripts and define which features shall be represented as variabilities in the derivation models. On the other hand, steps 5 and 6 shall be carried out by the application engineer or the test engineer from the project, who will be held responsible for determining the elements which will be included in the new test scripts, and for ensuring that these will be executed properly.

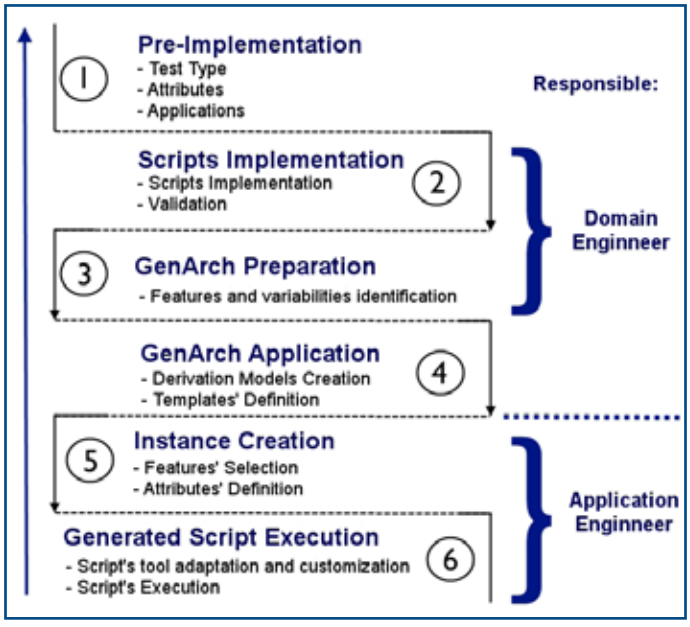


Figure 1 - Steps of the proposed approach

(1) Pre-implementation step:

During this step, we define which are the test types and attributes that will be verified, along with the used web applications. It is also important to determine the performance demands for the selected attributes and for which operation this feature shall be validated, and of course also any other important detail that might influence the test results. It's mandatory for the users to be present at this stage.

We shall define the test types to be validated, such as load, stress or performance tests, along with the attributes that will be verified for each of these (e.g. response time, bandwidth, etc.), according to the objectives that were set for each application. Usage will vary according to application's nature (e.g. search, login, form submission, etc.), and this will have an influence on all the work that will be done in subsequent steps. These features will be represented later on in the product derivation tool, which will make it possible to derive new test scripts or scenarios from the features representation for the specific application or for different applications.

(2) Scripts implementation:

After identifying performance test types to be in focus and the attributes (step 1), script implementation or identification of existing scripts are performed, so that each script shall relate to a feature defined in step 1. This task will enable the domain analysis to be executed in the next step. An example would be a set of web applications relating to the submission of forms.

In this step, we also validate the scripts for the respective applications, assuring their efficiency, and also reducing the possibility of future mistakes being introduced into the newly derived test scripts.

(3) Test scripts similarities and variabilities analysis:

Immediately after the defined and validated scripts from the previous steps have been made available, they are analyzed for the purpose of identifying similarities and variabilities among the scripts. This way, we can check if different test scripts that were implemented for the same test type perform similar operations (insert, remove, search) based on a similar attribute (e.g. response time).

Similarities and variabilities identification between these scripts allows us to correctly represent them in a product derivation tool. In our case, (using the GenArch derivation tool), this information is mapped to a set of models and templates.

(4) Derivation tools' artifacts specification:

During this step of our approach, we start working directly on creating the artifacts for the product derivation tool, and use all information gathered in the previous steps as a reference.. These artifacts will be used by the derivation tool to allow performance test scripts to be customized for specific applications needs.

With regard to the GenArch tool, three models need to be defined: feature, architecture and configuration. After creating these models, we need to define two further aspects:

- templates , which represent customizable test scripts; and
- constraints and dependencies between elements of the models regarding script generation.

(5) Automatic test scripts derivation for an application:

In this step, the engineer defines the desired test types along with the respective attributes. This is done, for all the features selected and values indicated in the previously created feature model. These settings are defined according to the desired test types for a specific application. After establishing this new configuration (feature selection), the engineer can start the automatic test script derivation process.

(6) Generated test script execution:

Executing the generated test scripts is the main focus in the last step of our approach. If the test script was only partially generated, we will first need to complete the missing information.

After making all necessary adjustments and customizations to adapt the test script to the selected automation tool, we have a complete test script ready for test execution. Finally, after test execution, results can be analyzed and the test scripts re-run as many times as needed, according to project's test plan definitions.

2. Trying out the approach

This section describes how we can apply the presented approach for reusing and customizing performance test scripts for web applications, by pointing out in general terms its characteristics and the activities that need to be performed in each step. Our approach is illustrated in a specific way, by using the following tools: (i) GenArch – product derivation tool (CIRILO, 2008); and (ii) JMeter – performance testing tool.

2.1 Test domain engineering

In the proposed approach, we must (in step 1) first define which test types will be run, and the attributes along with the applications available for testing. This task focuses on the test types of load, stress and performance tests, and considers response times, load levels and concurrency as attributes.

After that, test scripts need to be collected and implemented for the selected web applications (in step 2), allowing the test script generation. These test scripts have to be adjusted to assure that they will run performance validations for the selected attributes. Finally, the test scripts are validated on the respective tools to assure that they are correct and valid for being analyzed in the next step.

3.1.1 Common and variable features identification

During this phase (step 3) the analyses of the collected and adapted test scripts is started, which we called "Scripts' line" (figure 2). The main goal is to identify common and variable features among the chosen test types and the attributes for the selected web applications. To achieve this goal, implemented scripts are analyzed and studied in order to identify common and variable elements (figure 3). The findings will be utilized as elements of the required models.

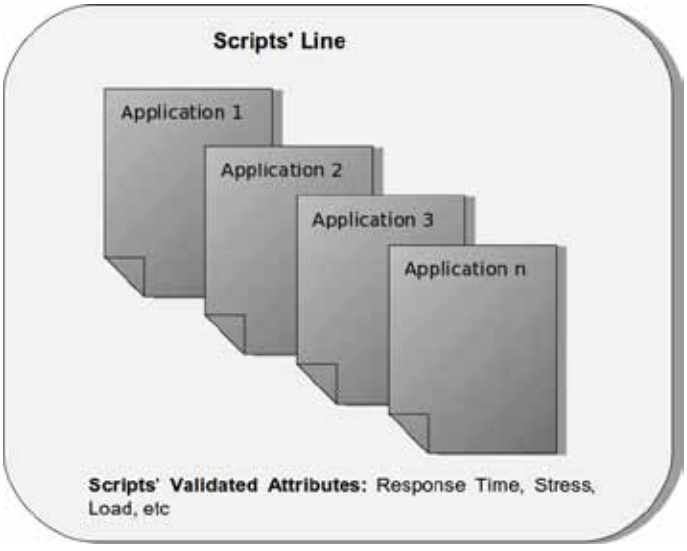


Figure 2 – Script's line representation

During the analysis a variety of common and variable elements can be identified among the test scripts, which will determine how applications behave in different situations and scenarios. This allows the identification of elements that represent variabilities and may be used for future test script derivation.

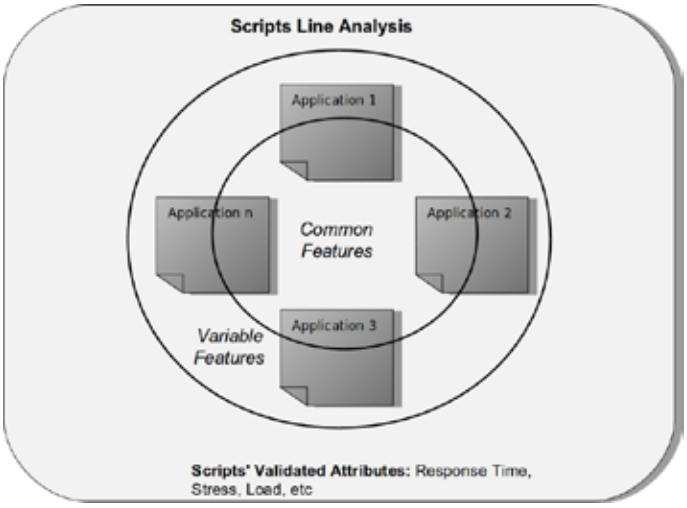


Figure 3 – Script line analysis

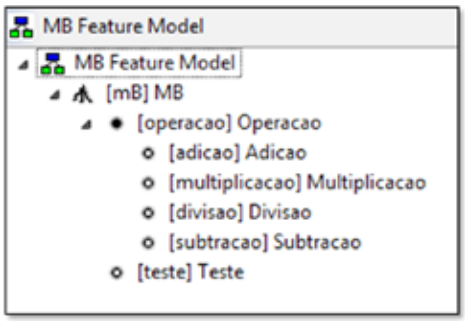
2.1.2 Creating derivation models

After identifying the test scripts' common and variable features, we start creating the derivation models (step 4), which is a task supported by GenArch plug-ins. In our research, we established that it is best to create these models manually, since code annotations used are restricted to Java applications on GenArch. This meant it was not possible to automatically generate an initial version using the tool. Models must be created and refined with the aim of allowing optimum simplicity and flexibility for the final test script generation.

These models are the central objects of the derivation tool used, and from these we will be able to derive test scripts according to our specifications. Therefore creating and refining the models is an essential task in order to successfully perform derivation. In the following section, we briefly describe the models used.

Feature Model

This represents all variabilities identified during the previously performed analysis of the test scripts. It is designed such that it uses a simple and direct language to allow test engineers to easily understand its elements and to determine during the instantiation phase (described later) the desired characteristics to be included in the new test scenario.



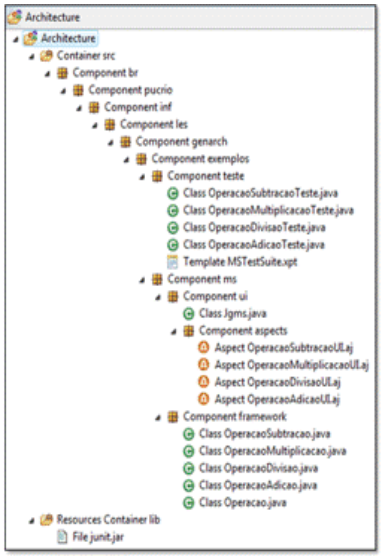
3.4 - Feature Model Example

Architecture Model

This model visually represents SPL's implementation elements. It does so in a way that allows mapping out the relation between implementation elements (solution space) and features found in the feature model (problem space). Like the other derivation models, this one must also be manually created based on the results of the analysis made in the previous step. This model is basically made up of a set of test script templates that are processed to generate tests for specific web applications.

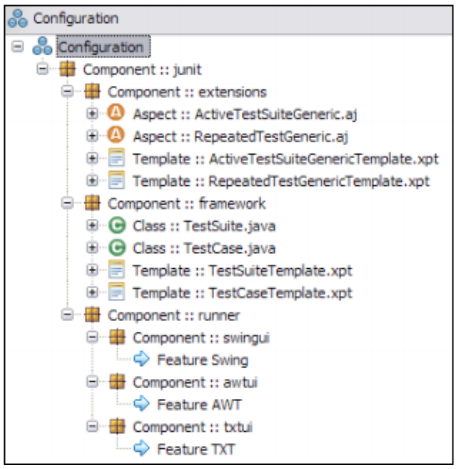
Configuration Model

Finally, we define the last model required for our approach, the so-called configuration model. This model specifies the existing relationships between the elements from SPL's architecture and feature models. It is considered a fundamental tool to connect the problem space (feature model) with the solution space (architecture).



3.5 - Architecture Model Example

The configuration model displays the mapped relationships in a tree-view format, just like the architecture model does. However, it shows an explicit indication of the architecture dependencies to identified features. It contains only those architectural elements that have explicit dependencies to a feature to be instantiated; all others are considered mandatory features and will be included for every new SPL instance.



3.6 – Configuration Model example

2.1.3 Customizable test template specification

The next step of our approach involves the specification of test templates, which is necessary to enable the implementation elements (classes, interfaces, features and files) to be customized during a new instance or product derivation. Templates shall be defined following the items identified as variabilities (from stage three), and considering these aspects during implementation. The aim is to include the maximum number of elements and to thereby minimize the work to be done by the engineers for the derivation of a new instance.

2.2 *Test script derivation*

Once all models and templates have been created and refined in a way that represents the implementation and variabilities of the software product line's architecture, the GenArch tool is ready for the derivation of new instances / products of the SPL, which in our case will represent performance test scripts.

In the next sections, we will present the final steps of our approach, which are needed in order to correctly create and derive a new instance by using the GenArch tool. We will also indicate which actions need to be performed by the engineer after the new instance has been derived.

2.2.1 Test script generation

In the fifth step of our approach, the following actions shall be performed with the intent of allowing the creation and derivation of a new instance:

(1) Initially, the engineer must create a feature model instance in order to decide which variabilities will be part of the generated test script and supply it to the derivation tool, specifying for each selected variability its attributes value (if required).

Testing & Finance 2010

The Conference for Testing & Finance Professionals

June 7th and 8th, 2010

in Bad Homburg (near Frankfurt a. M., Germany)

More information and registration at www.testingfinance.com

Keynotes by

Nitin Bhargava

Barclays Bank, UK

BJ Rollison

Microsoft Corp

Knut Jessen

DekaBank

Prof. Dr. Hermann Schulte-Mattler

Fachhochschule Dortmund

Rex Black

RBCS

Sponsors

CS Consulting AG

Exhibitors

BSM, CaseMaker, hp, iSQI, logica, macros, Microsoft, spirit se, T-Systems

Supporting Organisations

VOB SERVICE, te testing experience, Díaz Hilterscheid

Testing & Finance 2010 - The Conference for Testing & Finance Professionals

June 7th and 8th, 2010 in Bad Homburg

Day 1	Track 1	Track 2	Track 3	Track 4 - Finance
9:10	Opening Speech, José Díaz			
9:15	Key Note Nitin Bhargava - Barclays Bank, UK			
10:15	Break			
10:20	"Exploratory testing explained" Alon Linetzki	"Allheilmittel Test-Automatisierung?" Thomas Lingenfelder	"Managing a Lean test process; is testing waste?" Chris Schotanus	"Zweite MaRisk Novelle - Auswirkungen auf die Banken IT" Timo Schuhmann
11:10	Coffee Break			
11:30	"Test Governance structure" Holger Bonde & Thomas Axen	"TM2010 - Doppelpass zum Erfolg Unsere Erfahrungen mit TPI®, ISTQB®-Standards und deren Anwendung" Tom Schütz & Dr. Peter Hübner	"SCRUM & Testing; Back to the Future" Erik van Veenendaal	"Post Merger Integration einer Hypothekenbank" Volker Sobieroy & Dr. Jörn Rank & Patrick Esperstedt
12:25	Key Note "Qualitätsmanagement im Kontext der IT-Zielarchitektur bei der DekaBank" Knut Jessen - DekaBank			
13:25	Lunch			
14:40	"Random Testing in a Trading System" Noah Höjeberg	"Testautomatisierung durch MBT" Florian Prester	"Ausgelagerte Testautomatisierung in einem agilen Softwareprojekt" Michael Böll	"Aktuelle Änderungen im Aufsichtsrecht" Dr. Karl Dürselen
15:35	"Governing Testing of Systems of Systems" Bernard Homès	"It Won't Work, it Won't Work. It Worked" Jamie Dobson	"Roles and responsibilities of the tester in a Scrum team" Danny Kovatch	"Umsetzung neuer aufsichtsrechtlicher Anforderungen mit BAIS Erfahrungsbericht eines BAIS Anwenders" Cornelia Rickels-Moser
16:25	Coffee Break			
16:40	"Start and Exit Criteria for Testing" Hans Schaefer	"Knowledge is Power: Do you know where your quality is tonight?" Alexandra Imrie & Hans-Joachim Brede	"BAIS – Optimale Transparenz im Meldewesen" Christian Karner "Konzeption – Entwicklung – Testen Überlegungen zu einer zukunftsträchtigen IT-Architektur für das Meldewesen eines Kreditinstituts" Hosam Elsayed & Martin Paul	"Erfahrungsbericht Testautomation im Landesbanken-Rating" Volker Töwe
17:35	Key Note "How we test at Microsoft" BJ Rollison - Microsoft Corp.			
18:40	Social Event Dinner + Theatre and Chill out			

Day 2	Track 1	Track 2	Track 3	Track 4 - Finance
9:10	Key Note "Das Basel II Puzzle: angemessene Eigenkapitalausstattung auf dem Prüfstand", Prof. Dr. Hermann Schulte-Mattler - Fachhochschule Dortmund			
10:15	"Test Process Improvement: Critical Success Factors" Graham Bath	"Flexible testing environments in the cloud" Jonas Hermansson	"Agile approach for testing in a high risk legacy environment" Jan Rodenburg	"Verbriefung und Refinanzierung – Plattformstrategie in der Erstellung und Nutzung von Standardsoftware" Klaus Leitner
11:05	Coffee Break			
11:25	"TMMi – how are we doing as an industry?" Geoff Thompson	"Risk based test planning" Dr. Hartwig Schwier	"IT-Governance, notwendig oder überdimensioniert? Wie ich mich der Herausforderung einer effizienten IT stellen kann." Arjan Brands & Oliver Rupnow "Requirements Management: Effizient, integriert, risikobasiert" Thomas Köppner	"Neue Europäische Bankenaufsicht - Auswirkungen auf das deutsche Meldewesen " Nicolas Jost
12:20	Key Note "Satisfying Test Stakeholders ", Rex Black			
13:20	Lunch			
14:30	"Improving quality and saving costs using static analysis: a case study" Dr. Mike Bartley	"Testprozess bei der Volkswagen Financial Service AG - Keine Theorie sondern gelebte Praxis" Ingolf Gäbel & Yvonne Sternberg	"Qualität von Anfang an – Anforderungsbasiertes Testen mit Microsoft Visual Studio Test Professional" Matthias Zieger "Requirements Engineering and Certification" Benjamin Timmermanns	"Ertragsorientiertes Liquiditätsrisikomanagement - Erhöhung der Erträge durch bessere Liquiditätsrisikoanalyse in Zeiten rückläufiger Ergebnisse" Prof. Dr. Stefan Zeranski
15:20	Coffee Break			
15:25	"TMMi - Model Development and practical expierences" Matthias Rasking	"SOA testen - der inkrementelle fachliche Integrationstest" Ronald Grindle	"Agilität im Wasserfall - Scrum: nur Ganz oder Gar nicht?" Gernot Glawe	"Pros und Cons zur Einführung einer Leverage-Ratio als Ergänzung der risikosensitiven Basel II-Normen" Dr. Uwe Gaumert
16:20	Closing Session, José Díaz			

(2) After completing the selection of feature model variabilities, the engineer requests the derivation of this new instance. The GenArch derivation tool will check the entire architecture model, individually processing its elements and verifying in the configuration model if the element is dependent on one or more features. It will therefore create new instances only for the required elements, in accordance with what was requested in the feature model of the new instance.

(3) Finally, the tool produces an Eclipse framework project as a result of the derivation process. This contains only those implementation elements that match the product first designed in the initial feature model configuration and specified by the application engineer.

2.2.2 Generated script final customization

Due to our project's characteristics and constraints, which we will describe in the last section of this article, we decided to organize the project using the GenArch derivation tool, in order to ensure that the generated Eclipse project could contain the test script specified according to the variabilities and its values set in step (I). However, we still need to go through the sixth and last step of our approach, where the generated test scripts will receive final customizations and adjustments in order to enable correct execution using the performance test tool JMeter. In this section, we will describe the necessary steps to ensure a perfect coupling and execution of the generated test scripts.

(I) Initially, the .jmx (JMeter test scripts standard extension) file available inside the Eclipse project, which is generated by GenArch during new instance derivation, must be loaded into JMeter.

(II) Next, JMeter is used to capture all information provided by the browser, with the intent of capturing transactions (operations) that could not be represented in GenArch models, but which are still part of the test script.

(III) The captured JMeter data must be inserted into the correct execution position within the generated test script. All extra information needed, like customized variables, need to be manually configured in the added transactions, in order to ensure that the items previously configured using GenArch can be correctly accessed;

(IV) Next, test script needs to be run for the first time, in order to ensure that it works as designed. If it doesn't, we need to make the necessary adjustments and corrections;

(V) At last, we have the final test script, which consists of all the information generated by GenArch and JMeter's recorded transactions data which have been added. This will finally be available for test execution as planned by the test team.

3. Deriving a performance test script example

Figure 4 below shows a feature model created for deriving performance test scripts as described in the previous sections. The picture shows how the GenArch derivation tool shows the selection of a new test instance, in which we can define the features we want to include for the new test script together with its respective values.

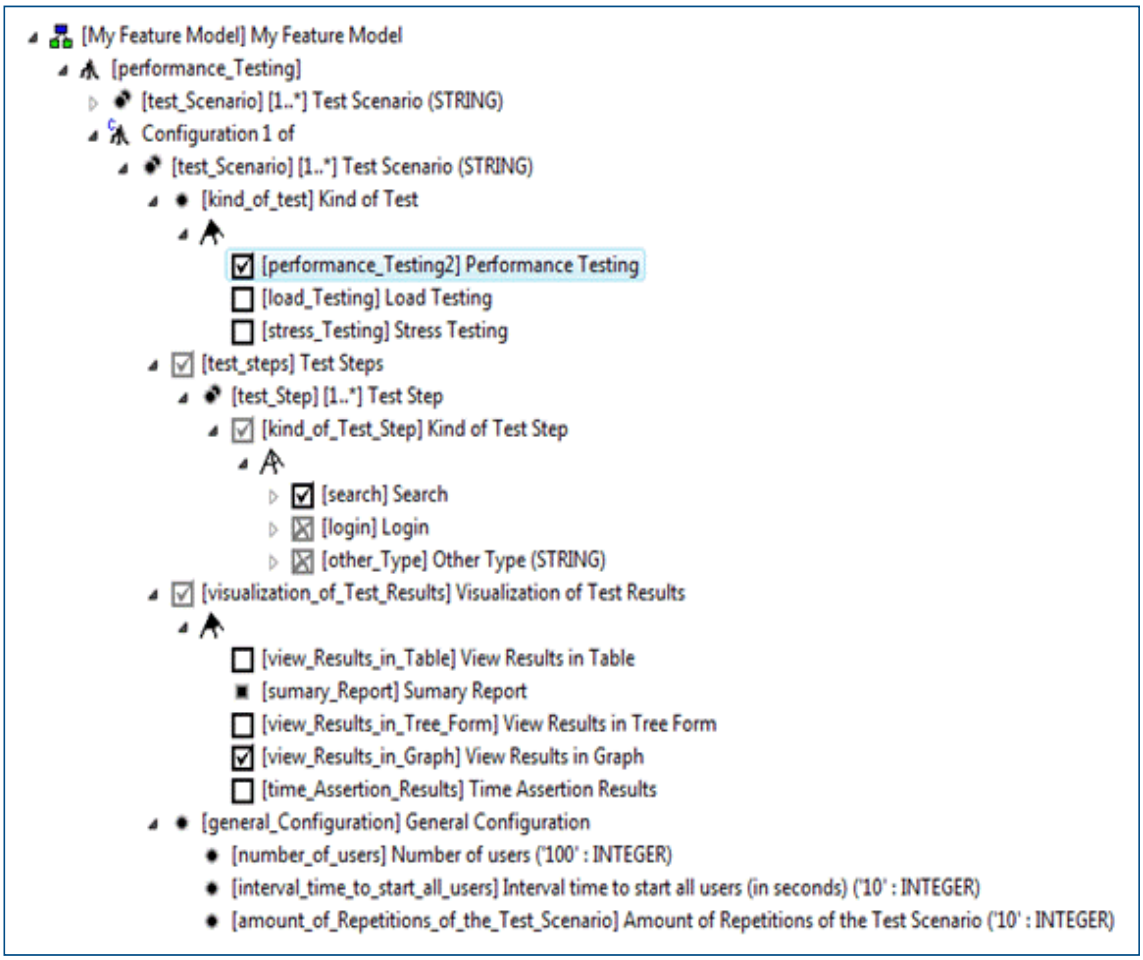


Figure 4 – Configuration of a new test instance

For example, the value of the Number of Users element has been set to 100 (one hundred) and can easily be modified by double clicking on it. We can also see that the selections of elements like *Search* and *View Results in Graph* and several features have automatically been eliminated due to restrictions previously defined. The selection determines the template customization with JMeter elements (code), which are related to the features selected.

In our example the selection shown has produced the test script displayed in the picture below, which was produced with the intent of validating the response time of a search operation for a web search application.

As we pointed out before, we had to perform some adjustments as a final step in order to include all desired features in the test script and to ensure its proper execution. To make the generated test script complete and ready for execution, missing transactions and operations were added producing the ini-

Figure 5 – Derived Test Script

tially desired test script shown in the picture below.

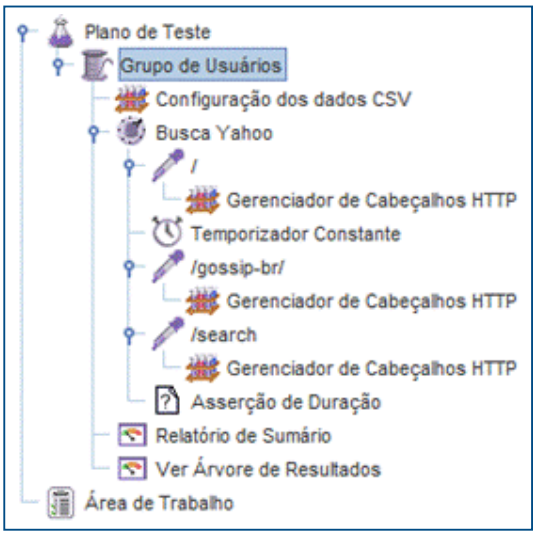


Figure 6 – Final Test Script

4. Final Considerations

During the development of our work, several constraints could be observed that could affect the full utilization of our approach. We could also point out several enhancements and studying opportunities for developing the described approach further and make it a real alternative to all existing performance testing techniques.

These are some of the constraints and opportunities identified:

- **Limited feature model representation possibilities**. During domain analysis performed to support the creation of derivation models, some details needed to be left out in order to facilitate an easier and faster understanding of the tool to create new performance test script instances. On the other hand, some features like simulating user abandonment for load tests, multiple threads or multiple virtual users groups, among others, could not be modelled in a way that would produce a correct derivation, restricting its use to the manual customization task performed in the last step of our approach. Future enhancements on our work may successfully model these features.
- **Reuse between applications and test type**. During the development of our research, we detected that the reuse of test scripts may occur among different types of performance tests and also among different applications. Regarding performance test types, our work concluded that different performance, load or stress test scripts vary mostly with regard to configuration parameters, but generally present a similar structure. Besides that, we observed that the test scripts of different applications share several common elements, allowing a greater reutilization of element definitions.
- **Approach usage with other non-functional test types**. We also identified that the defined approach could also be used for deriving test scripts for other non-functional requirements, like security, reliability, etc. With regard to security, for example, it is possible to apply our approach, even including Jmeter, to simulate scenarios that involve resources exhaustion to identify possible security failures related to this condition.
- **Eliminating customizations and adaptations through approach restructuring**. A research extension with the purpose of identifying alternatives is to look into possibilities to make it possible to eliminate the task of adapting the generated script for correct execution. This would contribute for the removal of the recording operation for transactions that are not included in the derivation tool, which would make the script generation a lot easier.
- **Automatic model derivation through test scripts**. This is about defining a way of adapting the existing annotations feature, already used by GenArch, to enable an automatic model derivation from a group of test scripts. This could reduce the amount of work that currently needs to be done in script analysis and model creation, which in our research were done entirely manually.

In our work we defined an approach for the derivation of performance test scripts for web applications, which is composed of six steps. This has the main goal of deriving new test script instances, but also has the objective to allow an easy way of adapting these test scripts to other test types and scenarios.

In this article, the approach developed during a software engineering master course accomplished at C.E.S.A.R.(Recife Center for Advanced Studies and Systems) institute was described in some detail. For further information regarding the results achieved and/or any other questions, please contact one of the authors.

Biography

José Carréra, MSc, has been test engineer at C.E.S.A.R. (Recife Center for Advanced Studies and Systems) since 2006 and Professor of Computer Science at the Faculdade de Tecnologia de Pernambuco, Brazil, since 2010. He holds a master degree in software engineering, graduated in computer science, and he is a Certified Tester — Foundation Level (CTFL), certified through the ISTQB (International Software Testing Qualifications Board). His research interests include performance testing, exploratory testing, agile methodologies and software quality in general.

Uirá Kulesza is an Associate Professor at the Computer Science Department (DIMAp), Federal University of Rio Grande do Norte (UFRN), Brazil. He obtained his PhD in Computer Science at PUC-Rio - Brazil (2007), in cooperation with University of Waterloo and Lancaster University. His main research interests include: aspect-oriented development, software product lines, and design/implementation of model-driven generative tools. He has co-authored over 70 referred papers in international conferences, journals and books. He worked as a research member of the AMPLE project (2007-2009) - Aspect-Oriented Model-Driven Product Line Engineering (www.ample-project.net).

HANDBUCH

TESTEN
IN DER FINANZWELT

Das Qualitätsmanagement und die Software-Qualitätssicherung nehmen in Projekten der Finanzwelt einen sehr hohen Stellenwert ein, insbesondere vor dem Hintergrund der Komplexität der Produkte und Märkte, der regulatorischen Anforderungen, sowie daraus resultierender anspruchsvoller, vernetzter Prozesse und Systeme. Das vorliegende QS-Handbuch zum Testen in der Finanzwelt soll

- *Testmanagern, Testanalysten und Testern sowie*
- *Projektmanagern, Qualitätsmanagern und IT-Managern*

einen grundlegenden Einblick in die Software-Qualitätssicherung (Methoden & Verfahren) sowie entsprechende Literaturverweise bieten aber auch eine „Anleithilfe" für die konkrete Umsetzung in der Finanzwelt sein. Dabei ist es unabhängig davon, ob der Leser aus dem Fachbereich oder aus der IT-Abteilung stammt. Dies geschieht vor allem mit Praxisbezug in den Ausführungen, der auf jahrelangen Erfahrungen des Autorenteams in der Finanzbranche beruht. Mit dem QSHandbuch sollen insbesondere folgende Ziele erreicht werden:

1. *Sensibilisierung für den ganzheitlichen Software- Qualitätssicherungsansatz*
2. *Vermittlung der Grundlagen und Methoden des Testens sowie deren Quellen unter Würdigung der besonderen Anforderungen in Kreditinstituten im Rahmen des Selbststudiums*
3. *Bereitstellung von Vorbereitungsinformationen für das Training „Testing for Finance!"*
4. *Angebot der Wissensvertiefung anhand von Fallstudien*
5. *Einblick in spezielle Testverfahren und benachbarte Themen des Qualitätsmanagements*

Herausgegeben von Norbert Bochynek und José M. Díaz Delgado

Die Autoren
Björn Lemke, Heiko Köppen, Jenny Siotka, Jobst Regul, Lisa Crispin, Lucia Garrido, Manu Cohen-Yashar, Mieke Gevers, Oliver Rupnow, Vipul Kocher

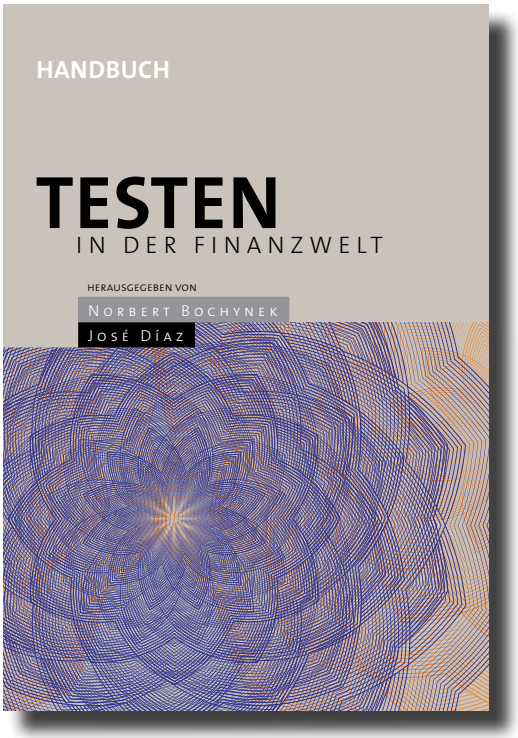
Gebundene Ausgabe: 431 Seiten

ISBN 978-3-00-028082-5

1. Auflage 2010 (Größe: 24 x 16,5 x 2,3 cm)

48,00 € (inkl. Mwst.)

www.diazhilterscheid.de



Daddy's performance test

Column

by Thomas Hijl

Saturday 05:30 - Kick-off

Kick-off at 05:30 AM. At 05:30 I was informed that a performance test was due to take place. Still in my bed, I had no clue what was going to happen and was not prepared at all. Of course, like with any test project, I had taken all measures to prepare myself. Still, I did not realize what was coming and was leaning back, awaiting my test assignment.

06:00 AM - The assignment.

It was not a structured A4 paper with a signature, as any project management method would teach you, but it was an assignment, like in many test projects, defining itself along the way. So, as an experienced acceptance and test manager I conducted a quick scan to get a better picture:

- 05:30 first signs of a test assignment
- 06:00 the project has started to take off
- 06:30 first results delivered to conduct a plan

The first test results:

I did a test of the response times while building up the pressure.

- 6-8 minutes between each response
- Duration of the pressure load 60-90 minutes

Mmm, where does this lead to? I was instructed that I had to wait until the response times would be between 3-4 minutes, before calling in more resources. Until then, there would be no budget available. So, me and my partner kept testing and measuring ourselves, without any external help.

07:00 AM - System overload.

Big problems surrounding the test object. Environmental issues came to the surface, and I was running left and right to keep the project going. Pffff, heavy times for a manager. But, ... "This is your job, mate!" Keeping yourself together, supporting your colleagues to keep up the good work, gathering the last details before the final test will take place and preparing to rush off to the emergency lab, when it all gets critical.

08:00 AM – GO / NO GO.

I managed to take control of the situation again and continued measuring response times and pressure.

- 3-4 minutes between each response
- Duration of the pressure load 60-90 minutes

Damn, I better call for backup. "Hello, I think I have a critical situation here on my hands and I urgently need your expertise." On the other side of the line: "Stay calm, grab everything together and come to the emergency lab, where we will help you to execute the final performance tests in our professional environment." I hung up and prepared everything for the move to the back-up environment. However, I saw that the test object was about to collapse. With our last energy we managed to get everything on transport. What a trip!

09:00 AM - Arrival at the lab.

When we arrived at the lab, it all went very quick. I had all the resources I needed to finalize the testing. Pressure and stress went up to their maximum, but it all kept performing well. What a joy being part of this team!

09:52 AM - The delivery.

The stress test took only 20 minutes and before I could say "haute cuisine", the project deliverable was there. Brown hair, blue eyes, breathing, moving,... all fully functional. Man,.... It was a stressfull project again, like most, but ,... what a great performance. I thank all the testers, builders, management and back-up troops for fantastic teamwork. Julian is healthy, his mum is beautiful and I am exhausted ;-)

Testing might never be very structured, but with capable "hands-on" management it will always be a success.

Thomas Hijl is acceptance manager and partner at Qwince BV. He started in 1996 as consultant / project manager for Philips Semiconductors (a.k.a. NXP) working on technology projects in the EMEA, APAC and AMEC regions. He switched to test management because of his ambition to improve project deliverables. Now, he serves clients with managing acceptance throughout the project lifecycle.

Playtesting: part of game design and development

by Andrea de Vries & Ewald Roodenrijs

Have you ever discovered a bug in a game? Or have you noticed that the behavior did not match the explanation? And what about the numerous cheat sheets that you can find all over the Internet - are those bugs? These are all good reasons to start testing these games. Video games of all shapes and sizes are becoming increasingly popular. Last year the game industry has grown to a 17.5 billion Euro industry. It has even grown during last year's Financial Crisis, and it will continue to grow in the coming years. This industry consists of many different types of games, like (online) console or PC games, mobile games, casual games and serious games.

While gaming, have you ever discovered a bug in a game? Or noticed that the behavior does not meet the expectation? Have you ever looked for game cheats in the Internet? These examples may come from games that have not been tested well enough. How do we test games in order to get the most information about their quality? Playtesting is a testing technique to generate the necessary feedback and insight in the game experience and game flow from various possible players (testers) of a game. With playtesting it is possible to collect valuable feedback so that the experience of the game is exactly how the designer envisioned it.

Playtesting does not involve (professional) testers that check the game rigorously and test each element of the software for defects. Playtesting involves testing the games functionally, otherwise known as quality assurance testing. Examples: keeping the game 'on' all night and check if it still works the next morning, take out the controller during gameplay or constantly change your weapon during the game for a long time. Because playtesting is not a functional testing session for the game, quality assurance should not be done in a playtesting session.

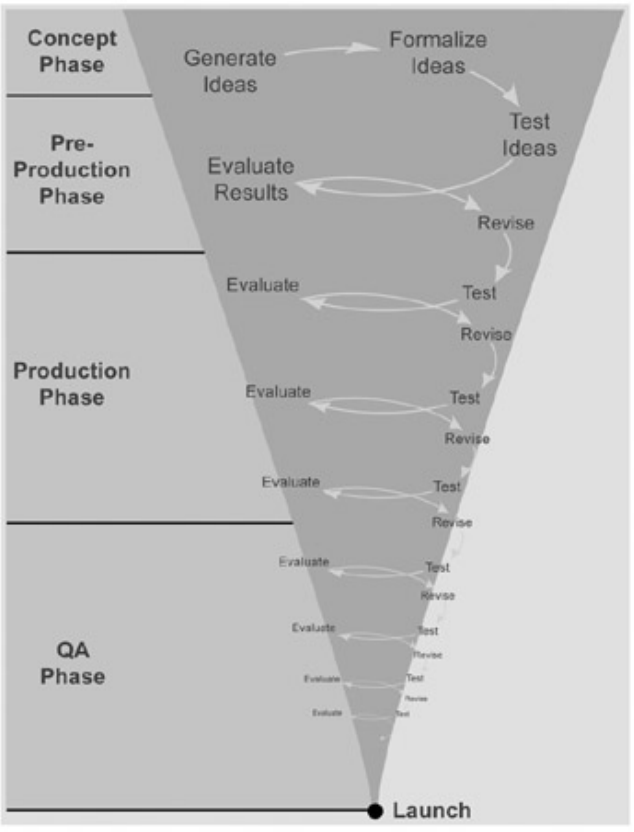
A playtest is also not the same as a security test. A security test should be performed by a security specialist, and in a lot of games security needs to be checked as soon as possible (I highly recommend reading the book "Exploiting online games" on this subject). With Massively Multiplayer Online Role Playing Games (MMORPGs), for instance, it is of great importance that the game is secure. These kinds of tests are, like quality assurance testing, not playtests.

About playtesting

Playtesting is an important activity performed in the game design and development process. The goal is to gain insight in how people (players) experience the game. With playtesting the flow of the game is also tested: this is the balance between the challenge and the ability of the player. A game should be complete, balanced, and fun to play. This is where playtesting comes in. Developing a game should be targeted on its audience. This can be done by keeping a close relationship with a player, his/her needs and his/her perspective on the game. Playtesting generates a continuous stream of data that are used to evaluate the game. As a result, the game can be revised early in the development process.

We often say that software testing needs to start as soon as possible in the software development process. This is also true for games. Game developers need to engage in a technique to test their product as early as possible. Playtesting is not a small feature in the design process, and it is not only done at the end of development. It should already be done in the design phase. At the end of the project, it is only possible to change a number of top-level features, because by then the core gameplay is already fixed.

Playtesting can be done by all people in the development process; designers, developers, management, marketing and/or testers,



i.e.people that know the product. However, you can also use people unfamiliar with the game, or with gaming in general. This is important because they have no relation with your product and therefore have no benefit from giving substantive criticism. The most ideal playtester is one who fits the target audience.

During the playtesting process, you use testing cycles in the different phases of development. The figure below shows how these testing cycles get tighter and tighter as the development process moves forward, signifying smaller and smaller issues to solve and changes to make [Fullerton, 2004]. This way you merely perfect the game as the development life cycle draws to an end, you are not changing it fundamentally.

Methods of playtesting

There are different methods for playtesting:

- *One-On-One Testing*: Have individual testers play the game and observe them. During the tests you take notes and possibly ask them questions.
- *Group Testing*: Have a group of people play the game together. During the test you observe the group and ask questions as they play, if needed.

There are also different ways to evaluate a playtesting session. Of course, these methods can be combined to fit the tests of the game:

- *Feedback forms*: Each person who tests gets a list with questions to answer after playing the game. The game designer then compares the results.
- *Interviews*: An in-depth interview of a tester after a playtesting session.
- *Open Discussion*: A one-on-one discussion or a group discussion after a round of playtesting. This can be done with a freeform discussion or a more structured approach to guide the conversation and introduce specific questions.

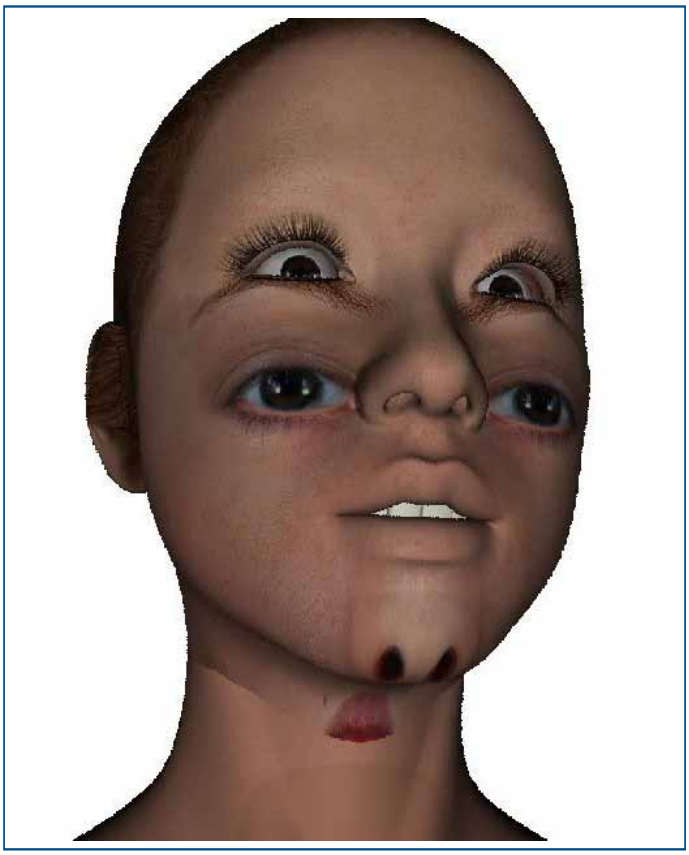
Like with many types of testing, it is important to record as much as you can. This recording can be done by recording the sessions with cameras. Webcams or cameras can show how players behave while they play the game. Another recording technique is using data hooks to log the actions players take while gaming. Analyzing this feedback can give valuable information about gameplay, the experience, the fun factor and the difficulty of the game.

Upgrade your playtesting

How do you perform a playtest when the game is not yet finished? When this happens you need to help the testers by giving them instructions about what they can or cannot do. For example, tell them to only pay attention to the gameplay and not the graphics because they are not finished yet. This is done with this example: an unfinished design version of Paladin Studios' Enercities.

The following image is another example where the game is not finished or has technical errors. Since playtesting is about the game experience and flow, errors like these should not affect or hinder a playtest.

How do you organize playtesting to achieve results? It is important to have a good understanding of the game. It is important to know certain aspects of the game you are playtesting, like the target audience, their feelings and the experience a player should have. When you know these 'needs' of the game, it is easier to filter the feedback that is given. You can sort out the feedback which does not affect those aspects.



The participation of the project team members (like the designers) in playtesting sessions can be an important factor for a successful program. Team members can spot problems more accurately than others; they know the game they have been creating. This method of playtesting can provide valuable information that was not found earlier.

The most objective information is received by gathering a big crowd for testing. Find as many people as possible. It is important to target large audiences, because you do not exactly know people's habits or backgrounds. A large audience can also help with future tests.

The importance of playtesting

Playtesting is important. Through playtesting you can determine whether the flow of the game is good and whether the players get the right experience. Playtesting is a critical part of game design and development that cannot be rushed through or sidelined. Obtaining feedback and analyzing it allows designers and developers to see the game mechanics for what they actually are and not what they should be. The use of independent playtesters is necessary to gather new and useful feedback.

The best advice is to start playtesting from the very moment the design phase of the product starts. If you wait until a game is finished, it is too late to make any changes to the fundamentals of

the game. If the core gameplay is no fun or not interesting, you have a faulty product. Based on the early use of playtesting, it is possible to react on the feedback from the testers and possibly change the game for the better.

Of course, it is not always pleasant to have someone point out the defects of a game. However, seeing or hearing players' experience, even if they are negative, is valuable feedback. Playtesting improves the flow of the game and therefore the game becomes better.

There are numerous ways to playtest. One thing that all playtesting methods have in common is the goal: to gain useful feedback from players in order to improve the product.

[Fullerton, 2004]

Fullerton, T., Swain C., and Hoffman, S., Game Design Workshop: Designing, Prototyping, and Playtesting Games (2004), CMP Books, ISBN: 9781578202225

[TMap NEXT, 2006]

Koomen, T., Aalst, L. van der, Broekman, B., Vroon, M., TMap® Next (2006), UTN, ISBN: 90-72194-80-2

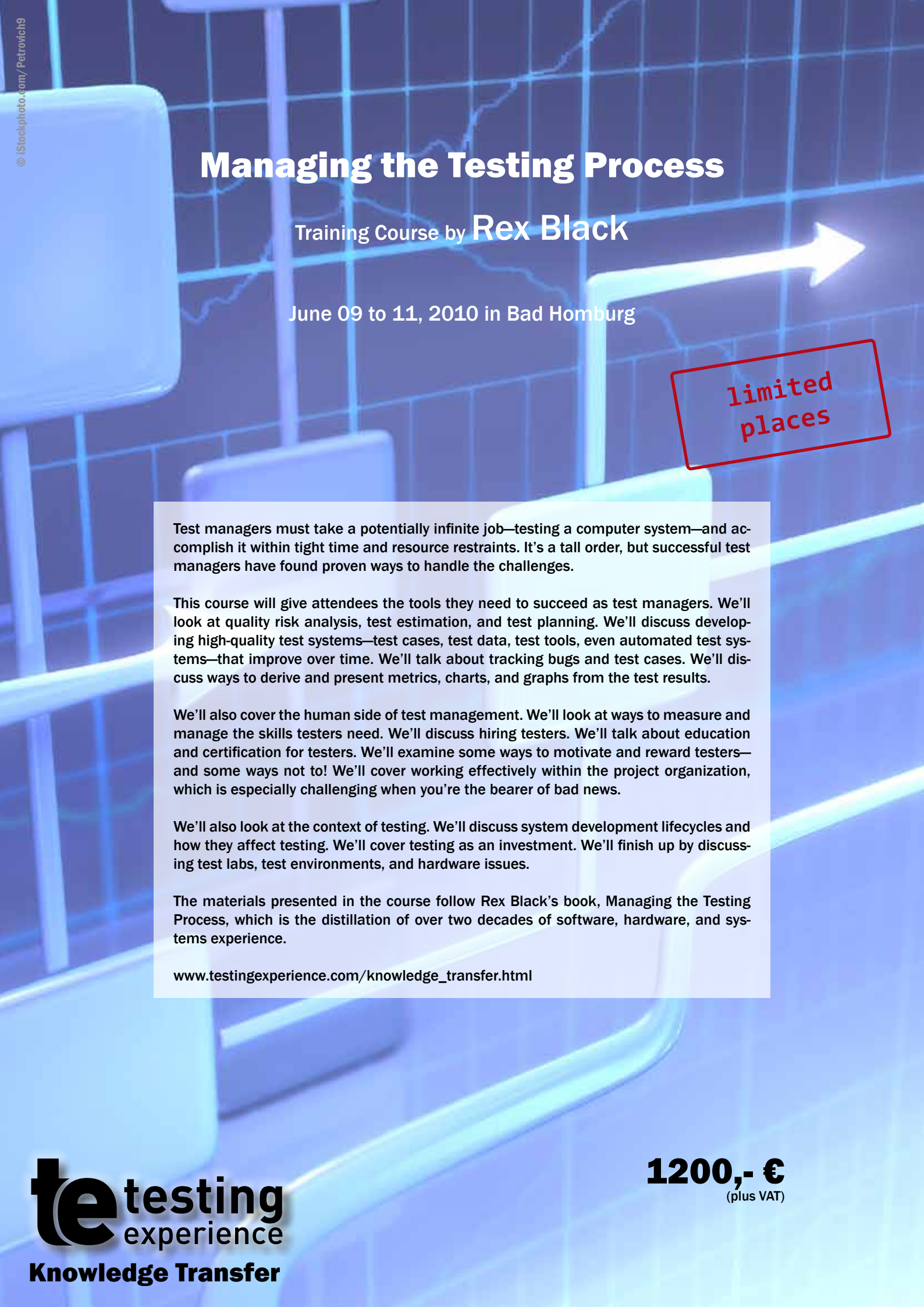
[De Vries, 2009]

De Vries, A., Rules of play Design Essay (2009), HKU GGD2

Biography

Andrea de Vries is currently a 2nd year student of Game Design and Development at the Utrecht School of the Arts. GDD teaches students to design and develop games. They are trained to work in a team and take on different roles like gamedesigner, programmer, modeller and animator. As game designers, students are also taught to critically think about games and the role of games and designers within the industry.

Ewald Roodenrijs is a senior test manager and a member of the business development team within Sogeti Netherlands. As a member of the development team, he works on different test innovations like model-based testing, clouds, crowdtesting, testing augmented reality and using new media in testing. He's also co-author of the book 'TMap NEXT® – BDTM', a speaker at (international) conferences, an author of various national and international articles in expert magazines, and has created various training courses for test management purposes.



Managing the Testing Process

Training Course by Rex Black

June 09 to 11, 2010 in Bad Homburg

limited places

Test managers must take a potentially infinite job—testing a computer system—and accomplish it within tight time and resource restraints. It's a tall order, but successful test managers have found proven ways to handle the challenges.

This course will give attendees the tools they need to succeed as test managers. We'll look at quality risk analysis, test estimation, and test planning. We'll discuss developing high-quality test systems—test cases, test data, test tools, even automated test systems—that improve over time. We'll talk about tracking bugs and test cases. We'll discuss ways to derive and present metrics, charts, and graphs from the test results.

We'll also cover the human side of test management. We'll look at ways to measure and manage the skills testers need. We'll discuss hiring testers. We'll talk about education and certification for testers. We'll examine some ways to motivate and reward testers—and some ways not to! We'll cover working effectively within the project organization, which is especially challenging when you're the bearer of bad news.

We'll also look at the context of testing. We'll discuss system development lifecycles and how they affect testing. We'll cover testing as an investment. We'll finish up by discussing test labs, test environments, and hardware issues.

The materials presented in the course follow Rex Black's book, Managing the Testing Process, which is the distillation of over two decades of software, hardware, and systems experience.

www.testingexperience.com/knowledge_transfer.html

1200,- €
(plus VAT)

testing experience
Knowledge Transfer

Agile Performance Testing

by Sowmya Karunakaran

To accomplish performance testing, we need to engage in a careful, controlled process of measurement and analysis. It is critical to understand that the aim of performance testing is not to find bugs, but rather to identify bottlenecks. Ideally, the software under test should be stable enough so that this process can proceed smoothly.

Most large corporations have performance testing and engineering groups today, and performance testing is a mandatory step to get the system into production. However, in most cases, it is pre-production performance validation only. Hence there seems to be a need for an efficient way to do performance testing. Most often it is believed that agile introduces chaos and is therefore not suitable for performance testing, which is a careful, controlled process. This article unfolds the fact that doing **performance testing in a more agile way** may increase its efficiency significantly.

The teams practicing XP always had working software due to the continuous integration practices. They also have a suite of automated tests. XP coupled with agile planning and automated performance regression tests is a big step towards agile performance testing.

Let's look at how some of the agile principles like test early, continuous improvement and collaboration can be applied to performance testing.

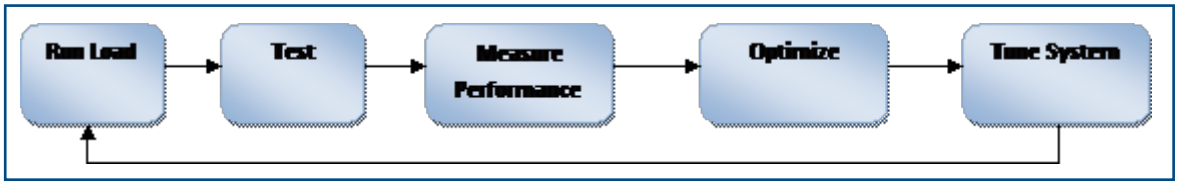
Test Early – Early Performance Testing

The main obstacle here is that many systems are generally colossal, if there are parts - they don't make much sense separately. However, there may be significant advantages to test-drive development. If you can decompose the system into components in such way that we can actually test them separately for performance, then all that we need to do is to fix integration issues, if any, when we put the system together. Another problem is that large corporations use a lot of third-party products, where the system usually seems to be a „black box" and cannot be easily understood, which makes it more difficult to test effectively.

Though it is well understood that it is better to build performance right up front, i.e. during the design stage, and pursue performance-related activities throughout the whole software lifecycle, unsurprisingly quite often performance testing happens just before going live within a very short timeframe allocated for it. Still approaching performance testing formally, with a rigid, step-by-step approach and narrow specialization, often leads to missing performance problems altogether, or to a prolonged agony of performance troubleshooting. With little extra effort, by making the process more agile, efficiency of performance testing increases significantly – and these extra efforts usually pay off multifold even before the end of performance testing.

Iterative Performance testing

This is crucial for success in performance testing, and an agile environment is surely conducive for this. For example: A tester runs a test and gets a lot of information about the system. To be efficient he needs to analyze the feedback that he got from the system, make modifications to the system and adjust the plans if necessary. The cycle below, if done in an iterative fashion until the system under test achieves the expected levels of performance, will yield better results and also serve as a baseline to perform regression tests.



Performance Optimization (Continuous Improvement)

Performance testing becomes meaningless if it is not complemented with performance optimization. Usually when the performance of a system needs to be improved, there are one or two methods where all the time is spent. Agile optimization works by garnishing existing unit tests with a timing element.

The process involves six easy steps:

1. Profile the application
2. Identify the bottlenecks
3. Find out the average runtime of the methods and create a baseline.
4. Embellish the test with the desired runtime.
5. Refactor / optimize the method.
6. Repeat until finished.

However, care must be taken to ensure that the data used for tes-

ting is good. Tests often don't use user data. Unit tests are usually happy path tests. That means the data they use may not be representative of the real user data. Therefore, optimizations against poor data could be false optimizations. For agile performance optimization to work, it is crucial to have access to user data.

Collaborative Testing

Performance testing is not just the role of the performance testers. XP coding techniques can be expanded to carry out continuous performance testing and fix performance bugs in a test-first manner. Developer techniques supplement deployed techniques and are necessary for a coherent performance testing strategy.

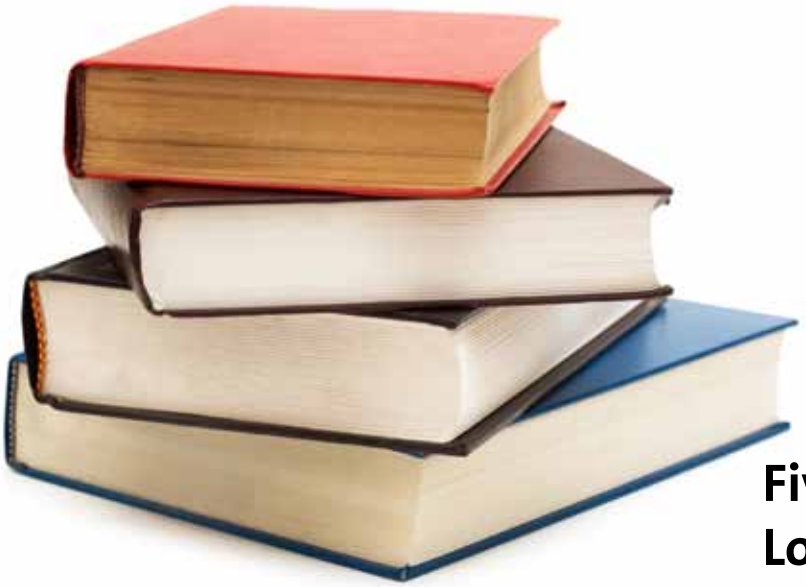
The synergy has to be created between the performance testing team and the development team to help address performance issues early and often. This is very much in line with the way agile teams operate. From a developer perspective, the two best techniques would probably be the deployed smoke tests and local load tests. These two techniques provide immediate feedback about the system. From the performance tester's point of view, the iterative testing and smoke tests will mean a lot. Of course, having sophisticated testing tools will be a boon.

Biography

Sowmya Karunakaran has been engaged in agile software development since 2005. She has worked in different flavors of agile, such as XP, Scrum, FDD and XP/Scrum hybrid. Currently she is working as an Agile Expert at the Agile Center of Excellence at HCL Technologies. She has presented many papers at various IEEE conferences and has co-authored the book "Model driven software development and integrated quality assurance", published by the IDEA group. Her interests involve model-driven architecture, agile methodologies and Human Machine Interface computing.

Next issue September 2010

Topic Metrics



Five Lessons in Performance, Load, and Reliability Testing

by Rex Black

Do you have system quality risks (or actual problems) related to performance, load, or reliability? Are you looking for some ideas on how to do performance, load, or reliability testing properly? If so, this article is for you.

In this article, I'll illustrate five lessons I've learned in performance, load, and reliability testing. I'll use case studies to show both success when the lessons were heeded—and failure when they were not. I'll provide tips for success and give warnings for common mistakes. All of the figures and case studies are real (with a few identifying details omitted), thanks to the kind permission of my clients.

Motivation Behind Performance, Load, and Reliability Testing

For the last 20 years I've worked in software testing, and many of the projects I've done for clients included performance, load, or reliability tests. Since I founded RBCS in 1994, performance, load, and reliability testing have made up a substantial part of our services mix. Such tests tend to be complex and expensive compared to, say, functionality testing, and even compared to compatibility testing or usability testing.

So, if it's complicated and expensive, why do our clients hire us for performance, load, and reliability testing? Because performance, load, and reliability bugs really hurt. Inadequate performance can lead to user dissatisfaction, user errors, and lost business, especially in Internet-enabled, business-to-business or business-to-consumer applications. Further, system failure under load often means a total loss of system functionality, right when the need is the highest—after all, the load usually comes from users trying to use your system. Finally, reliability problems can damage your company's reputation and business prospects, especially in situations where money or finances are involved.

Instead of asking "Why bother?", the better question to ask is: "Are you sure that there are not critical risks to your business that arise from potential performance, load, or reliability problems?" For many applications these days, there are such risks. Let me give you a few examples to help you see these risks, and to explain them to others.

If you are running an e-commerce business, suppose that 10% of your customers were to abandon their shopping cart due to sluggish performance. Do customers that have had a bad experience with a website, and who are famously one click away from the competition, bother to come back? How long can your company stay in business, if it is hemorrhaging customers in this fashion?

I was at a conference once, talking to a colleague in the software testing business. He told me that he had just finished a project for a banking client. During the course of that project, the bank estimated its average cost of field failure at over €100,000. Yes, I did say average; some field failures were much more expensive. Many of these field failures were related to performance, load, or reliability bugs.

If you can remember the dawn of the Internet e-commerce age, you might remember the Victoria's Secret 1999 online holiday lingerie show, timed propitiously right at the Christmas shopping season. This was a good concept, but poorly executed. The problem was that the servers couldn't handle the load. Worse yet, they degraded gracelessly: As the load exceeded limits, new viewers were still allowed to connect, slowing everyone's video feeds to a standstill. This debacle could have been avoided through sufficient performance, load, and reliability testing. Amazingly, more than ten years later, we still see these kinds of failures in web-based businesses.

The Five Hard-won Lessons

Like any complex and expensive endeavor, there are a lot of ways to screw up a performance, load, or reliability test. Now, I certainly can't tell you everything you need to know to avoid these mistakes—no one can—but I can give you some ideas based on my experience. In this article, I will stress five key lessons that should help you avoid common goofs:

1. Configure performance, load, and reliability test environments to resemble production as closely as possible, and know where test and production environment differ.
2. Generate loads and transactions that mimic real-world scenarios.
3. Test the tests with models and simulations, and vice versa.
4. Invest in the right tools but don't waste money.
5. Start modeling, simulation, and testing during design, and continue throughout the lifecycle.

These are hard-won lessons. I've learned these lessons through personal experience with successful projects, as well as by witnessing some expensive project failures when clients made these mistakes. Let's look at each lesson in detail.

Lesson 1: Realistic Environments

People commonly bring invalid physical-world metaphors to system engineering and system testing. But software is not physical,

and software engineering is not yet like other engineering fields.

For example, our colleagues in civil engineering build bridges out of standard components and materials like rivets and concrete and steel, but software is not assembled out of standard components to any great extent yet. Much is still designed and built to purpose, often from many purpose-built components.

Our colleagues in aeronautical engineering can build and test scale models of airplanes in wind tunnels. However, they have Bernoulli's Law and other physics formulas that they can use to extrapolate from scale models to the real world. Software isn't physical, so physics doesn't apply.

There are a select few software components for which some standard rules apply. For example, binary trees and normalized relational databases have well-understood performance characteristics. There's also a fairly reliable rule that says once a system resource hits about 80% utilization, it will start to saturate and cause non-linear degradation of performance as load increases. However, for the most part, it is difficult and unreliable to extrapolate results from smaller environments under scaled-down loads into larger environments.

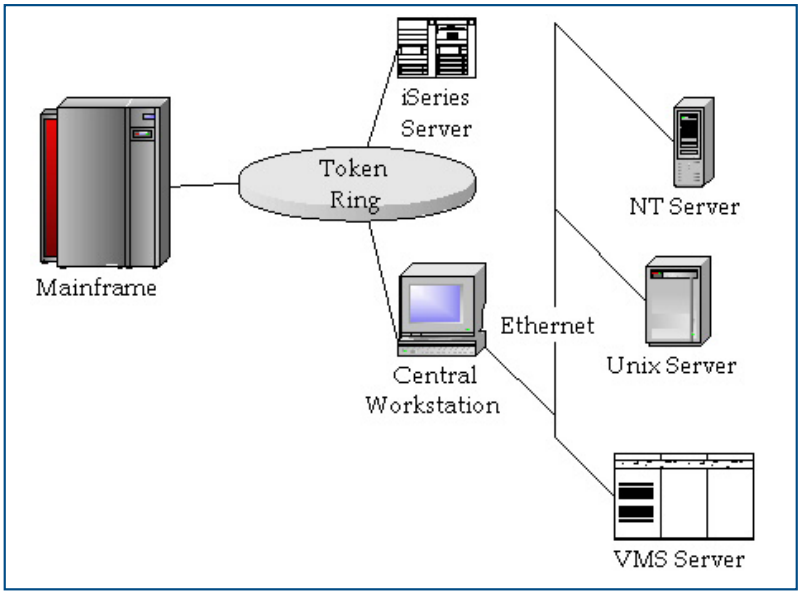


Figure 1: An unrealistically scaled down test environment

Here's a case study of a failed attempt to extrapolate from small test environments. One of our clients built a security management application that supported large, diverse networks. The application gathered, integrated, and controlled complex data related to the accounts and other security settings on servers in a managed network.

Before they became our client, they ran performance tests in a

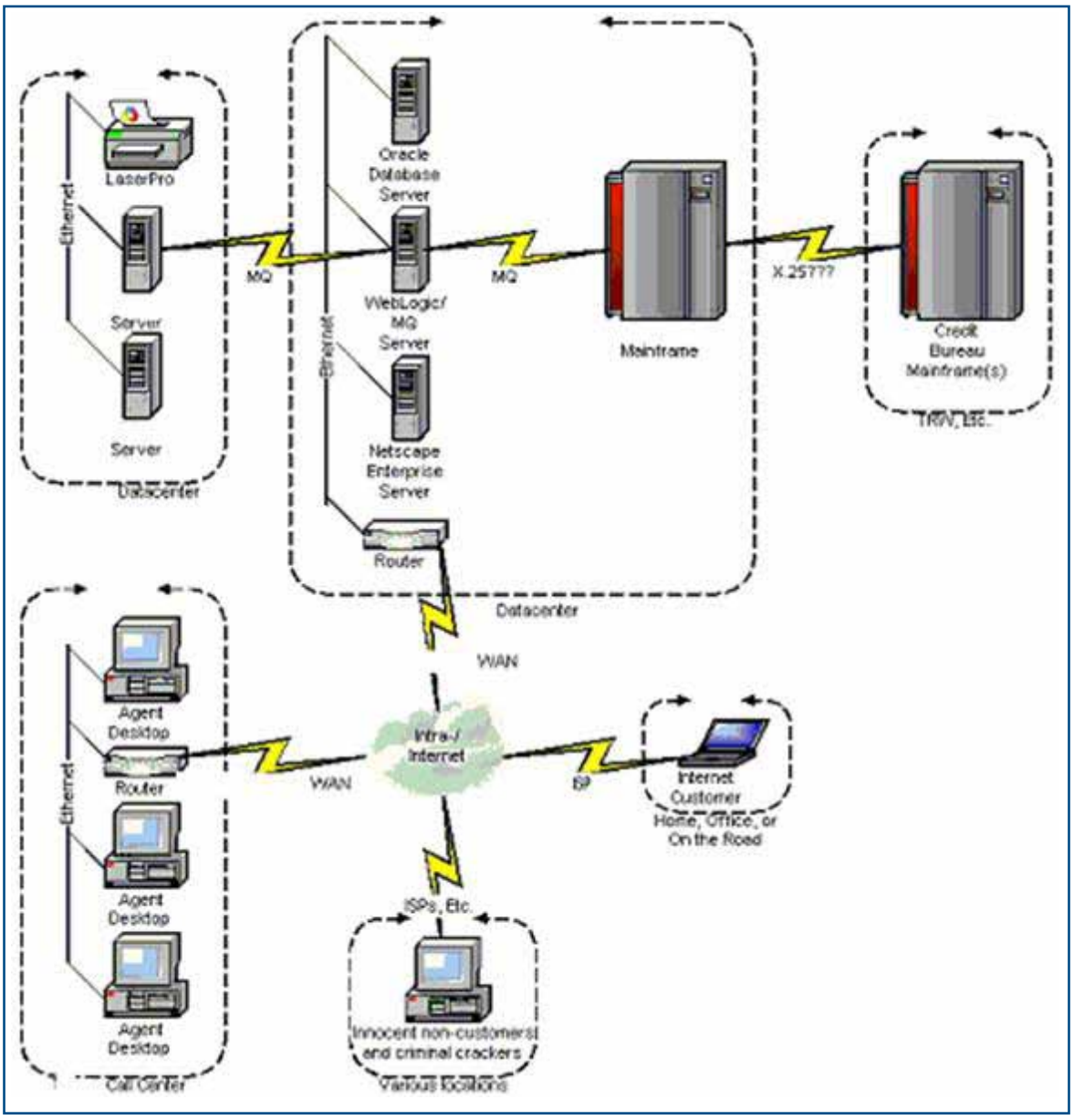


Figure 2: A realistic test environment

minimal environment, shown in Figure 1. This environment was okay for basic functionality and compatibility testing, but it was much, much smaller than the typical customer data center. After selling the application to one customer with a large, complex network and large, complex security dataset, the customer found that one transaction that the customer wanted to run overnight, every night, took 25 hours to complete!

While on this topic, let me point out a key warning: For systems that deal with large datasets, don't forget that the data is a key component of the test environment. You could test with representative hardware, operating system configurations, and cohabitating software, but, if you forget the data, your results might be meaningless.

Now, here's a case study of building a proper test environment. For a banking application, my associates and I built a test environment that mimicked the production environment, as shown in Figure 2. Tests were run in steps where each step added 20 users to the load, looking for the beginning of non-linear degradation in performance (i.e., the so-called "knees" or "hockey-sticks" in the performance curve). We put realistic loads onto the system from the call center side of the interface.

The only differences between the test and production environments were in the bandwidth and throughput limits on this wide-area network that tied the call center to the data center. We used testing and modeling to ensure that these differences would not affect the performance, load, and reliability results in the production environment. In other words, we made sure that, under real-world conditions, the traffic between call center and data center would never exceed the traffic we were simulating, thus ensuring that there was no hidden chokepoint or bottleneck there.

This brings us to a key tip: To the extent that you can't completely replicate the actual production environment or environments, be sure that you identify and understand all the differences. Once the differences are known, analyze carefully how they might affect your results. If there are more than one or two identified differences that can materially affect your test results, it will become very hard to extrapolate to the real world. That will call the validity of your test results into question.

Lesson 2: Real-World Loads

Once you have a realistic test environment in place, you are now ready to move to the next level of realistic performance, load, and reliability testing, which is the use of realistic transactions, usage profiles, and loads. These scenarios should include not just realistic usage under typical conditions. They should also include: regular events like backups; time-based peaks and lulls in activity; seasonal events such as holiday shopping and year-end closing; different classes of users including experienced users, novice users, and special-application users; and, allowance for growth for the future. Finally, don't forget about external factors such as constant and variable LAN, WAN, and Internet load, the load imposed by cohabiting applications, and so forth.

This brings us to another key tip: Don't just test to the realistically foreseen limits of load, but rather try what my associates and I like to call "tip-over tests." These involve increasing load until the system fails. At that point, what you're doing is both checking for graceful degradation and trying to figure out where the bottlenecks are.

Here's a case study of testing with unrealistic load. One project we worked on involved the development of an interactive voice response server, as you would use when you do phone banking. The server was to support over 1,000 simultaneous users. A vendor was developing the software for the telephony subsystem. However, during subsystem testing, the vendor's developers tested the server's performance by generating load using half of the telephony cards on the system under test, as shown in Figure 3.

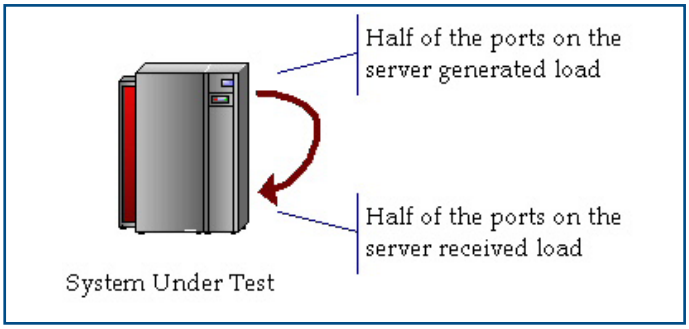


Figure 3: Improper load generation

Based on a simple inspection of the load generating software, the system load imposed by the load generators was well below that of the telephony software we were testing. My associates and I warned the vendor and the client that these test results were thus meaninglessness, but unfortunately our warnings were ignored.

So, it was no surprise that the tests, which were not valid tests, gave passing results. As with all false negatives, this gave project participants and stakeholders false confidence in the system.

My associates and I built a load generator that ran on an identical but separate host as shown in Figure 4. We loaded all the telephony ports on the system under test with representative inputs. The tests failed, revealing project-threatening design problems, which I'll discuss in a subsequent section of this article.

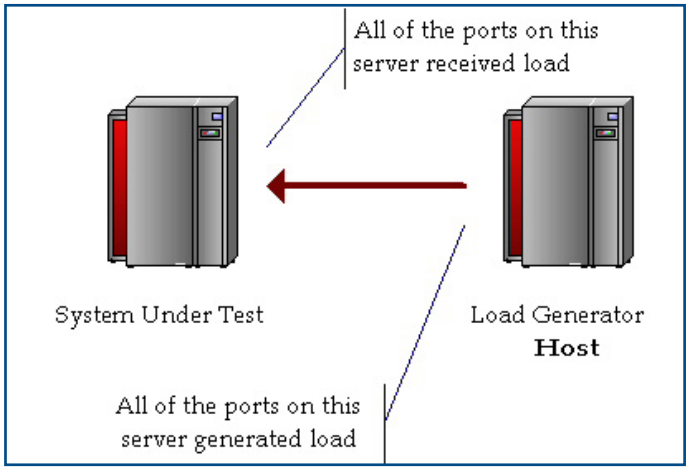


Figure 4: Proper load generation

So, this brings us to a key tip: Prefer non-intrusive load generators for most performance testing. It also brings up a key warning: Performance, load, and reliability testing of subsystems in unrealistic system settings yields misleading results.

Now, let me follow up with another key tip, to clarify the one I just gave: Intrusive or self-generated load can work for some load and reliability tests. Let me give you an example.

I once managed a system test team that was testing a distributed Unix operating system. This operating system would support a cluster of up to 31 systems, which could be a mix of mainframes and PCs running Unix, as shown in Figure 5. For load generators, we built simple Unix/C programs that would consume resources like CPU, memory, and disk space, as well as using files, interprocess communication, process migration, and other cross-network activities.

Basically, these load generators, simple though they were, allowed us to create worst-case scenarios in terms of the amount of resource utilization as well the number of simultaneously running programs. No application mix on the cluster could exceed the load created by those simple programs.

Even better, randomness built into the programs allowed us

Subscribe for the printed issue!

Please fax this form to +49 (0)30 74 76 28 99, send an e-mail to info@testingexperience.com or subscribe at www.testingexperience-shop.com:

Billing Adress

Company: _____

VAT ID: _____

First Name: _____

Last Name: _____

Street: _____

Post Code: _____

City, State: _____

Country: _____

Phone/Fax: _____

E-mail: _____

Delivery Address (if differs from the one above)

Company: _____

First Name: _____

Last Name: _____

Street: _____

Post Code: _____

City, State: _____

Country: _____

Phone/Fax: _____

E-mail: _____

Remarks: _____

1 year subscription

☐ **32,- €**
(plus VAT & shipping)

2 years subscription

☐ **60,- €**
(plus VAT & shipping)

Date

Signature, Company Stamp

to create high, spiking, random loads. We would sustain these loads for up to 48 hours. If none of the systems crashed, none of the network bridges lost connectivity to one or both networks, no data were lost, and if no applications terminated abnormally, then that was a passing test result.

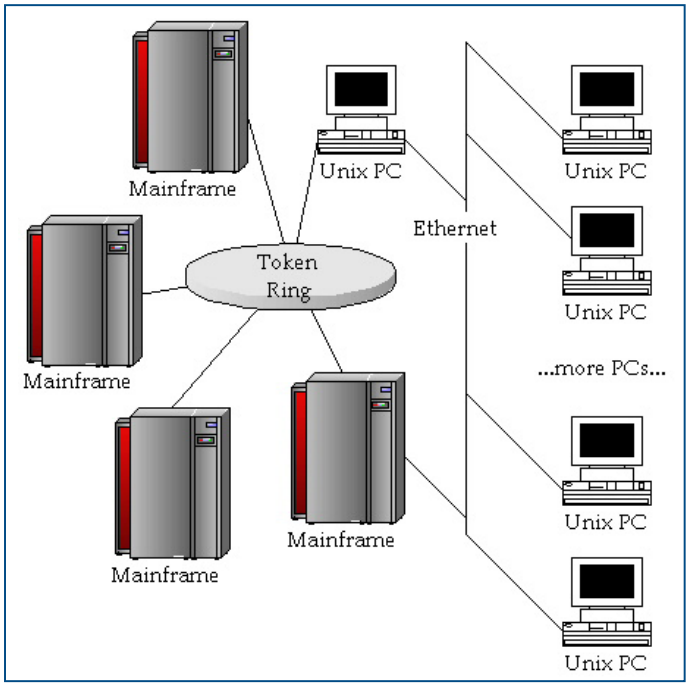


Figure 5: Maximum network configuration for the operating system

Lesson 3: Test the Tests - and the Models

Okay, so now it's clear that realism of the environment and the load are important. I mentioned earlier that performance, load, and reliability testing are complex endeavors, and that means that it's easy to screw them up. These two areas of realism are especially easy to screw up.

So, if realism of environment and load is important, how can we be sure we got it right, before we go live or ship? The key here is to build a model or simulation of your system's performance and use the model or simulation both to evaluate the performance of the system during the design phase and to evaluate the tests during test execution. In turn, since the model and simulation are equally subject to error, the tests also can evaluate the models. Furthermore, a model or simulation which is validated through testing is useful for predicting future performance and for doing various kinds of performance "what-iffing" without having to resort to expensive performance tests as frequently.

There are a couple of key tips to keep in mind regarding modeling and simulating. First, spreadsheets are good for initial performance, load, and reliability modeling and design. Second, once these spreadsheets are in place, you can use them as design documents both for the actual system and for a dynamic simulation. The dynamic simulations allow for more "what-if" scenarios and are useful during design, implementation, and testing.

Here's a case study of using models and simulations properly during the system test execution period. On an Internet appliance project, we needed to test server-side performance and reliability at projected loads of 25,000, 40,000, and 50,000 devices in the field. In terms of realism, we used a production environment and a realistic load generator, which we built using the development team's unit testing harnesses. The test environment, including the load generators and functional and beta test devices, are shown in Figure 6. Allowing some amount of functional and beta testing of the devices during certain carefully selected periods of the performance, load, and reliability tests gave the test team and the beta users a realistic sense of user experience under load.

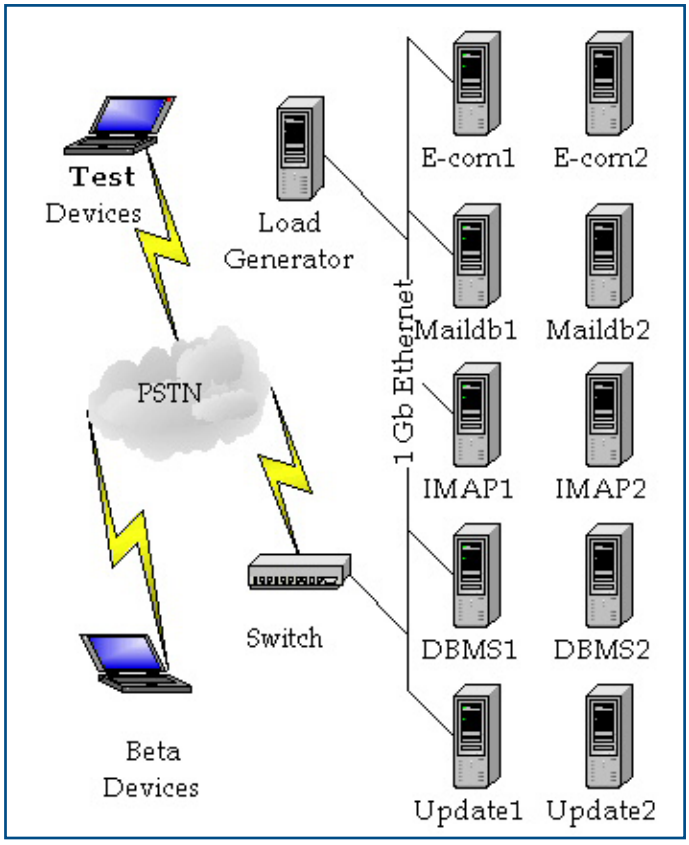


Figure 6: Internet appliance test environment

So far, this is all pretty standard performance, load, and reliability testing procedure. However, here's where it gets interesting. Because a dynamic simulation had been built for design and implementation work, we were able to compare our test results against the simulation results, at both coarse-grained and fine-grained levels of detail.

Let's look first at the course-grained level of detail, using an extract from our performance, load, and reliability test results presentation. We had simulation data for the mail servers, which were named IMAP1 and IMAP2 in Figure 6. The simulation predicted 55% server CPU idle at 25,000 devices. Table 1 shows the CPU idle statistics under worst-case load profiles, with 25,000 devices. We can see that the test results support the simulation results at 25,000 devices.

Server	CPU Idle
IMAP1	59%
IMAP2	55%

Table 1: Server CPU idle figures during performance tests

Table 2 shows our more fine-grained analysis for the various transactions the IMAP servers had to handle, their frequency per connection between the device and the IMAP server, the average time the simulation predicted for the transaction to complete, and the average time that was observed during testing. The times are given in milliseconds.

Transaction	Frequency	Simulation	Testing
Connect	1 per connect	0.74	0.77
Banner	1 per connect	2.57	13.19
Authorize	1 per connect	19.68	19.08
Select	1 per connect	139.87	163.91
Fetch	1 per connect	125.44	5,885.04

Table 2: Server transactions and times, simulated and observed during testing

Testen für Entwickler

28.06.10-29.06.10	**Berlin**
23.08.10-24.08.10	**Berlin**
18.10.10-19.10.10	**Berlin**
29.11.10-30.11.10	**Berlin**

Während die Ausbildung der Tester in den letzten Jahren große Fortschritte machte – es gibt mehr als 13.000 zertifizierte Tester alleine in Deutschland – wird die Rolle des Entwicklers beim Softwaretest meist unterschätzt. Dabei ist er beim Komponententest oftmals die treibende Kraft. Aus diesem Grunde ist es wichtig, dass auch der Entwickler Grundkenntnisse im Kernbereichen des Softwaretestens erlangt.

http://training.diazhilterscheid.com

IREB -

Certified Professional for Requirements Engineering - Foundation Level

25.08.10-27.08.10 Berlin

26.10.10-28.10.10 Berlin

01.12.10-03.12.10 Berlin

Die Disziplin des Requirements Engineering (Erheben, Dokumentieren, Prüfen und Verwalten) stellt die Basis jeglicher Software-Entwicklung dar. Studien bestätigen, dass rund die Hälfte aller Projekte scheitern aufgrund fehlerhafter oder unvollständiger Anforderungsdefinitionen.

http://training.diazhilterscheid.com

Es gibt nur einen Weg zum Glück.

gebrauchtwagen.de
Autos zum Verlieben.

As you can see, we had good matches with connect and authorize transactions, not a bad match with select transactions, a significant mismatch with banner transactions, and a huge discrepancy with fetch transactions. Analysis of these test results led to the discovery of some significant problems with the simulation model's load profiles and to significant defects in the mail server configuration.

Based on this case study, here comes another key tip: Be careful to design your tests and models so that you can compare the results between the two. Notice that the comparisons I just showed you rely on being able to do that.

Lesson 4: Invest in Tools, but Don't Splurge

Almost always, tools are necessary for performance, load, and reliability testing. Trying to do these tests manually is not an industry best practice, to put it mildly.

When you think of performance testing tools, likely one or two come to mind, accompanied by visions of large price tags. However, there are many types of suitable tools, some commercial, some open-source, and some custom-developed. For complex test situations, you may well need a combination of two or three of these types.

So here's the first key tip: Rather than immediately starting with the assumption you'll need to buy a particular tool, follow best practices in test tool selection. First, create a high-level design of your performance, load, and reliability test system, including the test cases, the test data, and the automation strategy. Second, identify the specific tool requirements and constraints for your test system. Third, assess the tool options to create a short-list of tools. Fourth, hold a set of competitive demos between the various vendors and with the open-source tools. Finally, do a pilot project with the demonstration winner. Only after assessing the results of the pilot should you make a large, long-term investment in a particular tool.

An obvious but easy-to-forget point is that open-source tools don't have marketing and advertising budgets. So, they won't come looking for you. Instead, search the Internet, including sites like sourceforge.net, to find open-source tools.

While on this note of open-source, free tools, here's a key warning: Don't overload on free tools. Even free tools have costs of ownership, and risks. One risk is that everyone in your test team will use a different performance, load, and reliability testing tool, resulting in a Tower of Babel and significant turnover costs. It happened to us on the interactive voice response server project, when we couldn't re-use scripts created by the telephony subsystem vendor's developers because they wouldn't coordinate with us on the scripting language to be used.

I want to reinforce this issue of making a tool selection mistake with a case study. It's one client's experience, again from before they became my client, but it has been repeated many, many times.

One of my clients attended a software testing conference, which is a good idea. They talked to some of the tool vendors at the conference, which is also a good idea. However, they selected a commercial tool based solely on a salesperson's representations that it would work for them, which, of course, is a bad idea.

When they got back to their offices, they realized they couldn't use the tool, because no one on the team had the required skills. So, they called the tool vendor's salesperson. Based on his recommendation, they hired a consultant to automate all their tests for them. That's not necessarily a bad idea, but they forgot to put in place any transition plan to train staff or maintain tests as the tests were completed. Furthermore, they tried to automate at an unstable interface, which in this case was the graphical user interface.

So, after six months, the consultant left. The test team soon found that they could not maintain the tests. They also found that they could not interpret the results, and had more and more false positives arising as the graphic user interface evolved. After a short period, the use of the tool was abandoned, after a total investment of about $500,000.

Here's a successful case study of test tool selection and implementation. While working on a complex system with a wide-area network of interactive voice response servers connected to a large call center, we carefully designed our test system, then selected our tools using the process mentioned earlier. We used the following tools:

- QA Partner (now Silk Test) to drive the call center applications through the graphical user interface.
- Custom-built load generators to drive the interactive voice response server applications.
- Custom-built middleware to tie the two sides together and coordinate the testing end-to-end.

We used building blocks such as the TCL scripting language for our test of custom-built test components. The overall test system architecture is shown in Figure 7. The total cost of the entire system, including the labor to build it, was under $100,000, and it provided a complete end-to-end test capability, including performance, load, and reliability testing.

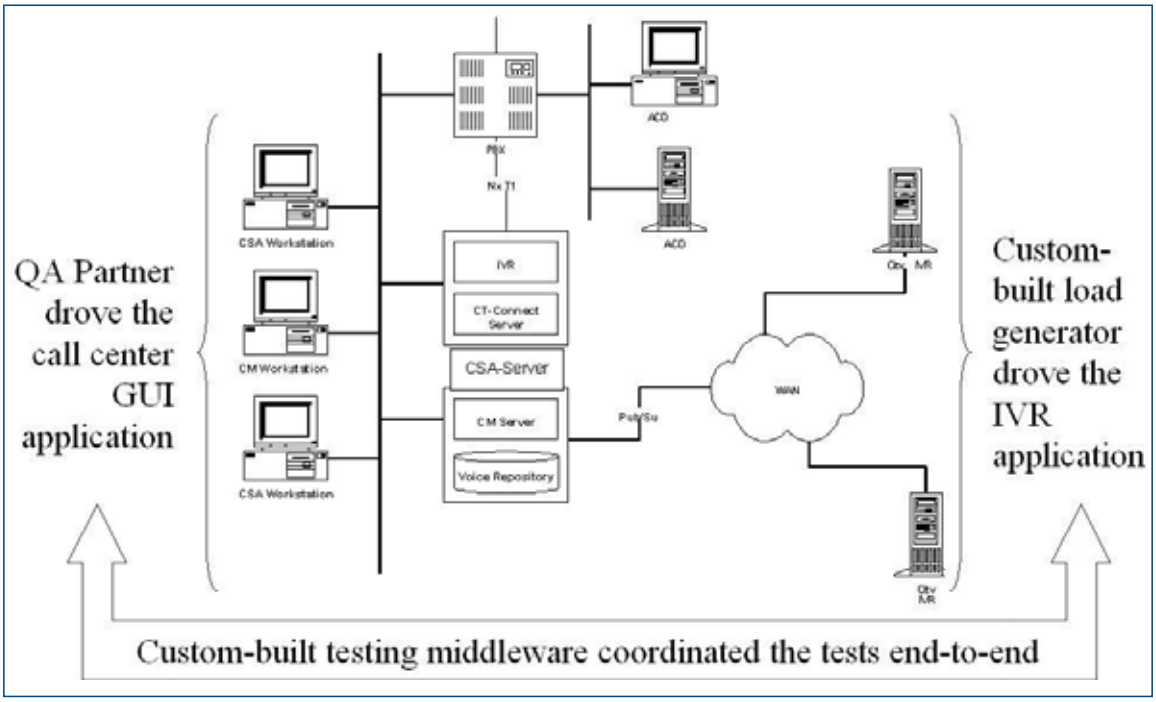


Figure 7: Good test system architecture

testing experience

Knowledge Transfer

limited places

Rapid Software Testing

3-day tutorial with

Michael Bolton

July 5-7, 2010 in Amsterdam, The Netherlands

Rapid testing is a complete methodology designed for today's testing, in which we're dealing with complex products, constant change, and turbulent schedules. It's an approach to testing that begins with developing personal skills and extends to the ultimate mission of software testing: *lighting the way of the project by evaluating the product*. The approach is consistent with and a follow-on to many of the concepts and principles introduced in the book *Lessons Learned in Software Testing: a Context-Driven Approach* by Kaner, Bach, and Pettichord.

The rapid approach isn't just testing with a speed or sense of urgency; it's mission-focused testing that eliminates unnecessary work, assures that important questions get asked and necessary work gets done, and constantly asks what testing can do to help speed the project as a whole.

One important tool of rapid testing is the discipline of exploratory testing—essentially a testing martial art. Exploratory testing combines test design, test execution, test result interpretation, and learning into a simultaneous, seamless process that reveals important information about the product and finds a lot of problems quickly.

www.testingexperience.com/knowledge_transfer.html

1400,- €
(plus VAT)

Lesson 5: Start Early and Stay on Course

Finally, a last lesson, as important as all the rest, is to begin thinking about and resolving performance, load, and reliability problems from day one of the project. After all, performance, load, and reliability problems often arise during the initial system design, when resources, repositories, interfaces, and pipes are designed with mismatched or too little bandwidth. These problems can also creep in when a single bad unit or subsystem affects the performance or reliability of the entire system. Even when interfaces are designed properly, during implementation, interface problems between subsystems can arise, creating performance, load, and reliability problems. Finally, slow, brittle, and unreliable system architectures usually can't be patched into perfection or even into acceptability.

So here are two tips related to this lesson to start with. First, as alluded to earlier, you can and should use your spreadsheets and dynamic simulations to identify and fix performance, load, and reliability problems during the design phase. Second, don't wait until the very end, but integrate performance, load, and reliability testing throughout unit, subsystem, integration, and system test.

And here's a warning on this topic: If you do wait until the very end and try throwing more hardware at the problems, understand that this approach might not work. The later in the project we are, the fewer options we have to solve problems, and this is doubly true for performance- and reliability-related design and implementation decisions.

Let me expand on a case study mentioned early to help reinforce this warning. During the interactive voice response server project mentioned earlier, one senior technical staff member built a spreadsheet model that predicted many of the performance and reliability problems we found in later system testing.

Sadly, there was no follow-up on his analysis during design, implementation, or subsystem testing. Once my associates and I got involved, during integration and system testing, the late discovery of problems led project management to try to patch the interactive voice response server, either by tweaking the telephony subsystem or adding hardware.

And here's what happened. Patching of one problem would improve performance and resolve that particular problem. However, we would discover the next bottleneck as we proceeded through our testing. Instead of solving the problem, we engaged in a slow, painful, onion-peeling process which didn't resolve the performance problems, as shown in Figure 8.

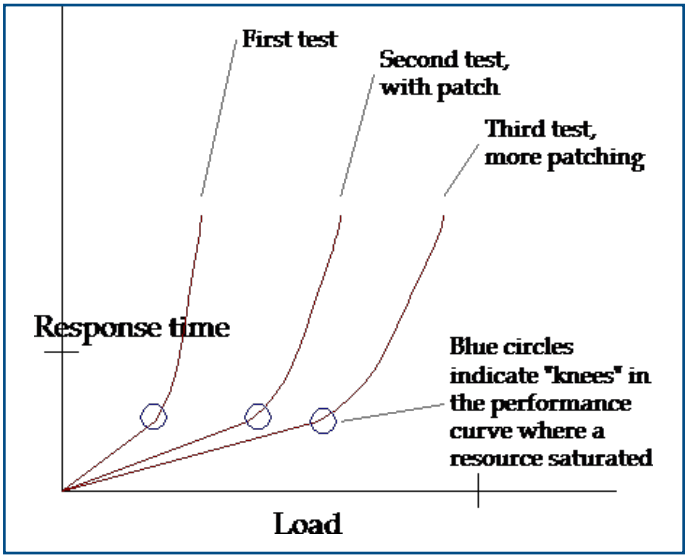


Figure 8: The problem with patching

Now, let's see a case study where we did apply performance modeling and testing consistently and throughout the lifecycle. For the Internet project discussed earlier, we started with a spreadsheet during initial server system design, as mentioned. That initial server design was turned into a simulation, which was fine-tuned as detailed design work continued. Throughout development, performance testing of units and subsystems was compared against the simulation. This meant that relatively few surprises were found during system testing.

This brings us to a final key tip: It's not likely that you'll get these models and simulations right from the very start, so plan to iteratively improve the models and simulations. Your tests will probably have problems at first, too, so plan to iteratively improve the tests to resolve discrepancies between the predicted and observed results.

So, Where Do You Stand?

In this article, I've shared with you five hard-won lessons, supported by a number of key tips and warnings, about performance, load, and reliability testing. If you're involved in performance, load, and reliability testing, the odds are that there are some useful ideas in this article that you can put to work for you. Now it's up to you to apply those ideas.

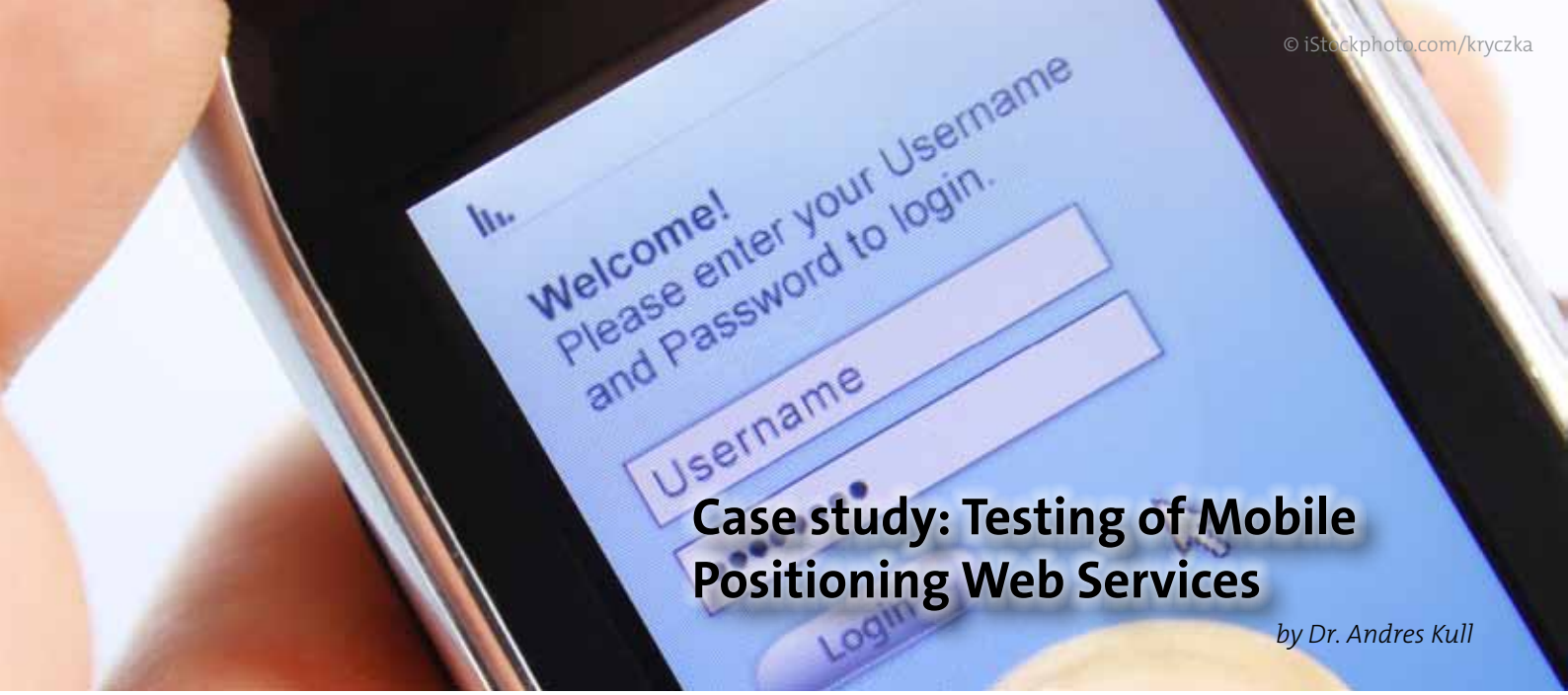
Start by assessing your current practices in performance, load, and reliability testing. How well—or how poorly—does your company do in applying each of these lessons? Which common mistakes are you making? What warnings are you not heeding? Which tips for success could you apply?

Based on this assessment, see if you can put together a short action plan that identifies the organizational pain and losses associated with the current approach, and a path towards improvement. The goal is to avoid expensive, unforeseen field failures by assessing and improving your performance, load, and reliability risk mitigation strategy throughout the lifecycle.

Once you have your plan in place, take it to management. Most of the time, managers will be receptive to a carefully thought-out plan for process improvement that is based on a clearly articulated business problem and a realistic solution. As you implement your improvements, don't forget to continually re-assess how you're doing. I wish you the best of luck in learning your own lessons in performance, load, and reliability testing through case studies of success!

Biography

With a quarter-century of software and systems engineering experience, Rex Black is President of RBCS (www.rbcs-us.com), a leader in software, hardware, and systems testing. For over a dozen years, RBCS has delivered services in consulting, outsourcing and training for software and hardware testing. Employing the industry's most experienced and recognized consultants, RBCS conducts product testing, builds and improves testing groups and hires testing staff for hundreds of clients worldwide.



Case study: Testing of Mobile Positioning Web Services

by Dr. Andres Kull

The case study was conducted to test Reach-U's (http://www.reach-u.com/) Workforce Management (WFM) application, a web-based mobile positioning application developed by Reach-U. WFM is an application that can observe a workforce's movement history, location and send SMSs to personnel via the web browser client. WFM's users are primarily companies that would like to track and plan their workforce locations in real time.

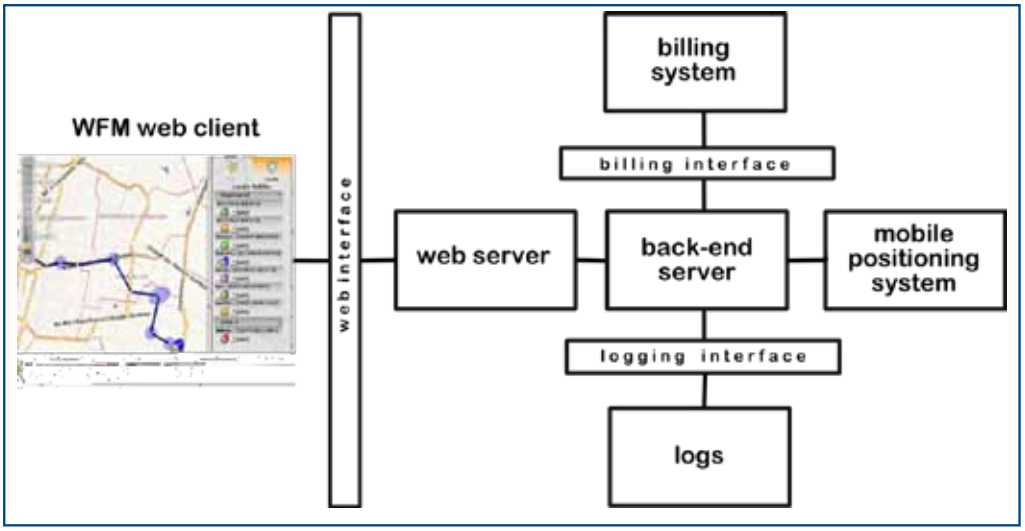


Figure 1: General architecture of WFM

The case study was performed to evaluate the applicability of model-based testing for testing web services via functional and load testing – use cases where an *operator* uses a WFM web-client to locate *targets* were tested. Only servers that are part of WFM were in the scope of the implementation under test (IUT). A web client was simulated by the test system. It is worth noting that testing the functionality of the web client itself was out of the scope of the case study.

The following aspects were evaluated in the case study:

- Overall feasibility of model-based testing technology for testing web services.
- Applicability of model-based testing technology for load testing.

Implementation

In the case study we used a test generator producing TTCN-3 test scripts automatically from the State Model (UML) of the IUT, a full-feature TTCN-3 test development and execution platform and the JMeter (JMet) load tester. Additionally, a system adapter was developed to communicate between WFM and the TTCN-3 test development and execution platform as well as the JMeter load tester.

Firstly, the TTCN-3 scripts generator was used to generate TTCN-3 test cases from the system model. The test cases generated from the model were executed on the TTCN-3 test development and execution platform. Test cases controlled the execution of JMeter as well. JMeter was used to generate the load for the server infrastructure of WFM. The test cases instructed JMeter to vary the load. Therefore, the test cases were communicated to two counterparts – the IUT and JMeter. In between, the system adapter was developed. The system adapter connected to the TTCN-3 test development and execution platform to WFM and JMeter. The interface of the system adapter towards WFM provided HTTP message traffic over TCP/IP. In this way the test cases ran on the TTCN-3 execution platform and simulated the WFM web-client. The interface of the system adapter to JMeter allowed JMeter to be controlled by HTTP GET messages over TCP/IP.

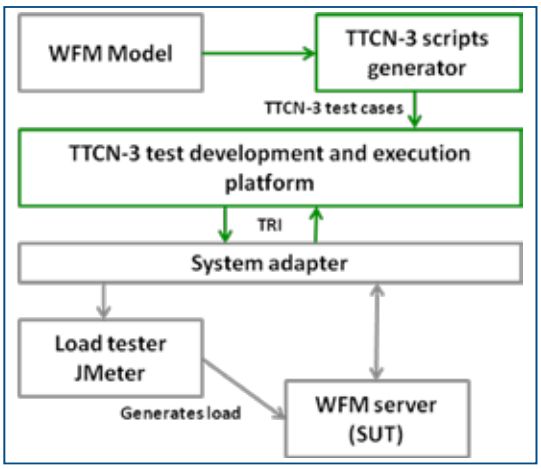


Figure 2: The test environment of the WFM

Results

- **Overall feasibility of model-based testing technology for testing web services.**

WFM is a complex distributed web application consisting of many server applications distributed/duplicated on different hosts and protected by a load balancer from overload situations. The case study demonstrated the feasibility of the model-based testing (MBT) technology using TTCN-3 script's automatic generator for functional testing of an IUT interface over web services.

- **Applicability of model-based testing technology for load testing**

In the case study, the load tests and functional tests were used in parallel to test the IUT. The case study demonstrated the applicability of the model-based testing in general for usage in load tests to increase the quality of overall testing. Moreover, it showed that MBT technology could be used beyond the borders of traditional functional black-box testing. In the case study, the functional tests were generated out of the WFM model and used in conjunction with a load tester in order to prove the expected behavior of the IUT in the event of heavy loads.

Biography

Andres Kull is a co-founder and CEO of Elvior OÜ. Elvior is a software testing services and tools company specialized in embedded and distributed software testing. Andres has a PhD (2009) in Informatics and Communication and an MSc (1985) in Control Systems from the Tallinn University of Technology. His 25-year professional career spans the areas of control systems for the chemical industry and model-based engineering of software-intensive embedded and distributed systems. Andres has held researcher positions at the Tallinn University of Technology and Technical Research Centre of Finland (VTT). His current specialities, in addition to the management of Elvior, are embedded software development and testing, model-based testing, and TTCN-3 testing and test tools.

Is Your Application's Performance Predictable?

by Suri Chitti

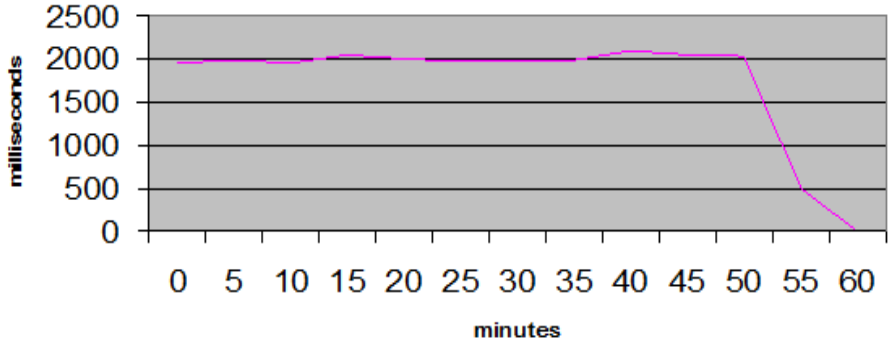
Let me start with a simple, yet realistic example. I have a web-based application which allows a student to register for a course. Identifying a load test for this application is straightforward. Say I have 400 students on the rolls and a student, on the average registers up to 5 courses each semester. With a little investigation, I may identify a maximum load of 200 students simultaneously trying to register for a single course. I may also find that an average registration takes 10 minutes and the general trend is that students register for their courses in one shot. Hence peak loads may stretch up to 50 minutes. To briefly outline how I would test this scenario, my load test would invoke 200 students (randomly chosen from a data sheet listing 400 students and the courses they need to register for) and have them simultaneously register for their courses, one course after the other. I would run this test for an hour (expecting registrations to be complete by then). Amongst many parameters that I will be monitoring, I am particularly interested in transaction response time. This is a key parameter that translates to how much time end users wait for responses from the application during specified transactions. I will stick to this parameter for the purpose of my discussion, though it will apply to any parameter of interest. At the end of my test I generate a report that will display response time plotted for the duration of the test (see figure 1). I will be happy to see that the transaction response time has not significantly crossed 2000 milliseconds at any point. The result indicates that during peak activity and maximum load my application will not take more than 2 seconds to respond. Assuming my acceptable response time limit is more, there is reason for cheering.

Yet, there is more work to do. I have established acceptable performance under peak conditions. However, I have not established predictability. I now need to try and understand the range of loads my application will be subject to. Assuming I investigate and find that loads range between 25 to 200 simultaneous registrations, I wish to see response times beginning at 25 registrations and ending at 225 with stepped increases of 25 at every six minute intervals in my run. In my result, I wish to see a responsive trend in transaction response time with load, similar to the green graph in figure 2. The joyful aspect with a responsive trend is that there is a linear correlation between load and response time in my operational load range. The correlation is a strong indicator of predictability. I will be unhappy if I cannot see the correlation, or even worse, if I see an erratic (red graph in figure 2) trend in response time with load.

If my application architect uses a web engine that can take loads very much above the range I have identified, I may notice a flat trend (purple graph in figure2) indicating there is no significant increase in response time with load in my range. However, my architect is, for performance purposes, using an elephant gun for bird hunting, and that will have its own problems, least of which is the money wasted on needless performance characteristics.

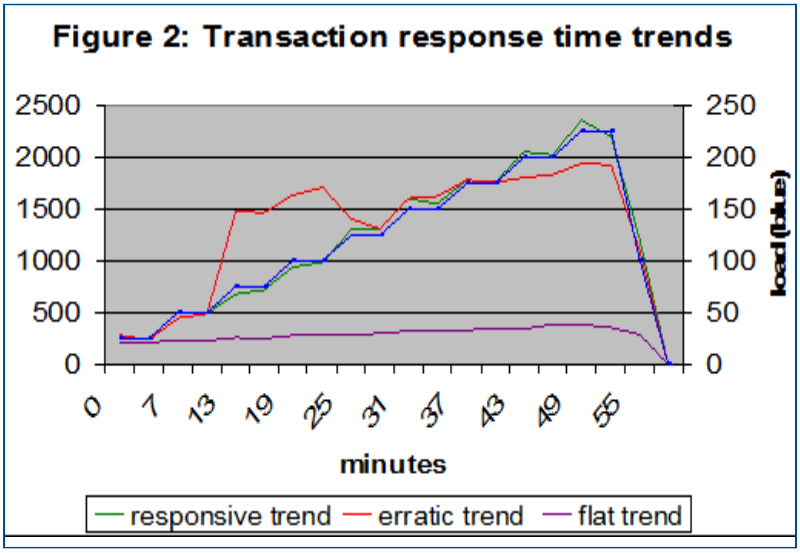
An application that shows a good correlation between transaction response time and load in the operational load range during testing is likely to also be predictable during operation. On the other hand, applications that erratically respond to load, or have patches of unpredictabilty in their operational load range, are also likely to have unpredictable problems during operation.

Figure 1: Transaction response time at peak load



This is even when the transaction response time remains below a certain value during the test (as depicted in the red curve in figure 2). Predictability is important for applications that are routinely exposed to different loads depending on occasion. For example, retail store applications encounter different loads during weekdays, holidays and festival seasons. Going the extra step to identify the operational range of loads and establishing predictability over this range is worth the effort.

Figure 2: Transaction response time trends



Biography

Suri Chitti is a software test professional. His extensive experience prominently includes testing software in several industrial verticals, automated functional testing and performance testing. He has worked for major companies that are involved in the production of commercial software, across the world. Besides testing, he enjoys swimming and watching the sky at night.

Testing e-learning platforms based on two perspectives: technological and pedagogical

by Delvis Echeverría Perez, Roig Calzadilla Díaz & Meiby Yeras Perez

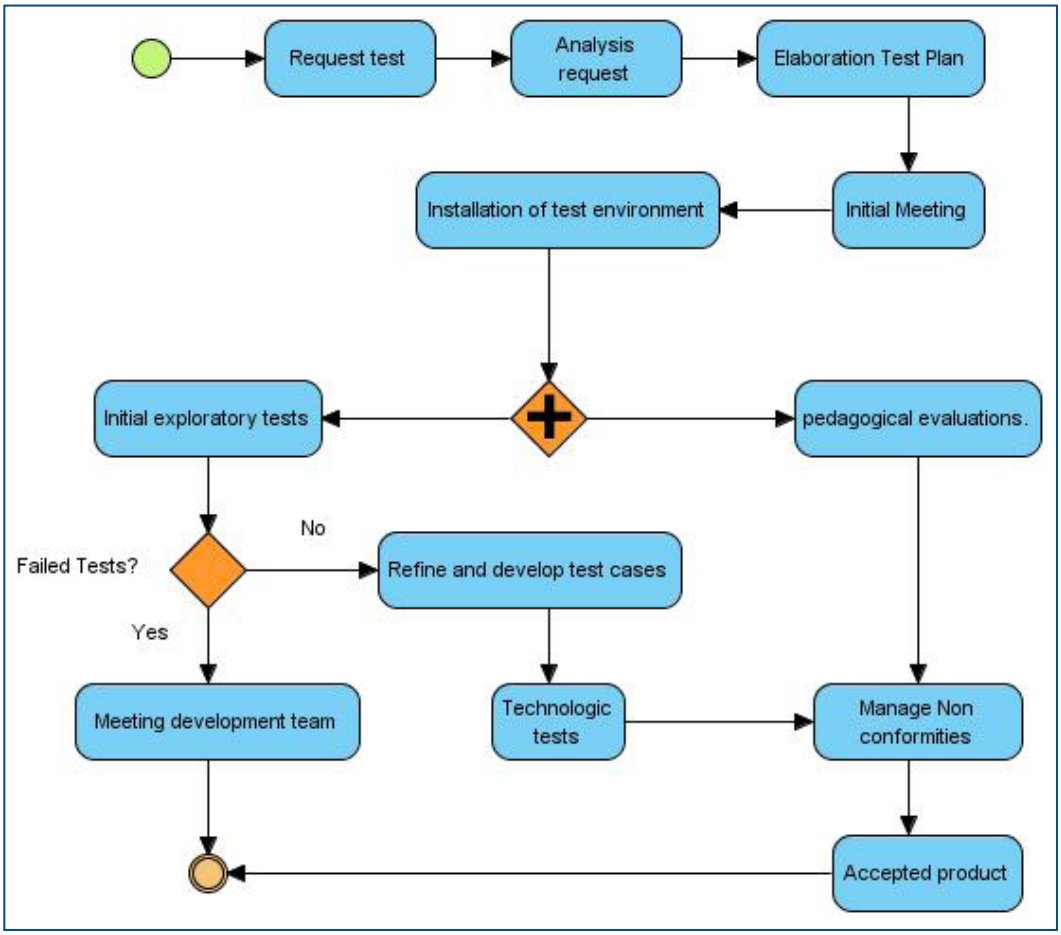
The popularity of online learning experiences offers great opportunities and benefits. However, significant challenges exist where the demand and interest in a product with higher quality are imposed, and these must urgently be tackled. Providing quality to e-learning platforms is critical for the organizations that develop them. Taking this situation into account, Industrial Laboratory Testing Software incorporated the evaluation of e-learning platforms into its services. This evaluation is done through a strategy that is structured in two perspectives: pedagogical and technological. It takes advantage of the teachers of its specialists, where the majority of these are informatics engineers. All evaluations have been satisfactory. With an absence of this testing on multiple platforms there could be both functional and methodological problems.

The 21st century knowledge-based society has declared the need for new spaces for its educational processes. The use of the Internet in education is something relatively recent. One could say that it is still in its initial phase. Many people observing the advantages and possibilities that the Internet offers have taken the initiative by creating their own classrooms online, and they have often made mistakes and needed others to help reorganize the process. Creating an educational system for the web is not a question of simply digitizing educational texts or reading electronic books. Initiatives of this sort can waste all the advantages and opportunities that the Internet offers for education. The development of an educational environment online entails some specific factors that differentiate it from an educational system based on a personal PC. The development of an online learning environment also has some special features that differentiate it from a conventional educational system.

Testing e-learning platforms.

Many of the hopes placed in the e-learning have not been confirmed in educational practice. From our point of view the debate has been centered too much on the technical components, for example in the platforms, forgetting what critical pedagogic variables need to be considered. In this article we consider some of these factors: technological competence of professors and students, institutional support, contents, social interaction, media, methodologies. [1]

Due to the characteristics of these systems, a strategy was developed for testing, keeping in mind not only the technological perspective but also incorporating pedagogical aspects based on



the experience of educational specialists educationally and their work with such learning environments.

Testing Procedure for e-learning platforms.

The procedure begins with a request for testing by customers, it is analyzed and a test plan is developed. At the initial meeting all details of testing are agreed and the installation of test environment is commenced. Once the environment is ready, two activities are started simultaneously: Initial exploratory testing and pedagogical evaluation. After completion of these activities, errors (non-conformities) are managed, concluding with acceptance of the product.

Technological Perspectives

Testing e-learning from a technological perspective focuses on assessing the quality of technology platforms through which e-learning is implemented. Many companies, independent agencies and universities developed assessment tools to determine quality based on a number of criteria, including functionality, accessibility, availability and potential hardware and software, and usability for the student, teacher, administrator.[4]

Current evaluation of e-learning platforms is characterized by software testing, which is merely technical and applies automated tools, checklists, manual tests and static analysis.

Pedagogical Perspectives

Testing e-learning from a pedagogical perspective uses approaches applied from traditional training and includes the models, research methods and data collection techniques applied in formal education. Approaches to drawing up pedagogical proposals for the evaluation of e-learning draw on models such as those of Stufflebeam (CIPP) (1987) Kirkpatrick (1999), Vann Slyke [5]

The assessments are based on checklists prepared by teachers.

Technologic Perspective (testing)	Automated Tools	Checklists	Manual Testing
Static Assessments (Document generated)		x	x
Functional Testing.	x		x
Environment Testing		x	

Technologic Perspective (testing)	Automated Tools	Checklists	Manual Testing
Usability Testing		x	x
Volume Testing	x		
Load and Stress Testing	x		

Technologic Perspective (testing)	Automated Tools	Checklists	Manual Testing
Evaluation of the goals and contents.		x	
Evaluation of the strategy and process of learning.		x	
Evaluation of the materials used.		x	
Evaluations of student development.		x	
Evaluations of the tutor.		x	

Used tools:

Functional Testing: Selenium.

Load and Stress Testing: Jmeter.

Volume Testing: Data Generator.

Checklist:

The overall objective of the checklist is to evaluate the specifications of the [name of device] of software development projects. This template has been prepared to guide developers, quality specialists or technical experts.

Testing e-learning platforms was implemented on a project, registering and correcting most of the non-conformities (NC) detected by the application of automated tools, manual testing and checklists. Involving several testers, distributed by different types of testing, and generating the following non-conformities (NC):

Abbreviated Checklist					
Critical?	**Indicator**	**Evaluation**	**Applicable?**	**Number of items affected.**	**Comment.**
Define if the indicator to evaluate is critical or not.	Question	The way to evaluate the indicator. The indicator can be 0 and 1.	Can be the indicator applied to this case?	To Specify the number of errors found on the same indicator.	To Specify the remarks or suggestions.
Example!					
Critical	Is it easy to understand and memorize the information presented in the application?	1		3	
Critical	Has the platform, tools for monitoring student´s progress?	1		1	
Critical	Can the access make from any navigator: Internet Explorer, Mozilla Firefox?	1		1	

Two perspectives.	Non conformities	Number of testers
Static assessments (document generated)	23	4
Functional testing	12	11
Environment testing	16	11
Usability testing	15	11
Volume testing	5	3
Load and stress Testing	9	4
Evaluation goals and Contents	4	3
Evaluation strategy and process of learning.	13	3
Evaluation of materials used Evaluations of student development.	10	3
Evaluations of the tutor.	8	3

Here are some of the non-conformities identified during the evaluation process for different types of testing:

Static assessments: (documentation generated in the system)

1. Format errors according to the rules of the templates proposed.

Functional tests:

1. Some of the features described in the requirements that were agreed with the client are missing and sometimes were poorly implemented.
2. When data are inserted into the system detected a problem with validation.

Testing environment:

1. Used terms and abbreviations that are not previously defined.
2. The link did not work properly.

Evaluation of training materials:

1. There were different types of resources and, there are only images.

Assessments of the tutor:

1. Not all tutors have extensive experience as online tutors.
2. The program provides students throughout the course of the existence of technical mentoring for troubleshooting.

Strategy assessment and learning process:

1. The complexity of the tasks is not appropriate to the content and in relation to teaching skills of students.
2. Not sought at any time by students who value the work done by the tutors.

Conclusions

The rapid and expansive growth of e-learning requires research on the impact of this new educational and technological alternative. Currently for the evaluation of e-learning systems many people follow the trend of putting pedagogy over technology. The union of the quality of the product with the quality of testing education, removes the tendency to evaluate separately the aspects of technology and pedagogy. If educational and technological objectives of a system do not match with the expected results, the system is not usable and leads to failure. The proposal has identified a procedure for introducing both perspectives, for development testing used automated tools, checklists and manual tests. The intention the research proposal is that today's e-learning platforms have the required quality..

References.

1. Cabero Almenara, Julio, La calidad educativa en el e.Learning: sus bases pedagógicas, 2006.[:http://scielo.isciii.es/scielo.php?pid=S157518132006000700003&script=sci_arttext]
2. Marrero Ponce de León, Zenaida. Resolución No. 63/2005, 2007, Ministerio de la Informática y las Comunicaciones, Ciudad de La Habana, Cuba.
3. Colás Bravo, Pilar, Rodríguez López, Manuel. Evaluación de e-learning. Indicadores de calidad desde el enfoque sociocultural. [http://campus.usal.es/~teoriaeducacion/rev_numero_06_2/n6_02_art_colas_rodriguez_jimenez.htm].
4. Colás Bravo, Pilar, Rodríguez López, Manuel. Evaluación de e-learning. Indicadores de calidad desde el enfoque sociocultural. [http://campus.usal.es/~teoriaeducacion/rev_numero_06_2/n6_02_art_colas_rodriguez_jimenez.htm].

Biography



Delvis Echeverría Perez
27 years old, Engineer in Informatics Science, graduated in 2007 from the University Informatics Sciences (UCI) in Havana, Cuba. He is currently a Quality Specialist in the software testing department of Calisoft, where he leads the sub-group Automated Tools for Software Testing Professor at the UCI. He worked as assistant manager of development and quality assurance in software development projects. He has participated in Release and Acceptance Testing of Software, both in Cuba and abroad.

Roig Calzadilla Díaz
26 years old, Engineer in Informatics Science, graduated in 2007 from the University Informatics Sciences (UCI) in Havana, Cuba. He is currently a Quality Specialist in the Software Testing Department of Calisoft, where he is a coordinator of the Software Testing Department. From July 2008 to April 2009 he represented Calisoft in acceptance testing abroad (in Spain and Venezuela). Professor at the UCI. He has participated in Release and Acceptance Testing of Software, both in Cuba and abroad.

Meiby Yeras Perez
Student, North Fort Myers School, Fort Myers FL.

Column

Time to change a winning team?

by Rob van Kerkwijk

During times of recession, people often seek ways to save costs – when buying groceries in the supermarket or when investing in IT. There is also an ongoing discussion about the purchasing-attitude in the current economy and the impact it will have on a company's ability to innovate. A natural reaction of buyers or decision makers is to become conservative in spending. But are there advantages of a more active and innovation-driven attitude towards spending and investing? Who will be the winners at the end of the economic downturn?

My personal view is that the winners are those that wish to change and innovate. You will agree with me that in times when interest rates are low and prices of real estate are relatively low, the decision to buy a house can work out well in the long run. By looking at and acting on beneficial market conditions, we can take advantage in this economically challenging period.

The economy often plays a key role with regard to change and innovation. In a prosperous economy, the urge to opt for changes is relatively small, under the cloak of 'never change a winning team', and requires an organization that is always looking to innovate and win in the market. In periods of stagnating growth, companies will increasingly be looking at innovation from the perspective of working more efficiently and finding cost-effective measures to ultimately maintain their existing margins and turnover. It is less common in these tough times to aim for growth, even though it might be the best time to take advantage of changing market conditions.

The bottom line for any type of business change is that IT must have the ability to make a difference and enable the organization's objectives and strengthen the position of the internal and external IT organization. In my opinion, this is the great opportunity for our vendors, service providers and other channel parties to help their clients see the opportunity and innovate to drive new business and generate more revenue from existing customers. To use the famous words of the new US President: "The time for change is now".

My experience – since 1986 - in the dynamics of ICT-Industry in various positions at leading IT-vendors, provides a broad view and understanding of our industry. My career started at Unisys, where I had various positions (Tele) Sales – Marketing – Product management – Telesales manager with an interesting evolution of Unisys' Product Portfolio. At BMC Software Sales & Partner Manager roles experiencing the evolution of a Infrastructure Management Vendor to become one of the leading and state-of-the -art Business Service Management Vendors. At OutSystems I entered the Application Development area, where OutSystems' Agile Methodology is tied to Agile Technology which guarantees fast & flexible delivery of web solutions. Partner recruitment and management is the primary focus to support our strategy, where business Partners are key to our success. Change and innovation are keywords of my profile.

Testing SOA - An Overview

by Himanshu Joshi

Many testing techniques and methodologies developed over the years also apply to web services based SOA systems. Through functional, regression, unit, integration, system and process level testing, the primary objective of testing methodologies is to increase confidence that the target system will deliver functionality in a robust, scalable, interoperable and secure manner.

Web services are the foundation of modern Service Oriented Architecture (SOA). Typical web services include message exchange between front end and back end using SOAP requests and responses over the HTTP protocol.

A web service producer advertises its services to potential consumers through Web Services Description Language (WSDL) – which is an XML file that has details of available operations, execution end-points and expected SOAP request-response structures.

Techniques such as Black, White and Gray Box testing applied to traditional systems map also well into web services deployments. However, the following characteristics of web services deployments present unique testing challenges:

- Web services are basically distributed and are platform and language-independent.
- Web services can be linked to dependencies on other third-party Web Services that can change without notice.

SOA frameworks are a combination of web components, mid-tier components, back-end and legacy systems. Unlike traditional testing approaches, the SOA testing approach should include all the aspects of business processes and also the integration framework.

The SOA test strategy should not only focus on just the functionality or the front-end clients, but also on the integration layers. It is therefore necessary to keep all these factors in mind when creating an SOA test plan or strategy.

The following types of testing should be performed to ensure proper end-to-end testing which enables coverage of the integration framework:

- Functional testing
- Interoperability testing
- Performance testing
- Security testing

To test SOA, you need to go far beyond merely testing a user interface or browser screen, as you will never find an attractive UI while testing these apps. Some of the major challenges for SOA testing include:

- No user interface for the services in SOA
- Application's lack of visibility into loosely coupled business level services
- Dependency on availability of any internal or external services that offer a business function to perform end-to-end testing.

The following aspects needs to be kept in mind when preparing an SOA test approach:

- A web service is an application without UI.
- A GUI application should be developed
- Changes to web service interfaces often demand GUI changes
- Security testing of any web services that require encrypted and signed data is not possible
- A web service should be validated for the client as well as at the service level.
- The overall cost involved is very high for testing, development & maintenance.
- Maintaining a pool of testing resources with SOA domain knowledge

While testing web services, SOAP requests need to be generated manually or using a supported third-party vendor testing tool.

Integrating a traditional functional testing tool or other third-party tools of this kind is not feasible due to their nature and limitations. Therefore some other web services testing tools come into question.

These tools provide robust, reliable, scalable and interoperable solutions. They are suitable for UI less testing. Just enter the WSDL location and they can create basic functional tests for you.

Most of the tools even check your WSDL compliance to WS-I. They also support asynchronous testing and data-driven testing using data sources.

These test tools provide a solution for load testing and one of their best features is their ability to fully reuse the functional tests for load testing, which saves precious time.

Tools like SOAtest (http://www.parasoft.com) integrate with Qua-

lity Center, TestManager and VSTS Edition for testers. It features a host of new and extended capabilities, including Microsoft .NET WCF support, integration with Microsoft VSTS, automated stub creation and deployment from existing test suites, continuous load testing, and advanced XML validation. I have used it in a few of my projects for functional and load testing, and I found it easy to use.

Security and penetration testing can easily be done using tools like SOAtest and SOAPSonar(http://www.crosschecknet.com/). I have used SOAPSonar for security testing. In my opinion, this is by far one of the best tools for penetration testing of web services. It has many straightforward options that you can perform easily.

The tool also offers functional, performance, compliance, and vulnerability testing modes with comprehensive identity management testing along with complex WSDL and schema parsing. The latter is something which should not be missed while testing web services.

ITKO's Lisa Test (http://www.itko.com/) allows you to test web services along with ESB without writing any code. It allows you to rapidly design and execute your tests, plus it has very good reporting features.

Greenhat's GH Tester ((http://greenhatsoftware.com/) is another tool that helps in functional testing, integration testing and performance testing; it can even import BPEL projects.

If you are looking for open-source tools, then soapUI (http://www.soapui.org/) is the best. It is great if you want to test your web service for free, as most of the features that it has cater to basic web service testing. However, there is also a professional version available with some additional features, but this comes at a price.

Most of these tools integrate with popular test management tools from vendors such as HP/Mercury.

Test data generation for any WSDL is possible and easily maintainable. Depending on existing legacy systems and back-end applications, web services test tools can be easily tweaked by developing test scripts to provide an end-to-end solution for backward compatibility.

To conclude:

In order to test SOA successfully, you must verify all aspects of a web service, from WSDL validation, to unit and functional testing of the client and server, to performance, load and security testing. There are tools available in the market, both open-source and commercial, that will help you. So, go ahead and test your web service.

Biography

Himanshu Joshi has 7+ years of experience in software testing that involved comprehensive testing of system, client/server and web applications in both Windows as well as UNIX environments. Himanshu is also involved in setting up Testing CoE, test labs and testing teams that can test middleware applications on UNIX/Tuxedo in SOA environment.

Himanshu is also interested in security architecture, vulnerability analysis (penetration testing) and research into new and innovative security technologies. Himanshu has worked in different industrial sectors such as CRM, security, healthcare, financial services and air/cargo operations involving client/server, middleware and Internet technologies.
Currently Himanshu is working with Dubai Customs.

ISTQB® Certified Tester-Foundation Level

in Paris, in French

Test&Planet: votre avenir au présent.

Venez suivre le Cours « Testeur Certifié ISTQB – Niveau fondamental » à Paris, en français !

Test&Planet, centre de formation agréé DIF, vous propose ce cours D&H en exclusivité.

Apprenez à mieux choisir Quoi, Quand et Comment tester vos applications. Découvrez les meilleurs techniques et pratiques du test logiciel, pour aboutir à une meilleure qualité logicielle à moindre coût.

Possibilités de cours sur mesure.

Pour vous inscrire ou pour plus d'information: www.testplanet.fr

Test&Planet

Diaz Hilterscheid

Masthead

EDITOR
Díaz & Hilterscheid
Unternehmensberatung GmbH
Kurfürstendamm 179
10707 Berlin, Germany

Phone: +49 (0)30 74 76 28-0 Fax: +49 (0)30 74 76 28-99 E-Mail: info@diazhilterscheid.de

Díaz & Hilterscheid is a member of "Verband der Zeitschriftenverleger Berlin-Brandenburg e.V."

EDITORIAL
José Díaz

LAYOUT & DESIGN
Katrin Schülke

WEBSITE
www.testingexperience.com

ARTICLES & AUTHORS
editorial@testingexperience.com

350.000 readers

ADVERTISEMENTS
sales@testingexperience.com

SUBSCRIBE
www.testingexperience.com/subscribe.php

PRICE
online version: free of charge -> www.testingexperience.com
print version: 8,00 € (plus shipping) -> www.testingexperience-shop.com

ISSN 1866-5705

Index of Advertisers

Training with a View

Díaz Hilterscheid

also onsite training worldwide in German, English, Spanish, French at
http://training.diazhilterscheid.com/
training@diazhilterscheid.com

"A casual lecture style by Mr. Lieblang, and dry, incisive comments in-between. My attention was correspondingly high. With this preparation the exam was easy."
Mirko Gossler, T-Systems Multimedia Solutions GmbH

"Thanks for the entertaining introduction to a complex topic and the thorough preparation for the certification. Who would have thought that ravens and cockroaches can be so important in software testing"
Gerlinde Suling, Siemens AG

- subject to modifications -

07.06.10-09.06.10	Certified Tester Foundation Level - Kompaktkurs	Hannover
09.06.10-11.06.10	Certified Professional for Requirements Engineering - Foundation Level	Berlin
14.06.10-18.06.10	Certified Tester Advanced Level - TESTMANAGER	Berlin
21.06.10-24.06.10	Certified Tester Foundation Level	Dresden
28.06.10-29.06.10	Testen für Entwickler	Berlin
06.07.10-09.07.10	Certified Tester Foundation Level	München
12.07.10-16.07.10	Certified Tester Advanced Level - TESTMANAGER	Düsseldorf/Köln
26.07.10-29.07.10	Certified Tester Foundation Level	Berlin
02.08.10-06.08.10	Certified Tester Advanced Level - TEST ANALYST	Berlin
10.08.10-13.08.10	Certified Tester Foundation Level	Frankfurt
16.08.10-21.08.10	Certified Tester Advanced Level - TESTMANAGER	Berlin
23.08.10-24.08.10	Testen für Entwickler	Berlin
25.08.10-27.08.10	Certified Professional for Requirements Engineering - Foundation Level	Berlin
31.08.10-03.09.10	Certified Tester Foundation Level	Barcelona
06.09.10-10.09.10	Certified Tester Advanced Level - TECHNICAL TEST ANALYST	Berlin
13.09.10-16.09.10	Certified Tester Foundation Level	Berlin
20.09.10-24.09.10	Certified Tester Advanced Level - TESTMANAGER	Berlin
04.10.10-07.10.10	Certified Tester Foundation Level	Berlin
11.10.10-13.10.10	Certified Tester Foundation Level - Kompaktkurs	Hannover
11.10.10-15.10.10	Certified Tester Advanced Level - TESTMANAGER	München
18.10.10-19.10.10	Testen für Entwickler	Berlin
19.10.10-21.10.10	Certified Tester Foundation Level - Kompaktkurs	Frankfurt
25.10.10-29.10.10	Certified Tester Advanced Level - TEST ANALYST	Stuttgart
26.10.10-28.10.10	Certified Professional for Requirements Engineering - Foundation Level	Berlin
02.11.10-05.11.10	Certified Tester Foundation Level	München
08.11.10-12.11.10	Certified Tester Advanced Level - TEST ANALYST	Berlin
15.11.10-18.11.10	Certified Tester Foundation Level	Berlin
22.11.10-26.11.10	Certified Tester Advanced Level - TESTMANAGER	Frankfurt
01.12.10-03.12.10	Certified Professional for Requirements Engineering - Foundation Level	Berlin
07.12.10-09.12.10	Certified Tester Foundation Level - Kompaktkurs	Düsseldorf/Köln
13.12.10-17.12.10	Certified Tester Advanced Level - TESTMANAGER	Berlin