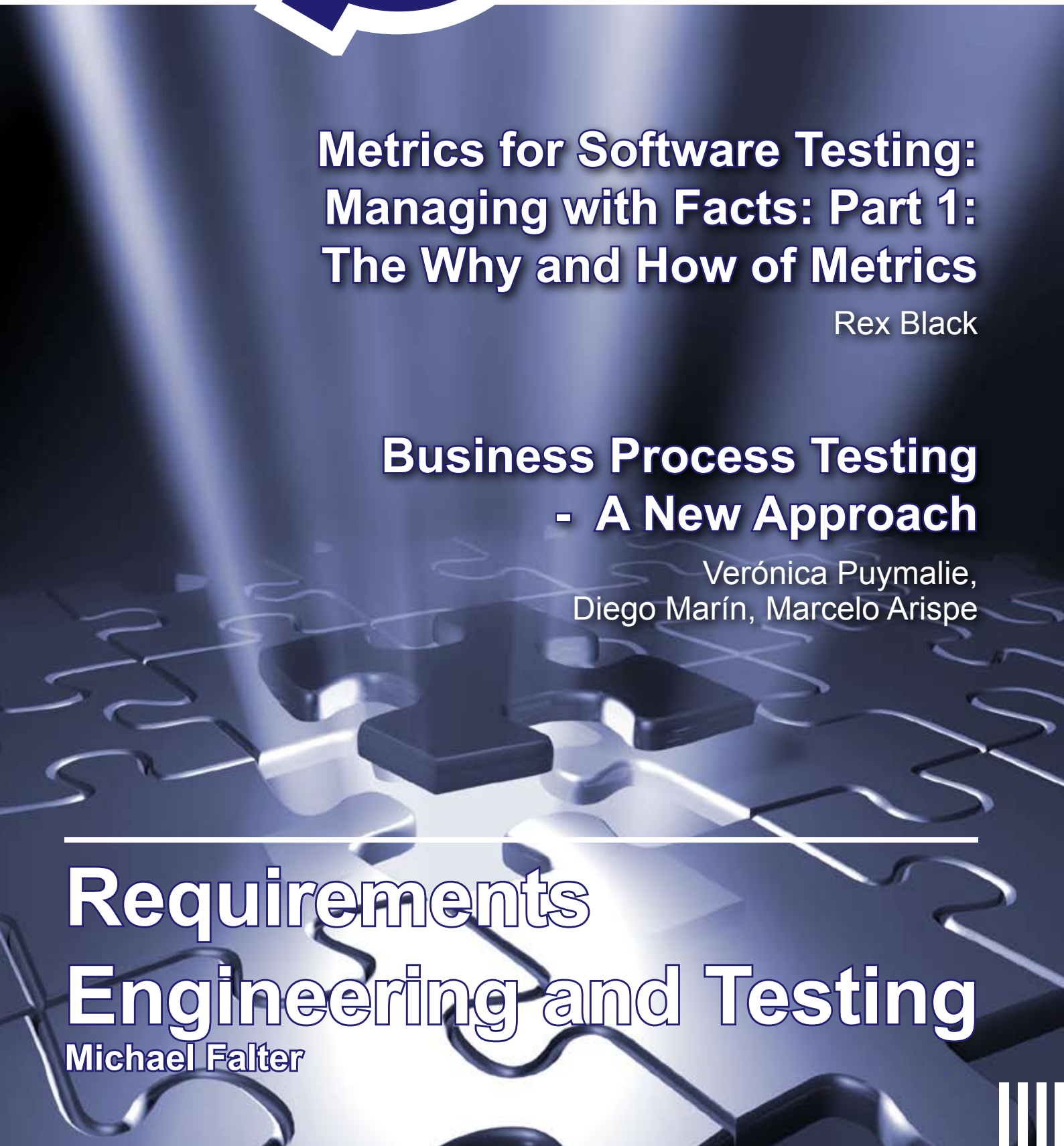
CORE

CORE Nº 2 - January 2011



Metrics for Software Testing: Managing with Facts: Part 1: The Why and How of Metrics

Rex Black

Business Process Testing - A New Approach

Verónica Puymalie, Diego Marín, Marcelo Arispe

Requirements Engineering and Testing

Michael Falter

Global Association for Software Quality

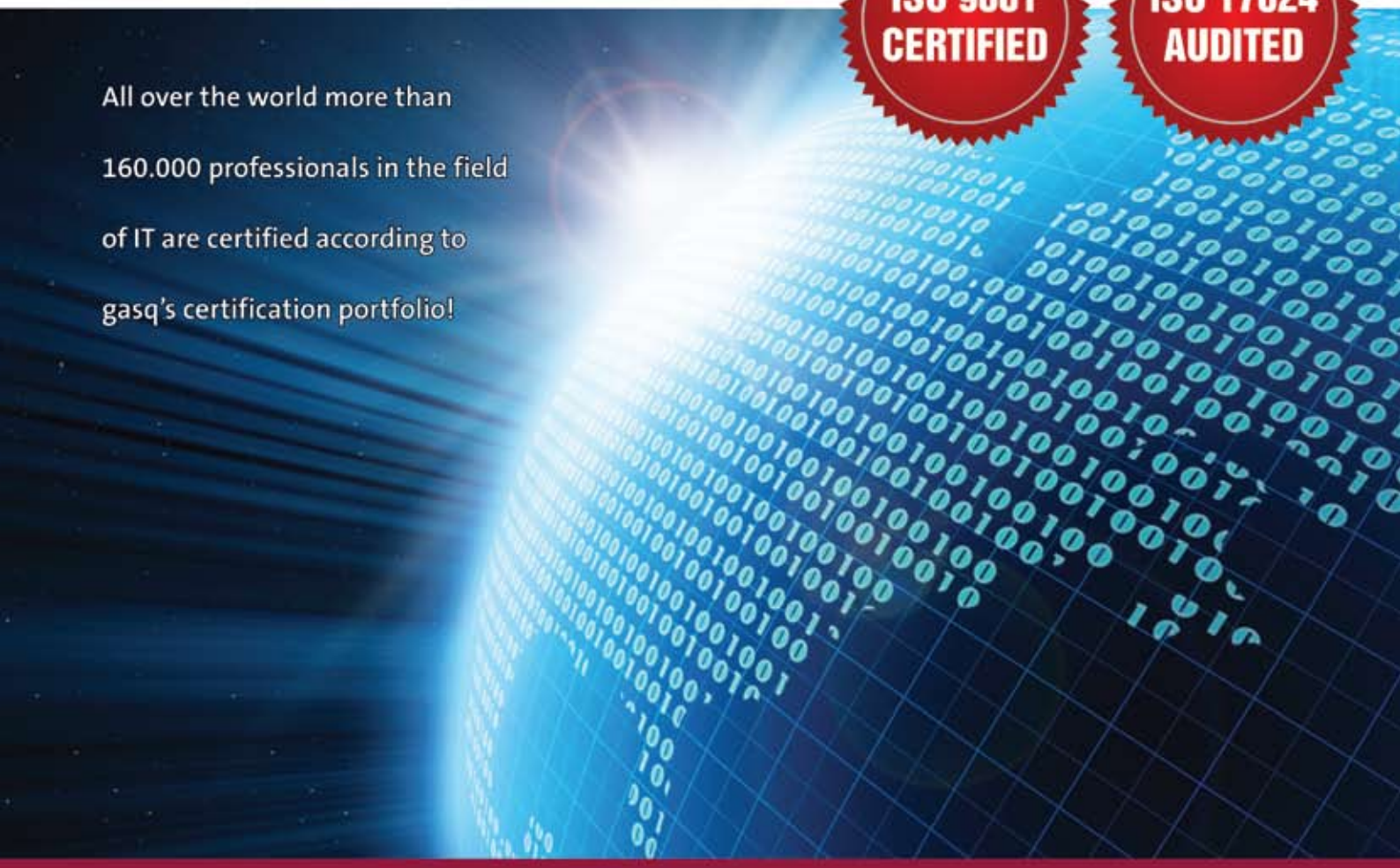
gasq – the global Certification Provider

gasq – the Global Association for Software Quality – is one of the leading Certification Provider for IT-personnel certifications.

ISO 9001 CERTIFIED

ISO 17024 AUDITED

All over the world more than 160.000 professionals in the field of IT are certified according to gasq's certification portfolio!



www.gasq.org

INDEX

Welcome

Management

Quality in Project

Software Engineering

Software Testing

Editor's Note

Quarterly c0re (4 numbers per year) is published by gasq Service GmbH.
www.coremag.eu

Chief editor:
Karolina Zmitrowicz
karolina.zmitrowicz@coremag.eu

Editorial Staff:
Bartłomiej Prędki
bartlomiej.predki@coremag.eu

Cooperation:
Krzysztof Chytła
krzysztof.chytla@gmail.com

Dariusz Paczewski
dariusz.paczewski@coremag.eu

Mailing address:
C0RE Magazine
c/o gasq Service GmbH
Kronacher Straße 41
96052 Bamberg
Germany

Advertisements:
info@coremag.eu

All trade marks published are property of the proper companies.

Copyright:
All papers published are part of the copyright of the respective author or enterprise. It is prohibited to rerelease, copy or modify the contents of this paper without their written agreement.

Persons interested in writing are asked to contact:
editors@coremag.eu

Editorial

New Year - New Opportunities. Following this sentence we would like to introduce the second issue of c0re magazine. In this issue among plenty of interesting articles, you can read about:

• Metrics for Software Testing: Managing with Facts: Part 1: The Why and How of Metrics. First part of the series by Rex Black. Author tries to answer to several questions. What's so great about metrics? How can we use metrics to manage testing? What do metrics tell us about the quality of the product? In this article and three following ones, author will show us some of the answers.

• Requirement Engineering and Testing. By Michael Falter. We all know that many software projects fail. We also know that the final cost of making changes to already deployed and running software can easily be 100 times higher than compared to the cost of changes in the analysis phase. Therefore, solid and systematic requirements engineering can be viewed as an early, integral part of the quality assurance process. In the article, we will even go further…

• Managing relations with vendor. One of the definitions states management as: „sequence of acts of getting people together to accomplish desired goals and objectives and objectives efficiently and effectively". In light of the definition – do we correctly manage relations with our software vendors? Dariusz Paczewski will answer the question.

We would like to thank our partners, contributors and authors – their help, insight and experience helps the magazine to deploy and deliver new and professional publications from all over the world. We appreciate your help!

Enjoy reading!

c0re Team

Helping Chandrashekar live his testing dreams

Author: ***Pradeep Soundararajan***

About

Pradeep Soundararajan
http://testertested.blogspot.com
+91-98451-76817
tweet: testertested
pradeep.srajan@gmail.com

I make attempts to meet a lot of testers so that I could learn from them. One such tester from whom I learnt an important lesson is Chandrashekar.
He was pretty silent and seemed like an introvert during our first meet. He had a story to say about how much difficulty he and his family faced yet he came out well to become a software tester.

Once he became a software tester, he wanted to do it well. That's how we met. He wanted to better and I wanted to help people who better. I thought I must have one of the best person to have demonstrated my passion towards testing till...

Chandru was diagnosed with cancer

It was shocking to me when Sunil called up once and informed me about Chandru being diagnosed for Blood Cancer. That night, I just couldn't sleep. I did sleep late in the night out of tiredness that kicked in for not sleeping.

The next day morning, I got up and immediately felt a hope that Chandru would recover from it because he is a fighter. Called him up and to pep up his confidence, I was shouting, "You are a fighter. You are a gladiator. You are a warrior. This is your biggest test"

After a couple of days when I met him at the hospital, I saw the confidence and fighting spirit of a warrior in him.

Chandru's testing passion

Even despite being treated for the most troublesome disease, Chandru is not giving up on his passion to test and learn more about testing. He is currently reading a testing book and trying to teach his girl friend how to test. He occasionally comes online, not necessarily to check emails, but to check what's latest posts from the bloggers he follows.
That's when I realized I am not one of those who have demonstrated testing passion yet compared to what Chandru is doing.

Lack of funds

Now, this tester, whom we need for the testing community because he is setting a great example has run short of money for treatment. The estimated funds required are INR 20 lakhs for multiple Chemotherapy cycles OR 50 lakhs if bone marrow transplant is to be done. Being the only breadwinner of the family and in hospital, he has no source of income.

Help Chandru

http://helpchandru.com/HowToDonate.aspx
http://helpchandru.com

• (CPAA - Cancer Patient Aid Association)
Cancer Patient Aid Association

Please enter ChandraShekar BN in projects input field.

• Pay Pal

Mail Account : daysofchandru@gmail.com

• You could also do NEFT/RTGS account transfer through Net Banking

Account Details :

Name : ChandraShekar BN

Account Number : 218010015960

Branch : Kormangala, Bangalore

Bank : ING Vysya Bank

IFSC Code: VYSA0002180

• You could also write a cheque and send it to address which would be provided on demand
Do email Sunil or mail to : sunilkumar56@gmail.com and let know about your donation, so that we could acknowledge after receiving your donation.

We thank you in advance for your help and may God bless you for this great help you folks are doing.

Chandru is definitely going to thank the Polish Testing community for the help they are offering and it is going to be the best way to bridge our friendship further.

Managing relations with software vendor

intermediate

Author: **Dariusz Paczewski**

About the author:

Dariusz Paczewski is a manager in e-banking test management team in one of polish banks. Computer scientist by training and a tester by passion. Dealing with software bugs for 5 years. Currently - due to performed duties - working as the interface between business and IT.

With the acquired experience he managed to work out effective methods to combine these two areas. Experienced with international projects.

Responsible for managing a testing team, contacts with vendors and the quality and efficiency of testing processes. Interested in: quality in projects, quality processes, organization, monitoring and improving the test process.

Introduction

One definition of management describes it as sequence of acts of getting people together to accomplish desired goals and objectives and objectives efficiently and effectively. Management comprises: planning, organizing, motivating and controlling (source: Wikipedia). In light of this definition - do we manage relations with our software vendors? Probably most of us does not – managing the test project is, in itself, enough difficult and absorbing. We expect our suppliers to meet the deadlines and assure desired software quality and at the same time we assume that the fulfillment of our expectations should not absorb our resources. This approach often causes problems when the critical moment arrives and it appears that our assumptions were incorrect. Is it possible to avoid the unpleasant surprises? The answer is yes, it is possible and also not so difficult. The only thing that is required of us is the introduction of a simple standard of work and understanding the basic mechanisms of cooperation.

Quality according to the customer. Quality according to the vendor

The basic principle of facilitating, or even enabling, effective collaboration is the awareness that we strive to attain one goal. In software, one of these objectives is a system of good quality. But what does ‘quality’ mean? In this world there are many definitions of this term - changing over time, depending on the context and environment. You can write volumes on this subject. At this point let's focus on the differences in the perception of quality between the customer and the vendor.

Why is there a difference in the evaluation of the software between the recipient, who is not satisfied with the product, and a supplier who claims that the system meets all of the specified requirements? The answer is simple: the software vendor tries to secure himself with the most objective evaluation of the product quality. He can do it by referring to the contracted requirements. This creates a problem on the client side – since the customer could have been unable to fully or clearly define his requirements or made incorrect assumptions about the quality of the vendor’s work. Given the system to be retrieved from the vendor the customer assesses it on the most subjective basis, mostly in terms of the degree of fulfillment of his requirements (including the untold, not clear, ambiguous and understood very differently than the supplier). Subjective assessment performed by the customer results in a series of bug reports that, according to the software vendor, are unjustified - after all the product works according to the specification. Both sides utterly believe that they are right and despite the fact that the time to release inexorably passes most likely this pat situation will not change. How do we resolve this problem? The best way is to avoid it or reduce the potential risk by involving the test team as soon as possible, most likely at the stage of requirements acceptance. Experienced testers will be able to verify the requirements in terms of clarity, consistency and completeness. Clarification of the requirements with testers involved allows to better present the customer expectations to the software vendor. The best way to do that is to use the testers, who have previously worked with this particular software vendor and know what mistakes are to be expected - their effectiveness will be better in comparison to the work of the fresh testers.

The second way to avoid this deadlock situation comes from the awareness that not every found defect is really a defect. Testers should verify the software against given requirements - in situations when the software produces unexpected output yet the behavior was not specified or was specified using ambiguous requirements they should not rise a defect against the software vendor but rather than that they should send the issue to the internal analysis team. In this way we avoid the long queues of uncertain issues and the software vendor will be able to focus on fixing "confirmed" issues, without unnecessary tension between both teams.

Understand your software vendor

What exactly is a company that develops software for us? It is certainly a unique entity with its own values, organizational and project culture. You cannot require that a company providing development services will align with your organization on mentioned matters if they are against their own rules and values. However, you can avoid surprises during the project by getting to know software vendor's approach, methods and values. Note the area particularly important to you - an approach to quality.

The best source of information about the company are its employees. It does not hurt to pay attention to statements about the previous projects, experience, training, overtime. With that data we can create a vision of our co-worker – does he have a trained staff (training, experience), whether the projects are under way in accordance with the plan and shall be concluded successfully (no overtime, rumors from previous projects). You don't have to be James Bond. Use different events, like project kick-off or other integration meetings, to gather information.

Equally important as understanding of the culture of the software vendor is to understand his goals – does he see the implementation of this project as a chance to prove his competence? Does he want to show us his best side? Perhaps it is totally the opposite and his position is so strong that he does not feel the need for the fair discharge of their duties? This could have a direct impact on the quality of the system under test as well as our project.

Know the rules. Make the rules. Play by the rules

To play by the rules you must first know them. The main document which regulates the rights and obligations along with the rules of cooperation between the customer and contractor is the contract. It is recommended to, at least, read the part that concerns us directly. To avoid surprises it's good to know what is expected from us, how much time we have to fulfill our obligations and the results of their negligence. Fortunately, the contract does not serve only the interests of the contractor - it is a two-edged weapon, imposing certain obligations on the contractor in the same way it does for us. People coordinating testing should be aware that they have the right to influence the elements of the agreement relating to their area or influencing it.

For the test coordinator the contract is a good place to define the procedures and criteria for retrieving the software from the vendor, schedule (relatively), penalties, defect categorization, assumed response times, communication channels, accountability, etc...

Reference to the contract is the most neutral and efficient way to claim what one believes should be done and the way it should be done. Both sides of the contract agreed to certain rules of cooperation, and if any of them fails to comply with the conditions of the contract the other one may assert it's rights relying on contract.

Like peer to peer

It is difficult to expect a fruitful cooperation, if one party does not respect or value the other. This principle works well in the production of software - if we fail to prove the vendor that we have competence in the area of testing, we can expect a significant reduction in quality. For now let's skip the internal damage that inexperienced staff member coordinating the tests can do to the project and the software itself and let's focus on the relationship between software vendor and customer. Software supplier quickly senses that "on the other side of the barricade" is a person who cannot ensure or verify the quality of the software. Where can it lead? The lightest of the symptoms can be horrendously long time for the tests on the vendor side. It's "just" a waste of money - after all, someone has to pay for time purportedly spent on verification of the software quality. Then we move to ignoring the basic principles of software development - such as lack of criteria for accepting the system for testing, lack of plan of internal vendor tests (not to mention the results), no known bugs/issues list, etc... This directly and negatively influences the quality of the system to be tested. At the end of such scenario we can expect a total relaxation of procedures on the side of our contractor and a drastic reduction of quality. Unfortunately this is not the end of the possibilities that our incompetence gives to the vendor – the area that probably will suffer the most are the defect and everything that is related with them. Without standards in this critical area, we can expect a long time to repair, rejection of reports, faulty patches and poor regression testing. Our lack of experience puts us at disadvantage when trying to escalate a problem or during any discussion – it's hard for the person who with insufficient competence to challenge the experienced, or just a clever-sounding, contractors.

How to prove your skills to software vendors? If you have them then there should be no problem - just consistently do your work and do not let the vendor to manipulate you in situations where you are confident in your judgment. New coordinators are in a quite good situation if they can gain experience under the guidance of experienced colleagues - the best is to participate in one or two projects as an observer or partially taking over the liability of older coordinators. If you do not feel confident and you do not have who to consult you can rely on the procedures – if there any. The worst case are incompetent people without any assistance who, for some reason, are responsible for custody of the tests. In such situation you can make a good face on the bad play - pretend that you know what you're doing while the learn as quickly and as much as possible from wherever you can gain some knowledge. This is not the recommended strategy - improper use of theoretical knowledge can only worsen the situation. A better solution is to play with open cards and officially transfer the risks associated with inexperience of the person coordinating the tests to the vendor (i.e. vendor should confirm that he will proactively support the inexperienced test staff on the customer side).

Instead of reacting – plan and prevent

Why do not we plan? The reasons are many and all can be a good epitaph on the grave built by the end of the project - "there was no time", "I thought that the project manager should do it", "there was no sense to plan such a minor thing", "it seemed to me that there's still time", "I have these resources", "I did not know that ..." - sounds familiar? If you are lucky, even despite the lack of planning,

you can manage to get your project to the end. Unfortunately, it is more likely that Murphy will be with you all the way and everything goes as bad as possible. How this translates into your working relationship with software vendor? Any failure in relation with the supplier may give a reason for changes in the schedule or excuse for poor quality of the system - the relevant clauses governing these issues are probably in the contract which we are obliged to respect. Well advised: do not go that way.

An important element of planning is to include all of the parties involved in the project. Such an approach is valuable for at least two reasons: first, we ask the supplier about how we can help him and what he awaits from us. In addition to the basic value which is, undoubtly, the time needed to prepare for the expected tasks such an attitude guarantees you a far better position if you have any problems with meeting deadlines by the supplier (you did everything you could to avoid this). Secondly – it's good if we identify and communicate our needs to the vendor as soon as possible. With the agreed tasks, deadlines and people responsible for them, we can better monitor their implementation. This goes the same for your internal resources – having everything planned and agreed gives you a better chance for achieving your goals.

For the professional planning you can use free software available at Gantt Project page - http://www.ganttproject.biz/.

Do not forget about the goal of the project

Please note that the overriding objective for the client and the software vendor should be timely implementation of the system or application. A lot of stressful situation can happen in the course of the project – for some it might be a natural reaction to look for guilty, shift responsibility or escalate the problems at increasingly higher levels. While coordinating the test project you should be aware that all of these actions not only do not bring any added value by actually make it impossible to enable it. As the coordinator you should focus on looking for the effective solution to the problem and restoring the continuity of work and only then, if it will be necessary, on looking why did the problem occur (i.e. to avoid the situation where you have to deal with the same issue again).

You pay – you demand

Preparing to conduct a test project and during its implementation you must be aware that your contractor did not become involved in building the system out of the goodness of his heart. Every working day of the supplier costs money - and it's usually not small. We have every right to verify the progress and effects of work for which we paid. To avoid the awkward situation the appropriate documentation, methods and frequency of inspections should be agreed with the vendor. Remember to agree and address your needs in advance – giving you access to reporting systems or generating reports may take some time.

Vendor's motivation

In previous parts of the article emphasis is placed on the management aspects associated with planning, organizing and controlling the work of your vendor. However, as mentioned at the beginning - the management is also a motivation.
Some of the motivating factors result from previous recommendations: it is far better to work in a controlled and orderly environment in which you can also count on support from the customer. Do not forget, however, that the supplier expects the financial gratification for a job well done. We can provide it in the contract in form of a bonus - by making the size of the premium depending on the quality of the delivered software. Using this method, remember that the same agreement may also impose penalties on the supplier if the delivered software fails to meet certain criteria.

As a method to integrate teams and improve relationships with the vendor we can use common social events. By far it is the most popular and effective method. Let us note, though, that the source of the idea and funding should not come from only one side. Ideally if it would be a joint initiative of both parties.

Summary

Despite the extensive range of topics on the agenda (from cross-company team-building, through competence and planning through to contract negotiation), the author still maintains the theory that the vendor relationship management is not difficult. Note that this whole process is easier and more natural if more and more organizations implement these simple fundamental rules.

The second thing that should draw your attention is the involvement of the test coordinator in the areas which don't seem to be directly related to his or her responsibility. Negotiation of contract, active involvement in the review of the requirements, the initiation of team-building events - you can say that this is not the responsibility of the coordinator but more of the project manager. Please note the fact that all these activities have an impact on YOUR work in the later stages of the project. The author recommends the most reasonable and pro active attitude you can think of. Remember the old polish saying –the sleep you get at night depends on how you made your bed before.

Last but not least – the things mentioned here are not the most sophisticated and only things that can be used in your benefit. These are the most simple and easy to introduce guidelines – which doesn't mean that they are not effective. Author encourages you to look after your own methods to improve the quality of work with your software vendor and – in result – the quality of the systems you test.

Good luck with the forthcoming projects, testing and effective vendor relationship management! ■

Traceability at the heart of the software development lifecycle: a powerful tool to increase the quality

intermediate

*Author: : **Eric RIOU du COSQUER** Co-author: **Krzysztof Chytła***

About the author:

Eric RIOU du COSQUER, Treasurer of the CFTL (Comité Français des Tests Logiciels), Test Manager at Orange-France Télécom. Eric is responsible for a team dedicated to Requirements and Tests Management at Orange, a French telecommunications company renown worldwide. In his current position, he is in charge of defining and implementing the processes defining Requirements and Tests management as well as of selecting and supporting the associated tools. His daily activities also consist of providing internal training courses and supporting the software projects of the Information Technology division of Orange.

Holder of the Foundation Level, Advanced Level Test Manager and Advanced Level Test Analyst certificates issued by the International Software Testing Qualifications Board.

About the Co-author

Krzysztof Chytła
Began his software testing adventure back in 2001 as a beta tester and went pro in 2007 after graduating from Wroclaw University of Technology. He was initially involved with mass-market mobile applications, then moved to J2EE systems such as customer care portal for one of the mobile carriers. Eventually switched to embedded system in the area of Telecommunications where he currently excels.

Main responsibilities: software testing and integration, planning, requirements analysis, test automation, coaching and remote support, test process improvement and documentation.

Holder of ISTQB Advanced Technical Test Analyst Certificate.

Introduction

The term «traceability» is an indispensable phrase for every software development glossary. It is widely used every day or nearly every day. It reflects a principal good practice upon which everyone agrees: “It is necessary to set up traceability links between different items being manipulated, starting with business needs, through functional requirements, towards the code”. The goal is to ensure – and to be able to prove - that what had been initially requested is delivered, verified, validated and easily maintainable.

But understanding and implementing this concept are often complex and it might be difficult to identify the desired traceability level in a specific context, to implement and to measure it.
This article will provide you with ideas on this matter in order to use it in an easy way, in the context of your software development project.

About traceability

Many definitions exist for the term «traceability», not only in the software development area. The formal one is a follows: “Traceability is the ability to chronologically interrelate uniquely identifiable entities in a way that is verifiable. Traceability is the ability to verify the history, location, or application of an item by means of documented recorded identification”[1].

Definition from the food industry is highly

interesting and might serve as a good example: “Traceability = ability to find, for a specific project, evidences for all the steps of its creation and to identify the origin of each one of its components. Product’s traceability allows, for instance, to identify the suppliers of raw material, the different places where the product was stored, the operations and equipment used in its manufacturing” [2], [3].

Sounds familiar, doesn’t it? If we replace “raw material” with “components” and “stored” with “developed and tested”, we are not that far from the meaning given to traceability in the software development world!

In CMMI-development, for instance, traceability is defined as: “a discernible association among two or more logical entities such as requirements, system elements, verifications, or tasks“ [4].

In fact, within each company, in each project, it is necessary to think about the meaning behind the word «traceability» before thinking about its implementation. It is advisable to have a clear understanding of the chosen definition and stick to it. Remeber that it works in both directions: top-down and bottom-up or forwards and downwards if you like.
Traceability can be divided depending on the point of view. Vertical traceability describes the interdependencies among the parts of a single work product or discipline (e.g. requirement - requirement). Horizontal traceability addresses the relationship of the components across collections of work products (e.g. design component – code component)

Items affected by the traceability

Many items can be traced between one another, especially if we cover the full software development lifecycle, from business needs elicitation to failures discovered after the product roll out. For example:

- Needs formulated by a customer
- Business requirements
- Product requirements
- Technical architecture elements
- Functional architecture elements
- Written code
- Tests cases
- Test sets
- Test environment components
- Test data
- Test results
- Incidents
- Defects
- ...

This list is far from being exhaustive. It can be modified or extended according to the context of one’s project.

Most useful interrelation s

Here, once again, the answer will vary depending on companies and projects. The most important thing is to choose the relevant subset that covers all the necessary relations. Keep it concise to avoid misuse or misunderstandings. The goal of traceability is providing useful information and saving time. Nevertheless creating traceability links does have a cost, a cost that pays off. Let’s keep that in mind. This will allow efficient communication and will be a helpful argument in convince the main stakeholders to approve the initial effort to set up traceability.

Apart from being in a particular context, such as Safety Critical Systems, it is not necessary to implement all the possible links but only the ones with an added value for the project. Let us try to identify the most useful links from the paragraph above. These should be mandatory in most cases.

customer needs – business requirements
On one hand, it will ensure that a need, which is often blurred or insufficiently described by small pieces of information or in many different ways (e.g., emails, documents, pictures…) is covered by one or more business requirements, clearly identified.

On the other hand, during business requirements reviews, it will allow finding out the source of the requirement and where to find additional information if necessary.

business requirement – product requirement
This vertical traceability link shows how the product should be developed having taken into account each business requirement. It ensures that no business requirement is forgotten and offers the possibility to identify the source of each product requirement.

A business requirement may impact several parts of a product and this link may be useful for analysing the impact of business changes on the design and implementation of the product.

product requirement – architecture element
In order to know in which hardware or software item of a product a requirement is to be implemented, it is necessary to create a link that will ensure that a requirement has been really implemented in the correct product. The link between human effort, ideas and the software that implements is brought to life this way.

product requirement – code
How will the modification of a requirement impact the code? Which product requirement will be affected by a defect found in a specific part of the code? How much code is needed to implement the requirement? This link is necessary to answer the above questions. It may be a direct link or an indirect one, for example, through architecture elements.

code – test case
This link, which is often managed within the development environment, is very important not only from the code coverage point of view but also does it help to quickly identify the regression test cases that should be executed again after a code modification. Moreover it makes automation of regression tests easier. This link should be considered at the component level.

product requirement – test case
Firstly, this link can and should be created early in the software lifecycle, it will allow verifying that a test (at least one) has been prepared for the associated requirement and, from the test’s point of view, to know what is actually being tested by each test.

Secondly, when the tests execution is started, the link will allow mapping of the execution results to the requirements providing the requirements coverage information.

Test-specific links: test case – test instance – test step - test environment – test data – test result – test log - test report
If we consider that a test case is a

scenario made up of steps, then it's instance can be executed in different environments, with different test data and of course with different test results at the end. Test steps can be generic (especially if automated testing takes place either keyword-driven or script based) and reused upon need.

We may talk about test instance as of the representation of the test associated to an environment and to test data. A test instance executed several times may have different results.

To be precise with this kind of traceability and documentation, it is highly recommended to follow the IEEE 829 test documentation standard.

test result – defect

Thanks to this link we can ensure that each incident observed during test execution has led to the creation of a defect report.

Used from a defect's point of view, this link is very useful for at least two reasons. First, the developer in charge of correcting the fault will easily find the associated test cases and be able to execute it again in order to reproduce the incident and understand the defect. Secondly, once the defect has been corrected, the tester in charge of verifying the correction could also use the link to find the associated test and verify that it has passed successfully!

Each link is useful by itself but linking them together brings oneself far greater benefits. The expected return of investment can increase as much as tenfold if compare to single links or none!

For example, if you have full traceability, that is: « Business Requirement – Product Requirement – Test – Test Instance – Test results – Defects » you will be able to know the status of the requirements coverage anytime and for each requirement you will easily answer the following question: "Has it been tested?", "What was the result?", if the result was negative: "What is the associated defect?" and so on.

Setting up the useful links and measuring traceability

Once the useful links have been identified, it is necessary to think about the way of implementing them. All the phases of the software development lifecycle are affected, as well as many items, many actors and different tools (e.g., requirement management, test management and execution, modelling, development, defects management, et cetera).

Each link may be created with or without a tool, manually or automatically. Considering the size of nowadays projects it would be virtually impossible to handle the traceability manually. Hence it is supported by number of tools both commercial and open source. The most popular ones are: HP Quality Centre, IBM Lotus Notes, Bugzilla, Mantis, Jira, Testlink. They have different interfaces but a common goal: making traceability clear and efficient.

Unfortunately there is no tool which offers the possibility to create and manage all the links. As a result of that we often have to develop bridges or interfaces between tools. The most important thing is to clearly explain and document the way of creating the links. Selection of the right tool that fits best to the project is a real challenge and should not be treated lightly. Serious consideration and wide consultation should be undertaken involving people from different parts of the project – potential end users. The role of trainings in improving people's skills cannot be deprecated.

Finally, we have to consider the measures that can be applied under specific conditions. Let us consider it in two ways:

1. The measure of the traceability itself : the objective is to measure the percentage of implemented links broken down by the total number of possible links (ideally 100%) and to make the appropriated decisions.

Do we have 100% of the requirements associated with at least one test case? Is every item or piece of software associated to a requirement or to an item of the software design? If not reasons should be given or omissions corrected. Visibility of problems is of the top importance.

2. Traceability allows better and more thorough measures. For example: "How many defects do we have in different parts of the code?", "Which requirements have been fully tested?",

"How many tests will have to be updated and re-executed in case a requirement is modified?

It may be difficult to define and implement the traceability. It has to be done project by project, depending on the objectives and needs regarding quality. Cleverly used, traceability does become a powerful way of improving software quality and satisfying the customers. Try it and see how it makes work easier and more effective. If traceability is correctly implemented no information gets lost or is missing when most wanted. All that gathered together positively affects quality of the developed product. Thanks to traceability the quality of the product can be better described and presented to interested stakeholders.

References:

[1]. "Glossary," ASME Boiler and Pressure Vessel Code, Section III, Article NCA-9000

---

[2] http://www.tracefood.org/index.php/Fundamentals:Traceability_definition

---

[3] http://www.azaquar.com/en/qsa/index.php?cible=trace_tracabilite

---

[4]Practical insight into CMMI by Tim Kasse, Artech House Publishers; 2 edition, August 2008 ■

Quality Matters. Though Expenses May Mean Even More

basic

Author: ***Stanislav Ogryzkov***

About the author:

30 years old. Specialist in enterprise-wide information systems, business process re-engineering (BPR), quality management (including testing). ISTQB Certifed Tester, Foundation Level, and a certified internal auditor of quality management systems (ISO 9000).

Graduate of Vladimir State University, Russia. MSc major in computer science, PhD major in technical science. One of the two first ISTQB Certifed Tester in Russia.

Since 2004: Quality Assurance Person at Inreco LAN (inrecolan.com), an offshore software development outsourcing company located in Vladimir, Russia. Since 2005: Quality Assurance Manager at the same company. Since 2006: Business Process Improvement Manager at the company. At last, since 2010, Chief Information Officer (CIO) at Inreco LAN.

See http://stanislaw.ru/eng/author/resume.asp for details.

For a few weeks I have been reading a very interesting book by Andrey Parshev, entitled “Why Russia is not America”. It is mostly an economic work, though a few political aspects could not be avoided in it. The book is quite arguable, from the economic point of view. However, there is an interesting statement about quality, which I would like to discuss below.

The author of the book, Mr. Parshev, states, “Quality and competitive ability are absolutely different things”. At first sight the statement looks strange and even contrary towards the modern understanding of that “quality rules”. :-) However, a little bit below in his text the author shows that goods of moderately worse quality may be sold better than goods of better quality. So, if quality is not the (main) criterion of competitive ability, then what? The author deduces that the main criterion of competitive ability is expenses (as compared to earnings). The more positive difference between your earnings and your expenses, the more competitive you are.

Can this all be true? Yes, look at China that produces most of the goods delivered all over the world. So was that buzz about “quality matters” just a useless buzz, “blah-blah-blah”?.. No, look at the same China – their goods became better and better, otherwise their markets may stop growing because in some industries, places and among some people there are different priorities for quality and prices. Another example: Russian software developers in some cases are preferred in the world for their generally better quality despite of their higher rates (as compared to other outsourcers), So what we can learn from that arguable statement on “quality vs. expenses”?..

In fact, nothing new. Do you remember the so-called “iron tetrahedron” (pyramid) in software development? Let’s look at the picture once again:

(Sorry for the “Quality” written backwards – I am sure you have got that humor.:-) So we have “Functionality” of some “Quality” created with available “Resources” in specified “Time”. Here “Resources” include financial resources, i. e. our expenses. So if you would like to increase you competitive ability by decreasing your expenses, you will definitely “automatically” increase the time you need to complete, or (more likely) decrease the quantity and/or quality of the functionality. The problem of finding the optimal balance in the “iron tetrahedron” still remains unresolved in its general case.

The paradox of software development is, if you spend more resources on software testing you get software of better quality

that increases your competitive ability anyway. Meanwhile, in this case you also increase your expenses thus decrease your competitive ability. How may the paradox be understood?

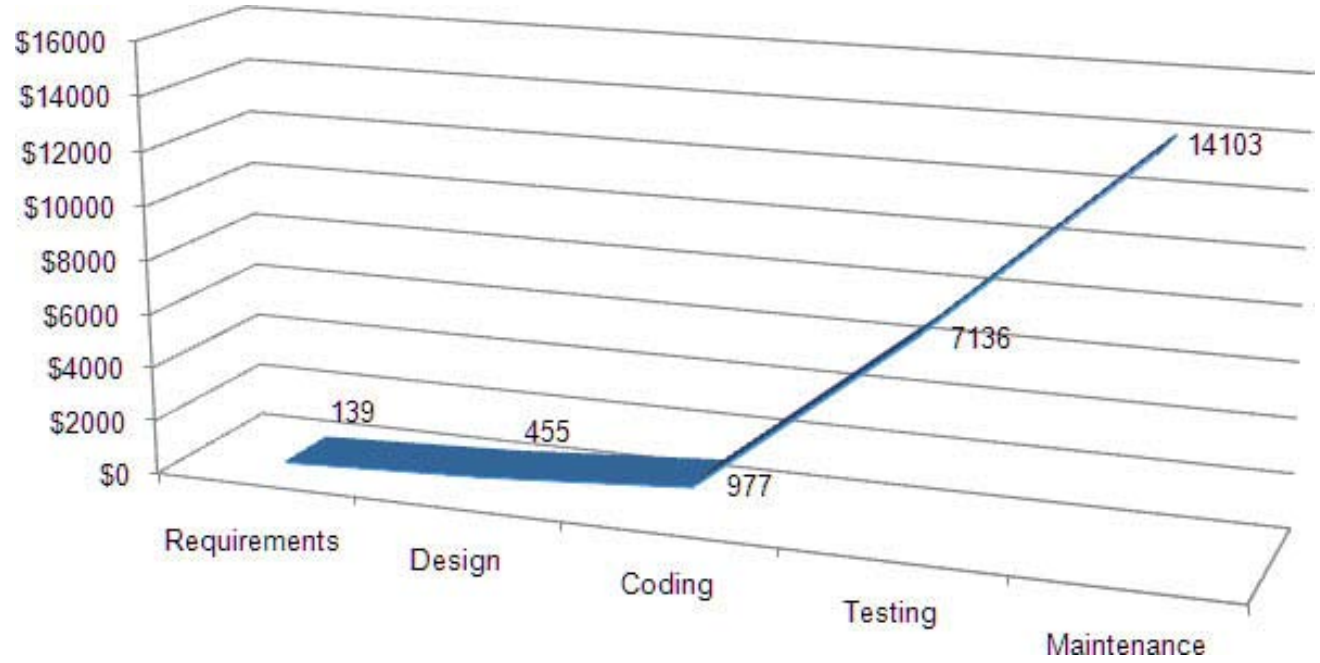
Let's recall that software testing is about finding defects in software. :-) And there is a so-called "cost of defect" (or, more precisely, "cost of fixing software defects") depending on when a defect is found and by whom. For example, the cost of a defect found by a professional tester during earlier phases of a project is less than the cost of the same defect found in the project later, and so on. The highest cost is the cost of the same defect found by a real client in the publicly released software.

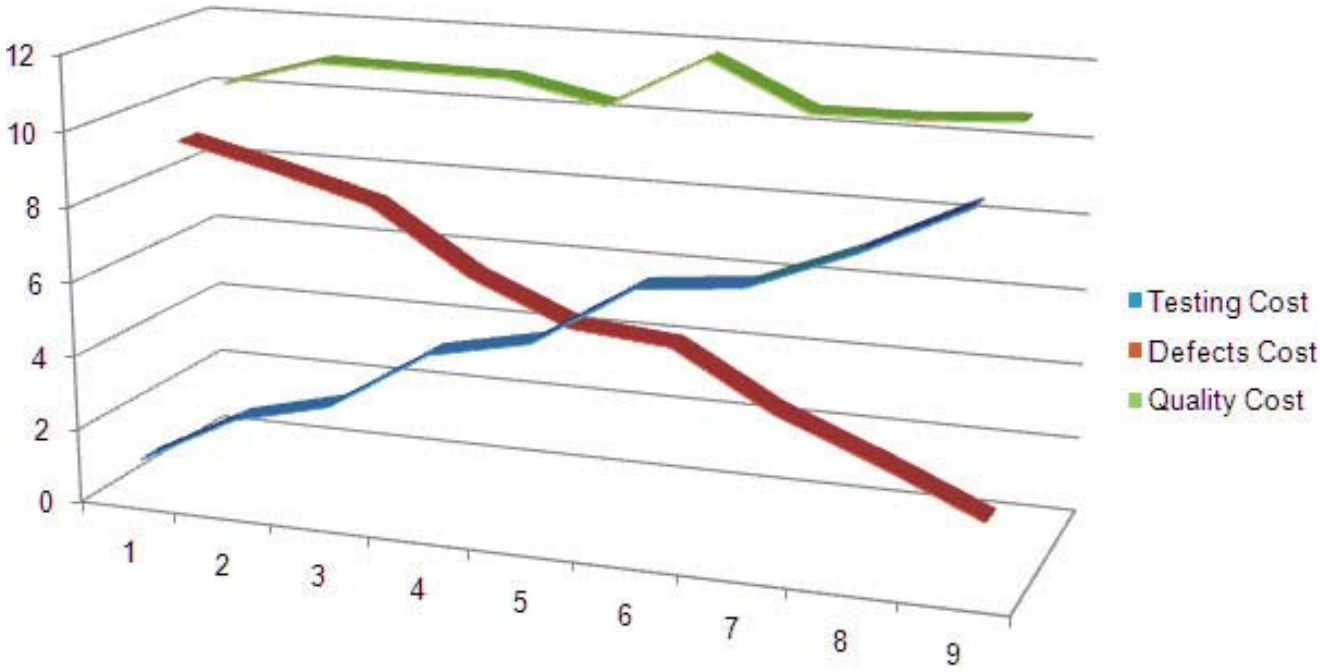
So if software testing increases expenses then it does so in a non-linear manner. Moreover, we may suppose that our testing costs in the general case are compensated by less defects costs, and the total quality cost level may be quite stable:

And if the total quality cost level is stable then software testing does not increase our expenses thus does not decrease our competitive ability. Moreover, software testing still increases our competitive ability by improving the quality of our software.

So you may relax – quality still matters. :-)

P.S. A few more words on "the lowest competitive ability of the Russian economy" (due to its severe climatic conditions) also stated by Mr. Parshev. The first contradicting example was partially given above, I mean, Russian software developers. The total and wide spreading of professional software testing in Russia may even boost that example (that is why I became a member of RSTQB/ISTQB). The second example is… Russian space rockets and ships that are not cheap but are much more reliable than any others. :-) ■





BQI
BEST QUALITY INSTITUTE

QUALITY DESERVES RECOGNITION

2010
AWARD WINNER
AGILE LEADERSHIP
BQI
BEST QUALITY INSTITUTE

■ ABOUT BQI

Best Quality Institute (BQI), based in Berlin and Munich, is the leading award institute, assessing the quality of companies and employees.

BQI is the first place to call for development of research studies and assessment models for diverse areas of your business. BQI is a pioneer in standardizing quality assessment of software and personnel. Independence, comparability and transparency is BQI's mission statement, allowing BQI to live up to the high expectations of its clients and participating companies.

■ AGILE LEADERSHIP AWARD

Agile methods become increasingly important in software development projects.

Agile development will soon be widely accepted by users, as its benefits are recognized by a growing number of individuals. Winning one of BQIs Awards demonstrates your competence and experience using "Agile".

Register and get a study for FREE
www.bqi.eu
Enter voucher code
BQI-2010-GASQ-1377

BQI Best Quality Institute GmbH
München / Berlin

Otto-Hahn-Straße 13b
85521 Hohenbrunn / Germany

Tel: +49 (0) 30-609 891 710
www.bqi.eu

Hudson as an example of BuildServer tool

intermediate

Author: ***Jakub Kubryński***

About the author:

Habitually, Jakub is engaged in designing, implementing and optimizing the software processes and quality management. His functions include both the development of procedures and toolkits to support these tasks. He is also associated with the company Furba implementing projects in the field of consultancy and quality assurance in IT.

Contact the author:
j.kubrynski@furba.eu

Continuous Integration in a nutshell

As most readers may know, Continuous Integration appeared for the very first time in the article "Continuous Integration" by Martin Fowler, which is considered today as the father of the methodology. In short, he defines CI as follows:
" Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly. "

The life cycle of an artifact

The above definition of Continuous Integration entails a very important message: we seek to accelerate the production of better software. While full implementation of the CI may be a relatively serious challenge so far to eliminate the risky part of the already functioning in the manufacturing process may be carried out fairly painlessly. Let's stages of software development (deliberately skipping issues and design requirements gathering as irrelevant to the contents of the article):

Development
Compilation
Testing and quality analysis
Installation

While the first and last element again, I will cleverly omitted, however the two central devote virtually all of the current instance. Will try to explain that their properties spend awake at night and what the IT staff we can fix this by optimizing their processes and using tools such as build server that such action very well supported. But first a few words of introduction to the topic mentioned in the preceding sentence tool.

Features of a build server

There is a certain set of characteristics that can describe the correct software to act as the build server. Let's distinguish main construction phase of a system to determine such characteristics. Absolutely essential to this process is the source code - without it do not have anything to compile. Our server should then be able to download the source code. Therefore, integration with source code management system is the first feature of our server.

The next step is to run the compilation process. This step, depending on the technology used can have different

shapes, so since we are interested in software used in our team will require at least the currently used tools. This gives us a characteristic number two.

After getting output from compiler we should check its accuracy. Most obvious in this phase is to carry out the tests. Therefore, the ability to runt tests it another thing I expect from the tool. In addition, it is good to run analysis of the source code quality. So support for the instruments of Quality Assurance is the next desired functionality.

We still have to answer the question: when does all this have to happen? Now that these steps are not performed in isolation from the actual manufacturing process, it would be able to determine (the more detail the better) moments in time and events to launch a decisive share of building.

The above basic set of properties can of course be extended - often dependent on the scale of deployment - requirement such as providing authentication, intuitive handling, storage history, automated deployment etc., which determine the choice of a particular product from the market (ie. TeamCity, CruiseControl, Bamboo and described further below, Hudson).

As an example I will show use cases, best practices and benefits arising from the use of so-called build server. I will use Hudson server for that.

Who is that Hudson

What is the best in using the above tools in daily work? For me, Hudson is an additional person working in a team, performing many repetitive and very basic things. Tasks such as documentation generation, test preparation, production of artifacts, deployment, run tests, etc. influence the creative tasks of programmers. That is the moment when Hudson shows his benefits. He performs duties regularly, promptly and deterministically. With the extended configuration system, we can thoroughly customize the software to our needs. But what we have to do to enable the work efficiency of nour new colleague?

Working with a new friend

The hero of this article part is automation.

The basic requirement is Hudson is automated build. Fortunately, we come here with numerous plug-ins, allowing usage the most of build tools (such as Make, Maven, Ant, Msbuild, etc.). In the case of less popular technologies (such as PowerBuilder, Delphi, etc..) we can also use shell scripts, which with more or less effort, can automate almost everything.

In addition, we must remember that Hudson's duties are limited to the following:

checkout source code from the repository, launch the build process, running "post-build" actions

The above range of possible actions obliges us to use the so-called principles of a clean build (by the way, regardless of whether we use Hudson or not we should implement that). In another nomenclature, we can also meet the definition of CRISP build, which is comprised of the following characteristics:

Complete - all that is required for the compilation (except heavy elements such as application servers, etc) is in the repository. This applies mainly to libraries and files needed to properly create a compilation (of course, in the case of technologies that manage their own relationships - such as Maven - to comply with this requirement will need to take care in a separate repository for artifacts)

repeatable - in other words, the deterministic - gives the same results every time you start

Informative - carrying information about

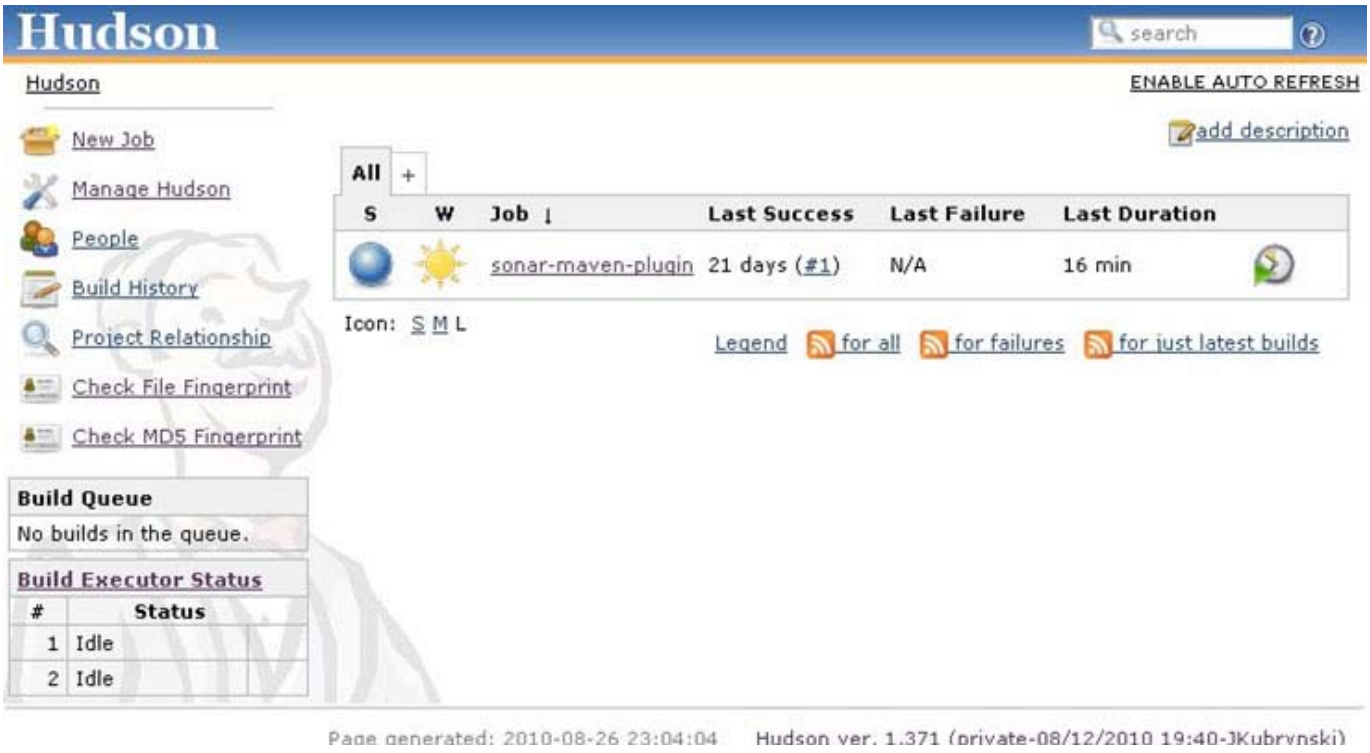


Figure 1 - The main Hudson page

their "health" - if the compilation failed, we know which one class of errors, if the unit tests ended without success, we know that, and where the code failed, etc.

Schedulable - giving a run at the right time with the triggers or timing.

Portable - that is such a programmer to emerging allegations need not correspond to the famous "works for me "

Once we have prepared compiled artifact , the next step is the automatic deployment. This ensures that both the tightness of the manufacturing process, and deployment acceleration of subsequent versions of the software. It also enables a smooth launch of test environments, which opens up further opportunities with Hudson.

As we know, our new colleague is doing well in running the test software entrusted to him. But here again we must refer to the word of the day (automation). Tests, like the build and deployment process, must be automatic. Fortunately, at the current level of IT solutions is not limited us to verify only the simplest elements of the application. With the help of the Hudson, we can run both unit tests (xUnit), functional (eg Selenium), performance and integration.

Build optimization

We come to the stage where we should consider whether a full build of the system (including compilation, deployment, testing and often a static analysis of source code) has to be not too protracted. How long does the programmer who made the change a few lines of code will want to wait for effects? For small applications, where the build takes a few minutes, we do not have to dwell on the theme builds differentiation. At a time, when the full process is extended to several hours, we can not do already without the introduction of some improvements, such as segregation of individual tasks between different types of builds. In practice, this amounts to a de facto set up several jobs for such a system, for example, organized as follows:

developement - runs automatically when it detects changes in the source code repository, consisting of only compile and run unit tests,

integration - caused for instance twice a day, to verify compliance of the current version of our application with related systems,

night - as the name suggests, usually run at night and covering all elements of the process - here a few hours is not a problem, because it even the most hard-working team has relax during the night…

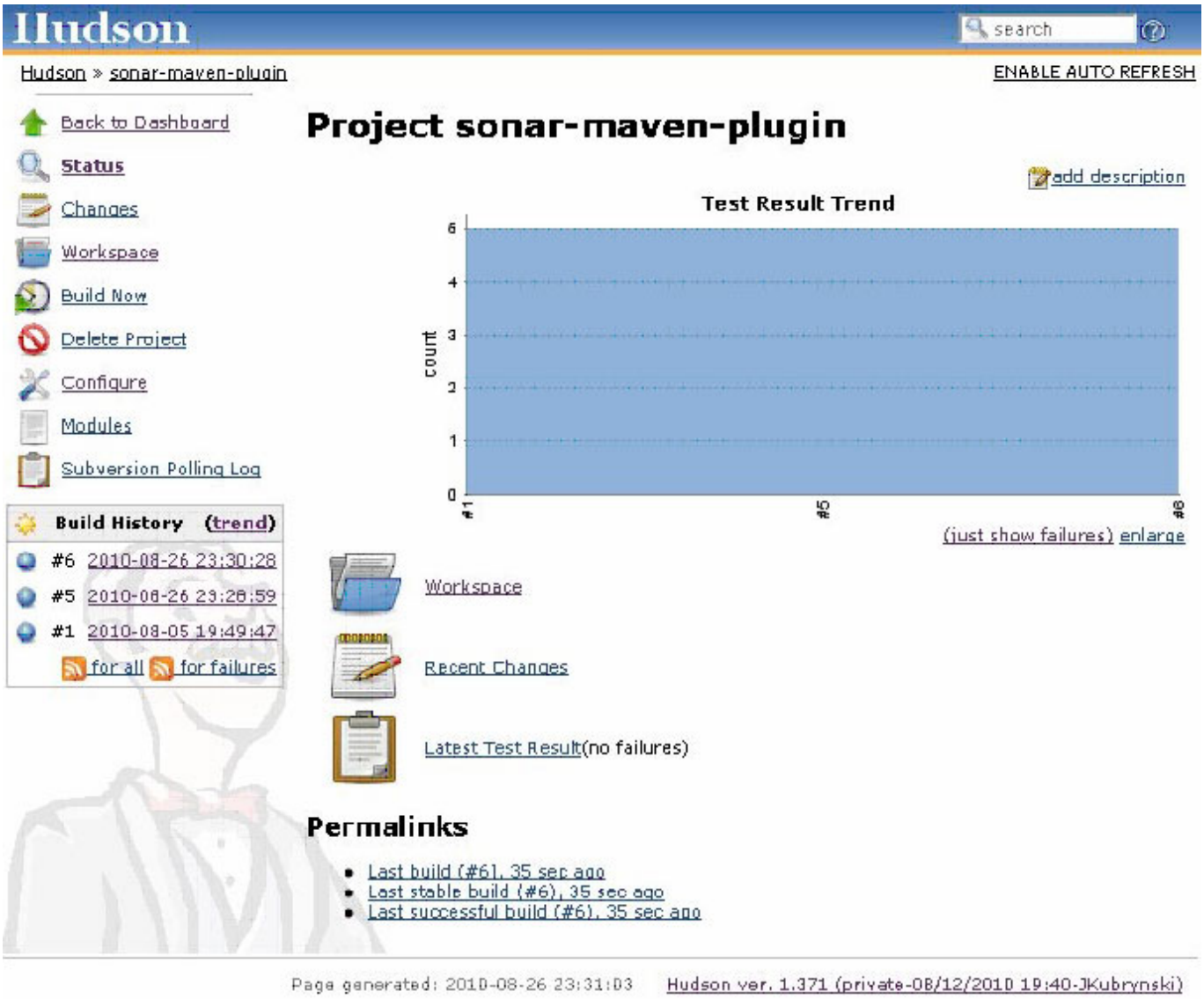


Figure 2 - Design View

Further advantages

What are the additional pros from the implementation of Hudson? In addition to those mentioned before is to serve as guardian of the team. Good configuration actions to be performed at various build stages make it possible for the immediate notification to the developer compiled the code revision to the erroneous execution of unit tests or lack of appropriate versions of libraries.

In addition, using the build server tools in manufacturing process, we deprive ourselves of the problem often found under the title "compiles for me. " In the longer term, this approach also leads to the decentralization of knowledge on how to build, test and deploy systems. It should also be aware that we can utilize the Hudson, not only to classical building systems, but also to perform many administrative tasks and maintenance, not necessarily related to the compilation environment.

We grow in strength

The next stage of initiation into the use of the Hudson is the use of extensions (otherwise known as plug-ins). We have the choice of over 300 (!) ready to run plug-ins grouped in functional terms, such as code management, builds triggers, build tools, etc.. As a result, they cover most of the activities and the challenges confronted with daily work. But the situation a is little more complicated in the case of ongoing deployments in large companies, where a lightweight (agile style) approach of Hudson collides with severe procedures - such as the necessity of providing multiple levels of permissions, auditing, etc. As a rule, it is combined with the necessity to extend set of default plug-ins with out own. Fortunately, there comes some help with very friendly API, enabling the transformation of the Hudson in a very powerful tool. After the appropriate coding and configuration we get a tool fully prepared to work and integration with external systems. As a confirmation, please notice that the biggest ever made by me to implementation, and also one of the largest in the world), is close to 200 applications built for more than 70 servers.

Quick Start to Hudson

Download a WAR file from the address http://hudson-ci.org/latest/hudson.war

1. Set the environment variable pointing to HUDSON_HOME directory in which you want Hudson files

---

2. Run the application using the command java -jar hudson.war

---

3. At http://localhost:8080 we have the main dashboard of Hudson

---

4. Click "Manage the Hudson" and "Configure System" in order to install tools (JDK, Maven2)

a. in bo Maven "Install automatically" select the version 2.0.7 and enter the name (eg, Maven 2.0.7)

b. in box JDK "Install automatically" choose 6 Update 21 and enter the name (eg JDK 1.6_21)

c. At the bottom of the page click "Save"

---

5. To add a new task click "New Job "

a. Enter the name of the task (eg, sonar-maven-plugin)

b. Select the "Build a Maven2 project"

c. Click "OK"

---

6. In the configuration window of job set:

a. The path to the project in SVN (Source Code Management => Subversion): http://svn.codehaus.org/mojo/trunk/mojo/sonar-maven-plugin/

b. Build Triggers - select Pool SCM and write "* * / 1 * * *" (which will pool SVN every hour to search a new version of the code)

c. Build:
i. Root POM: pom.xml
ii. Goals and options: clean install

d. Click "Save"

---

7. We have been transferred to the Design view, where we choose the "Build Now" in order to create build.

---

8. The process of building, we can peek keep going into the current process (shown in box Build History) and then selecting "Console Output"

---

9. Without any additional settings you can get the result of compiling when reviewing project workspace.

Summary

The everyday growing popularity (even certified by the amount of downloads) confirms the validity of this way to optimize the development process. Similar conclusions can be drawn also from my past experience. Costs related to the implementation of both the server and adaptation of compilation of our applications to the general requirements are much less than those incomes obtained in return of the level of security, which includes the already mentioned rapid compilation error detection, failure in the course of testing, and storage of information in an accessible way to how to build. In the next article will present further employee that will support our efforts, the grateful on behalf of the Sonar ... ■

Requirements Engineering and Testing
Two Sides of the same Coin

intermediate

Author: ***Michael Falter***

About the author:

Michael Falter is a Vice President and Head of the Solution Line PQM at PENTASYS AG, a software company in Munich, Germany. PQM stands for "Project- and Quality Management". Michael is looking back at more than 20 years of experience in the IT industry. He holds a degree in Computer Science of the University of Augsburg in Germany. Important stations in his professional career were debis and T-Systems. Michael is a PMI certified Project Management Professional. Michael works for PENTASYS AG since 2001, where he focused on project management and quality assurance.

Many software projects fail, that's a fact. The actual percentage varies, depending on how you define "fail". Even a project, officially finished successfully, may in reality be a failure, because at the end, the software did not really meet the requirements of the users. The final cost of making changes to already deployed and running software can easily be 100 times higher than compared to the cost of changes in the analysis phase. Therefore, solid and systematic requirements engineering can be viewed as an early, integral part of the quality assurance process. In this article, we will even go further. We will claim, that iterative requirements engineering, embedded in an "agile project", will produce better and more cost effective software than "classical" approaches.

Meet the requirements

What makes you sure, the software you are developing meets the requirements of your end users? You might say: "I am testing the software. If I finally find no bugs; the software does what it is supposed to do." Right or wrong? Maybe wrong! Your software might work without blocking or crashing or eating up all of the memory. Still, your software is probably not doing what your end user EXPECTED. So, the critical question is: how do we assure, that our software does what is expected?

How do we find out what's expected?

Well, you already guessed: good requirements engineering! But what is "good requirements engineering"? How do we achieve this? The answer is quite simple: requirements engineering and testing have to be in one process. They have to live in a very close relationship. And this is the moment when we discover, that "classical" software development process models like waterfall or the very much formalized V-Model XT all have a problem here. Requirements analysis and testing are separate phases there, often with many months of time in between. But "times are changing" and people, namely end users are changing; therefore requirements are changing. So, in the end, the testers are testing features which might be already irrelevant to the end user.

Focus, focus, focus

What exactly is relevant to the end users? This leads us to the first important step for the requirements engineer. Identify the relevant stake holders, the real decision makers and process owners. Be careful not to waste your time with the "wrong" people. The next important step is to get structure into the cloud of requirements you will find at your customers site. It has become a good practice to group requirements into

- Business requirements
- User requirements
- Functional requirements
- Non functional requirements

The order of these requirement facets is not random. Business requirements come first, then user requirements and so on.

Cut it down

It was already said, that requirements engineering and testing have to be one process. To achieve this, there are "requirements" for the requirements

specification. For one, requirements have to be documented in a form suitable as input for testing. In the best of all worlds, requirements are documented to be understood by end users and the test system as well. This is very difficult to accomplish, especially if the software system to be developed is quite large and complex. The way out of this dilemma is: agile software development. If it's done right, the system is broken down into manageable packages. The most important (for business and users) parts of the software system will come first. Because the packages are small, there is a good chance that stake holders (end users) will be able to understand even quite formal requirement specifications. The next important step for the requirements engineer (together with the stake holders) is therefore: "Cut the beef into eatable pieces".

We modularized already! What's the difference?

It is important to understand the difference between the work of a software architect in a "classical" development process, where he/she cuts a complex system into components and modules and the work of a requirements engineer within an "agile team". For example, the software architect typically is used to split functions of the system into three tiers. He will decide which functions have to be located in a data management layer, which ones belong in a middle tier with communication and mediation functions and which ones are facing the end user at some kind of client. This is a purely technical way of looking at a software system (still makes some sense by the way). But from a user perspective, it's not of interest at all.

Requirements specification: Quality first

The challenge for the requirements engineer in an "agile team" is to identify chunks of functionality, in most cases containing all three tiers of the system and specifying them in a way, where end users are able to understand (and agree) and the testers of the team are able to test them in a very early stage. A good practice to document requirements for developers is the "use case". As implied by the name, use cases describe certain activities of the user, not technical functions. If the requirements are elicited, analyzed and documented with this in mind, then the requirements specification ideally is:

- Complete
- Correct
- Verifiable
- Unambiguous
- Consistent
- Implementable

Requirements which adhere to these quality criteria are well suited to be part of the quality assurance and testing process. In this case, there is a good chance that the testers will test functions having a one-to-one relationship to the requirements from the business/user side.

And again: what do we test?

This brings up another relationship with agile software development: Test Driven Development. Using this method, the team, after getting the requirements specification, immediately writes a test for these features. The test will be run, before the actual code for the required function is written. If it does not fail, then either the test program is defect, or the function required is already implemented. There is another inherent benefit in using such a method. Not only do we want to meet the requirements of the end users, we also want to avoid writing functions NOT required, just because a developer felt good writing "cool" code. This is accomplished by making sure, that only so much code is written, that the test is passed.

User acceptance, no more, no less

The next level of such a procedure is the "Acceptance Test Driven Development". This form of testing is more or less completely driven by the business/user side. Naturally, it only makes sense on a level, where some of the basic functions of a larger system are already implemented and can be used "stand alone". This fits the general procedure in an agile development. The team might for example use Scrum to "manage the development process". In that case, every two or four weeks, the team hands something "useable" over to the business/user side. The quantum tested is typically directly related to a user story, not a use case. The reason for this is that user stories are documented in the language of the user. An example for a user story would be: "as a warehouse manager, I want to check the stock of ACME-Cans".

Manage requirements

All of what we have discussed so far might sound quite reasonable to most of us. But, a requirements specification could be perfect; an acceptance test could enjoy your end users. It's all in vain, if it's just a single shot. As the project goes on, requirements change or new requirements pop up. This is the moment you need a thorough management of the changes. A method assuring that changing requirements and acceptance testing stay synced. That calls for test automation with regression testing and management. Traceability is the key. You might come into a situation, where one of the users, your customer, will ask you: why does the system behave like "a" and not like "b" and who asked when for "c"?

Everybody likes to win

The customer wins, if his user community gets exactly what they needed, at that point in time. All the "nice-to-haves" have been shifted to a version 2.0 in a consensual process. The project was carried out in time and on budget.
The software producer wins, if he was able to secure his margin (he has to feed his engineers) and if the customer is a good reference (which will help with the acquisition of new projects).

"Good requirements engineering": the answer

As we have seen, requirements engineering should not be a separate phase in the progress of a software project. It has to be firmly interwoven with quality assurance and management. Admittedly, this is much harder to implement in cases where "classical" project process models are still in place. To some respect, this creates a plea for agile software development, which is an iterative development process by itself. Iterative requirements engineering might look more expensive compared to a classical approach in the beginning, but the opposite is true. In the end, iterative requirements engineering has saved money and produced software fitting to the needs of the customer. ■

HOW DOES IT FEEL ...

... to have won an award this year?

Why to Apply:

Earn Recognition and Make a Difference

> Learn more about the award application process!
How to Apply:

> www.bqi.eu

> www.bqi.eu

BQI is a leading institute for business awards. Our focus is the performance of companies and employees. BQI's Leadership Awards are highly prestigious awards for outstanding achievements by businesses in the categories of Innovation and Quality.
Contact the BQI Award office via **info@bqi.eu** today!

Business Process Testing – A New Approach

intermediate

Author: ***Verónica Puymalie, Diego Marín, Marcelo Arispe***

About the authors:

Verónica Puymalie

Experienced IT & QA professional with more than 9 years in the industry. She has a unique experience in combining her development skills, database knowledge and testing understanding in large projects.

She has worked in several application domains and has a wide vision in the IT field as well as an strong experience on QA field, Test Case Design & Execution, Functional Testing, Regression Testing and White/Black box Testing.

Main areas of interest are focussed on Quality Assurance and Business Process Testing Methodology. Contact: +598 2 5185600, **veronica.puymalie@tcs.com**

Diego Marín

Diego has a degree in SystemsAnalysis with more than 13 years of experience in the IT field. He has been under different roles in the SDLC, from Analyst to Tester. During the latter years he has specialized in testing, and with main focus in Test Management.

He also has coached several teams in Testing Fundamentals; a task that he loves and do with passion.

He is an ISTQB Certified Tester and has worked in different fields, including Banking, Retailing and Healthcare. He has also worked for the United Nations for a specific program, under the Social Strengthening Areas for elder people in Uruguay. He has also lead TPI sessions for different testing departments.

He has led several testing teams under different locations, both functional and automation teams. He is currently leading a team which works under the Business Process Testing methodology in Tata Consultancy Services.

Contact: +598 2 5185600, **diego.marin@tcs.com**

Marcelo Arispe

Marcelo has worked at Software Assurance area for the last three and a half years. During this time he has performed several tasks: From designing test plans, verifying requirements, conducting manual and automating software verification. He worked under different roles from Tester to Test Lead, which allowed him to manage a group of people and achieve Team goals.

He has a Bachelor Degree on Systems Engineering, and he is an ISTQB Certified Tester. Among his experience, He has also acquired knowledge in the Web Development and Database Management, by which he has developed strong skills at these areas. He bilingual between English and Spanish, and he also has working knowledge of French and Hindi. He is familiar with Microsoft and Open source technologies as well as various peripherals, data entry and file updating.

Contact: +598 2 5185600, **marcelo.arispe@tcs.com**

Abstract

Business Process Testing (BPT) is an approach, developed by Hewlett Packard for its Quality Center Tool, with the aim of reducing gaps, most commonly present between Development and Business Analysis.

In this paper we will describe how this new approach can be used to improve the Software Development Process by achieving a more detailed, comprehensive and effective Quality Control process.

Introduction

Hewlett Packard (HP) has a single sentence to describe the main use of

their Business Process Testing (BPT) in a company as part of their Quality Center Tool:

“Bridge the quality gap between subject-matter experts and quality engineers.” (1)

With this single sentence, HP tries to catch the attention of IT Managers to buy this new approach to reduce the miscommunication between Subject Matter Experts (SME) and Quality Engineers.

BPT, in fact, provides a very powerful and versatile way to develop Test Cases starting from Use Cases by the introduction of the Business Components concept. The objective is to focus in Business Process, describe those using Business Components and finally generate the Test Cases that will cover that Business Process.

A Business Process is defined as a series of components which are built of steps; those components with their steps, when they are executed together, they create a value to the customer or to the product being shaped.

The way that BPT has to implement this new approach is by the use of their granular unit: a Business Component. There is a direct correlation among the screens that are part of an application, and the Business Components that will enable the creation of Test Cases. The work flow that makes this mechanism to work may be described in the following steps:

The new QC module called “Business Components” enables the user to create and manage reusable Business Components.

The Test Plan module enables the user to build a Test Case by dragging and dropping the Business Components into a new Business Process Test item. This will enable the user to debug those Test Cases and generate all the needed data to make different planned scenarios.

The Test Lab module enables the user to run Business Process Tests and manage the results.

Apart from this, there is another important fact that makes BPT a unique approach to enhance and optimize the work that is done in the Quality Area: the direct connection between Quality Center and Quick Test Professional (QTP). QTP has the feature of automating a Business Component, hence, once the Test Case is developed using the BPT feature, the Test Case is instantly automated.

We will cover all these features, along with the advantages and disadvantages of this new approach, and what was the need for its creation in the present White Paper.

BPT as part of natural evolution

The Software Industry has had several changes last years and old testing approaches that worked yesterday do not seem to work properly for today’s agile business environments, data warehouses, cloud computing applications and SOA systems. Nowadays, systems are aligned to serving business process and to improve data integration as well as disparate department applications. All these practical approaches, claim for

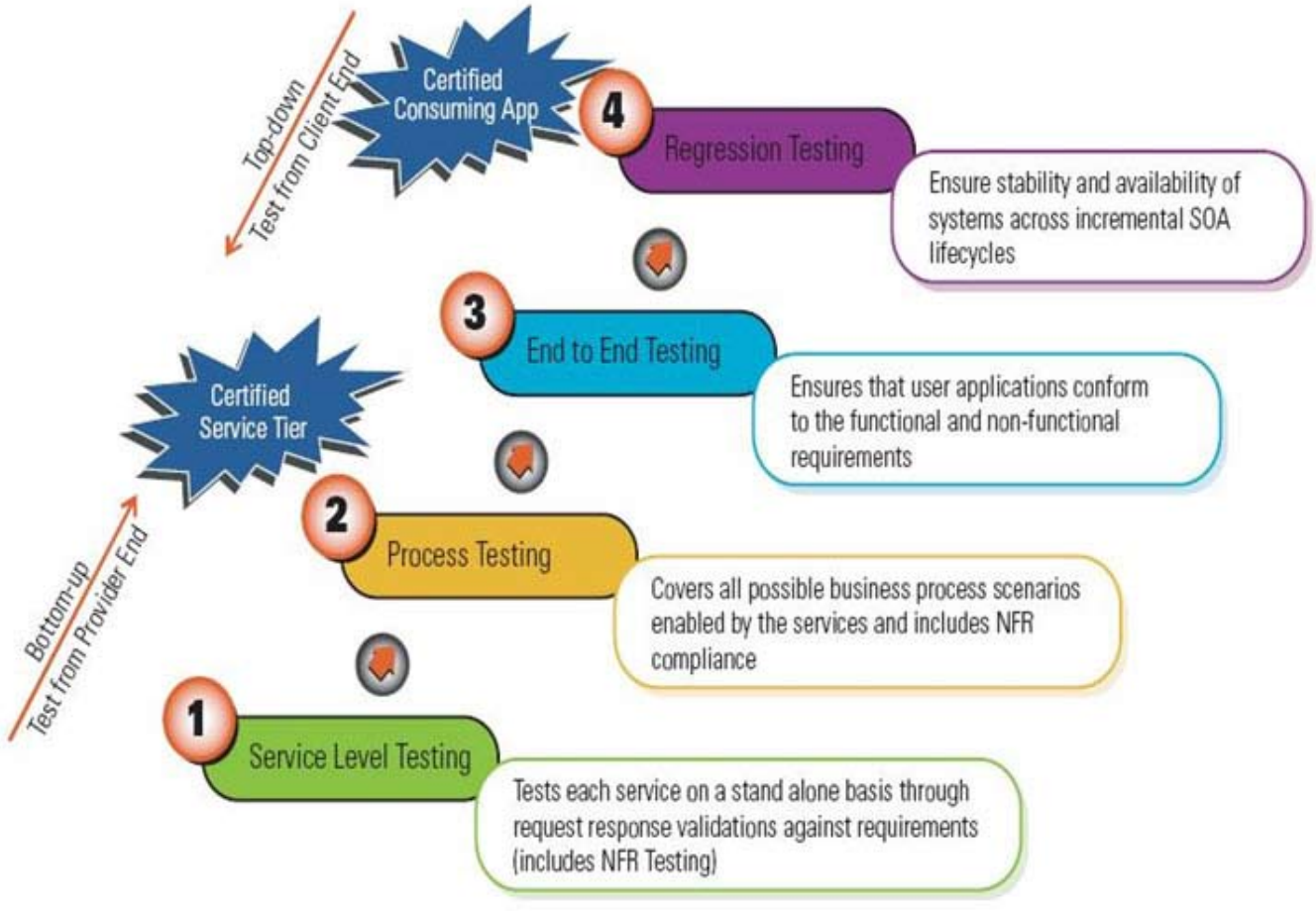


Figure 1 Testing Process Sample

automated testing to enter to the scene. So due to the accelerating nature of this evolution, the migration to test automation from conventional manual testing is a necessary step for business success.

Another key aspect that HP is focusing on, with this new Business Process Testing methodology, is that by aligning testing to the Business Process, there is a direct prioritization of functionality units being developed to business needs. With this approach, the most important components will be available, tested and working as expected, sooner in the testing process.

The diagram below provides a simple overview of a testing process:

Apart from the above mentioned items, one of the most challenging problems that nowadays IT projects face is the one related to communication, or better said, miscommunication. This miscommunication is not only related to the natural communication gaps that the IT crew has with the business counterpart, but also related so other additional facts that affect the relationship among team members. For example, in today's environment, it is common to have distributed teams, and due to this, communication plays a key role and may be a factor of success or failure for a project. Under such circumstances, Business Process Testing has an important role as a tool to try to reduce that communication gap among developers, testers, test analysts and Subject Matter Experts.

BPT leverages on the Keyword Data Driven technology that allows test cases to be represented in way in which SME's can collaborate effectively with QA and test without having technical skills. In this way functional subject matter experts can build, data-drive, execute and document tests without any programming knowledge and find the automation frameworks more handy & meaningful due to their continued involvement right from the beginning. This turns Test Automation towards being more business driven.

What is required for BPT

Let analyze if BPT really helps in bridging the quality gap between subject matter experts & test automation engineers.
Here are some facts to ponder before carrying on with the process of finding the true about this new trend.

The Yankee Group stated that about 90% of business mission-critical business processes have been automated by enterprise application. (5)

Besides, Gartner establishes that 80% of applications are not tested in a proper manner before they are released into production. However, pre-production testing is taking place, but it may lead to useless results if it is not focused in the business process. (5)

The 80% of the software development costs of a typical project are spent on identifying and fixing defects. (5)

With the above information in scope, it is clear that focusing on business process is a must for today's environment.

Notwithstanding this, there are other reasons on why to go for the BPT approach, as shown in Table 1 - Other reasons to focus on a new methodology.

Negligible involvement of the SME due to the high technical expertise.
Reusability and maintenance as a big issue.
Automation can start as early as it is necessary because they have to wait until the application is delivered to QA.
Defects being found in production instead of by your functional testing team that also hurts QA group credibility.

Table 1 Other reasons to focus on a new methodology

Along of these reasons, Business Process Testing is able to solve some of these persistent problems by simplifying and speeding up the testing software process by using Business Components.

System model

As you can observer in Figure 2 - BPT Interaction with Traditional Automation, QTP Experts can take part in the process only in some activities, leaving other tasks for Testers and SME's.

Create Components in Quality Center BPT module

The creation process for a Business Component in Quality Center is done via the BPT module[1].

The steps to create a Business Component are:

1. Create Initialization Component, where the user can detail the Description, precondition and post condition for each component.

2. Insert Input and Output Parameters. See Figure 3 - Parameter Definition in Quality Center.

3. Create steps for the parameters detailed.

1 The information provided in this white paper is related to Quality Center, Version 10.0.

How to create a BPT Test Case

The creation process for a BPT Test Case is done from the Test Plan module (as in a manual Test Case).

The steps to create a BPT Test Case are:

1.Create New Test Case

The creation of a Test Case includes the Name, Test Type, Preconditions and Objectives. (Figure 4 - New BPT Test Case shows how to define a BPT Test Case).

2. Drag & Drop Components to "Assemble" the Test Case

This is the main part where the components create the skeleton of the Test Case. (Figure 5 - Creating a BPT Test Case from Business Components)

3. Link Parameters between Components

This part is the entry of specific data into the components of the Test Case (Figure 6 - Specifying Data for a BPT Test Case).

Advantages

1. Once a Business component is

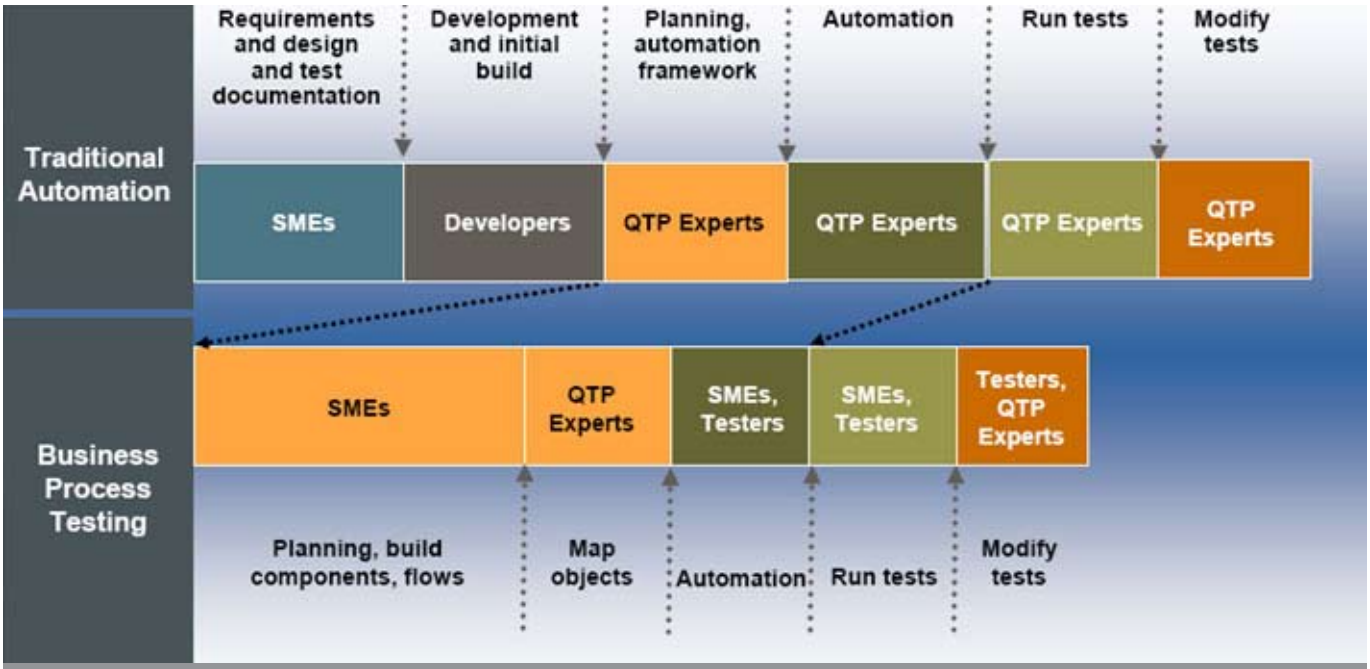


Figure 2 BPT Interaction with Traditional Automation (3)

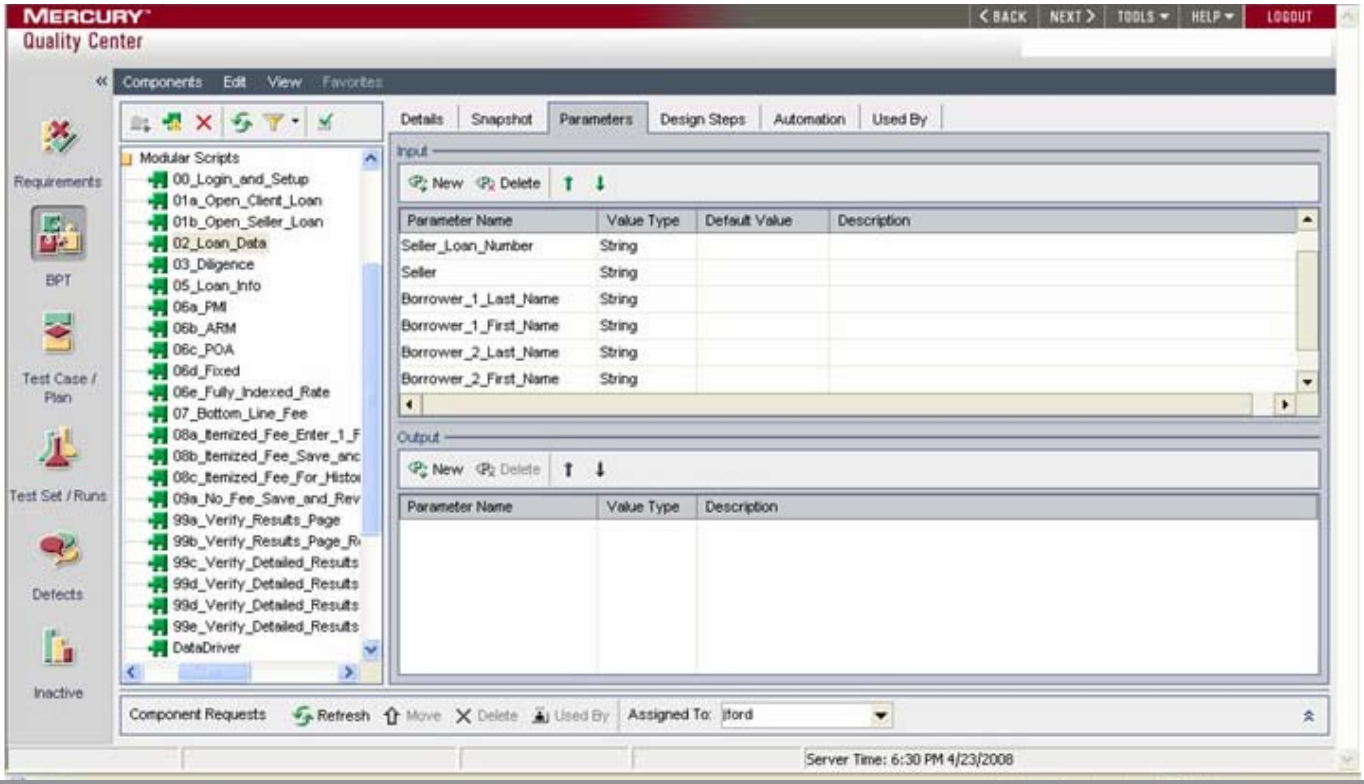


Figure 3 Parameter Definition in Quality Center

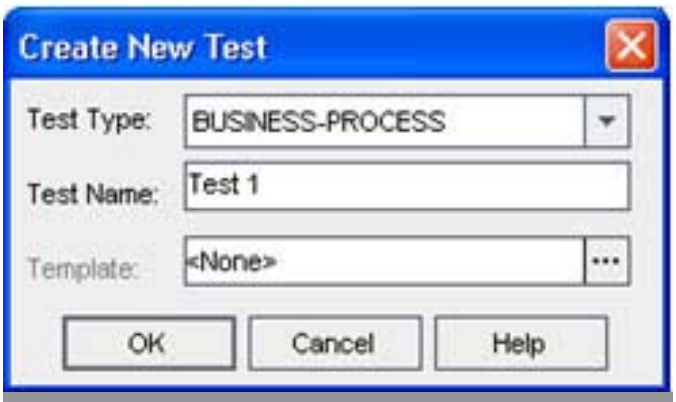


Figure 4 New BPT Test Case

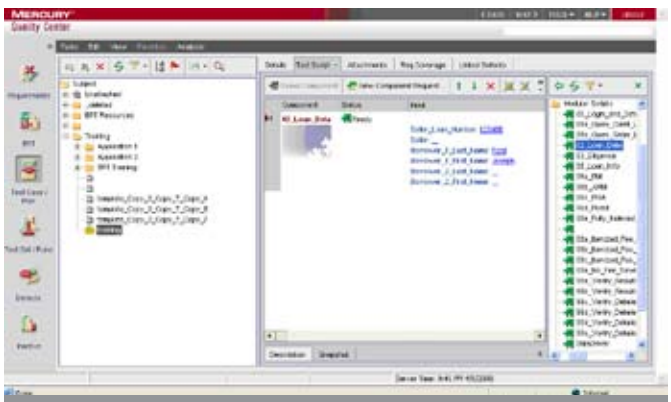


Figure 5 Creating a BPT Test Case from Business Components

created, it can be reused as many times as required for each test case, in any cycle of the Testing Process. That is why Regression Testing can be much more efficient with BPT due to this capability.

2. The design of Business Components is useful to line up Test Cases with Functional Requirements, even further to a deeper detail level, such as default values, type of fields, etc.

3. One tester can support more developers in a team because of the efficiency BPT provides.

4. Due to its high standardization level, Testing process gets more robust, regarding to the line and path all stakeholders have, such as development, business analysis or quality assurance.

5. Automation process improves its performance and efficiency due to the standardized way Business Components offer to it. Scripts written to execute Test Cases built from Business Components are also implicitly standardized.

6. In benefit of making life easier, Quality Center provides additional tools or plug-ins to handle Requirements, Test Cases or Defects, for example by providing already-made Microsoft Excel spreadsheets that have the capability to download or upload those work items. The current version of Quality Center does not provide an already made tool to upload Business Components. However, the OTA API that comes with Quality Center allows a user to create a macro that fulfills those requirements.

Disadvantages

1. The creation of Business Components implies the thorough description of every data Item that will be used on every screen of the application. This will imply that every label, tool tip text, drop down list's content, grid's contents, etc. need to be fully described. Due to this reason, the task of creating every Business Component is a very time consuming task.

2. In the current version of Quality Center, there are no reports available for Business Components. Furthermore, the database schema has naming conventions for the Business Component which are not clear. So reporting is hard to accomplish for this new module.

Conclusions

During our own experience and use under the BPT Methodology we found the following conclusions:

1. We were able to improve our communication with Development, Business Analysts, and Subject Matter Experts by speaking the same language of Business components. Making it more clear, detailed and comprehensible.

2. Automation process came even more easy and quick to implement after the test cases based from business components were designed due to the completeness of requirements specification and analysis.

3. As an overall conclusion, we can determine that the use of the BPT methodology makes communication among the different Testing stakeholders more reliable and easy, as well as comprehensive, in terms of reducing the gaps that are generally present on a team with different technical background.

4. As a final point, there is a seamless integration between Automation and BPT components, and that is the basic milestone, the key that opens the door to success, to actually reduce the time that is commonly present from the delivery of a particular feature, till its automation and finally implementation using any automation tool is done.

References

(1) HP Business Process Testing Software Datasheet https://h10078.

www1.hp.com/cda/hpdc/display/main/download_pdf_unprotected.jsp?zn=bto&cp=54_4000_100

(2) Michael Giacometti, HP customer perspective white paper: Best practices for implementing HP Quality Center software, 2007, P 3.

http://www.sqa.its.state.nc.us/library/pdf/HP%20Quality%20Center%20Implemenation%20Best%20Practices.pdf

(3) Span Technologies BPT

www.spantechnologies.com/SPANTechnologiesMercuryBPT.pdf

(4) Optimizing-manual-testing-with-hp-business-process-testing-software-whitepaper.

www.hp.com

(5) KDT - Sapient Business Process Testing Leveraging

http://edge.sapient.com/assets/ImageDownloader/150/Sapient_BusinessProcessTesting.pdf ■

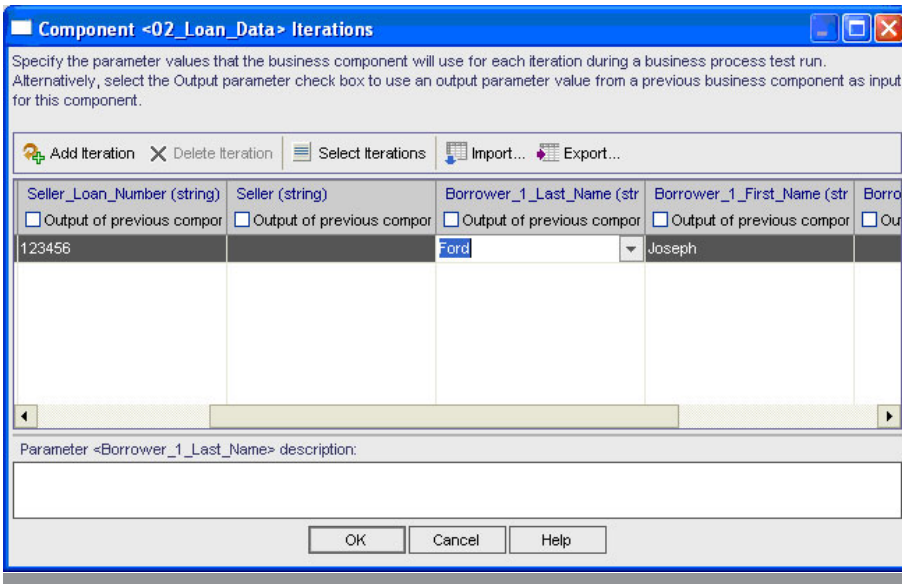


Figure 6 Specifying Data for a BPT Test Case

Ambiguity of defect severity definition

basic

Author: ***Andrey Konushin, Julia Salnikova***

About the authors:

Andrey Konushin

Expert in software testing and quality assurance. ISTQB Certified Tester, Advanced Level.

Master of techniques and technologies in informatics and computer engineering, Vladimir State University, Russia. His main research interests include information management and system analysis both in software testing and related fields.

Quality assurance expert and test manager at software development companies with big experience in tens of projects. CEO of "Knowledge Department Russia" – a coaching and consulting company, offering its services worldwide.

Since 2006 senior lecturer at Information Systems & Information Management chair of Vladimir State University. Andrey developed and implemented the "Software testing foundations" course for the University based on ISTQB Foundation Level Syllabus.

One of the founders of the Russian Software Testing Qualifications Board. Since 2006 he is the President of the RSTQB.

Julia Salnikova

23 years old. University degree in informatics and economics, specialization: Applied Computer Science in Economics, Vladimir State University, Russia.

Software tester at KAY-COM, information technologies company (http://kaycom.ru/). Julia has created a sophisticated sub-system of quality control in accordance with proven standards of quality in the information system of KAY-COM project management.

Bug tracking system is the central part of testing and programming communication during the software development. There are a lot of bug tracking systems existing, from the simplest and free to integrated tracking environments of high price. However, bug tracking system itself is the technical tool for fault information management. Such tool allows to formalize and automate defect management process, which is important part of overall development process. Both fullness and confidence of system faults information and order of fixing of these faults are depend on this tool. Thus, defect management is an organizational part of development process, but bug tracking system is implementation tool of this part. It seems, why investigate complex bug tracking systems, when defect description can be recorded into, e.g. Excel, and not to make additional efforts on buying, setting up and support of these complex systems? Mostly, information about defects are represented with a set of pre-defined fields, which serves for structuring this data. Such structure allows performing more valuable analysis of defects data. Moreover, defect usually has its lifecycle – set of states through which defect must pass to be fixed. Considering all these factors, it is clear that just excel sheets are not enough. However, some organizations still use Excel for defect tracking. This, undoubtedly shows maturity level of these companies.

In this article we will look at such defect attribute as severity. Working in different companies, projects and with different customers, misunderstanding of role of this attribute in defect description was not a surprise. Often, saying severity, people mean urgency of defect fixing, its priority. In fact, severity is the degree of impact that a defect has on the development or operation of a component or system [IEEE 610]. Severity is an important attribute from business point of view for which system is being developed. Exactly severity defines the impact of the defect on business-process. Usually, severity is defined by scale from 1 to 4. Different organizations in different projects and different bug tracking systems may use different scales, including the difference in "direction": "1" in one organization may be a higher severity, while in other organization it may mean vice versa – the lowest severity. Let us don't stop on this specifics.

Different understanding of the term “severity” leads to formalization of criteria by which severity must be defined. However, in practice, it is impossible to define these criteria for the company and use them in all projects. These criteria highly depend on the scope of project. Let’s have a look at the system of attributes which was developed in one of companies where we used to work.

Critical defect

• No complete functionality of the system (the user cannot perform the work) and there is no workaround of this problem.

• The application cannot continue working (buzzing).

• Application closed without warning.

• Compromise system security or integrity of user data.

Major defect

• Not available functionality, but there is a workaround.

• An unhandled exception are occurred.
Average defect

• Unexpected but handled by developer exception.

Minor defect

• The defect of non-business logic (insignificant defect does not lead to a breach or loss of user data).
• Deficiencies of user interface.

There are, of course, other classifications, but for clarity, we consider it such.

With such a scale, it is easy to determine the severity of the defect. Exceptions are rare cases where, for example, a grammatical error in the communication can lead to misunderstandings of the message, which in turn may lead to incorrect follow-up. Nevertheless, this classification covers uniquely the majority of new defects.

Such a classification has worked well so far, have not had to deal with systems in which data and user operations are more important than the actual running of this operation. This applies, for example, to banking systems, in which an incorrect calculation of the charge on transactions may cost a few hundred EUR to the client (and sometimes thousands), but more critically is that the operator cannot even guess that the charge is calculated incorrectly. Therefore, for such systems it is better not to perform an illegal operation, than do it wrong. This not applies to other, less critical systems. Consider several examples of defects of different applications in the context of this classification.

Example 1. Internet shop. When you click "More about the book" an error message appears.

In terms of the classification and the criticality of the system itself this is the failure of one of the functions of the system. In this case, severity is the highest.

Example 2. Automated teller working place in bank branch. After the registration and execution of currency exchange operations, order is generated incorrectly. According to the classification, such a defect can hardly be regarded as the most critical, because the operation is completed successfully, no error is raised. Nevertheless, in terms of business process - lack of cash order is critical, because the operation cannot be legally confirmed without it.

Example 3. Banking system. After removing service providers, the button “Apply Changes” on the form is not enabled, although it should be.

According to the classification, it is a critical defect, because the operation is not available. Nevertheless, it is more correct to assign this defect severity as “major” because the inability to delete records has almost no effect on the business process or user data (unless there are not many records and it is impossible to work with the application, which is unlikely in this context). In contrast to the operations that lead to loss of money from account or any other user data.

From these examples, it becomes clear that the criteria for determining the severity of the defect are ambiguous and highly depend on the scope of the application. At first glance, the same errors in the functioning of the application in fact have a different impact on be automated business process. Exactly this influence determines the severity of the defect. Nevertheless, the definition of severity should not be different for different members of the team of one project, because it can lead to distortions in the prioritization of bug fixes and new development. Obviously, a “common denominator” is needed and static criteria of severity cannot be used, these criteria should be evaluated and modified as necessary.

And as we said above, the prioritization of found defects fixing, of course, must take into account the severity, but the severity should not be the sole factor in determining the queue of correction. You should take into account such factors as team workload, effect of correction of the defect to other parts of the application and other factors. Defining of priority of defect fix is a more sophisticated and informal task which lies on the shoulders of dedicated staff, in contrast to the severity, which can be set by any member of the team who found the defect: from the developer to the customer. And for a more structured work it is desirable to formalize definition of severity. ■

Agile When It Works

intermediate

Author: ***Rex Black***

About the Author:

With a quarter-century of experience, Rex Black is President of RBCS (www.rbcs-us.com), a leader in software, hardware, and systems testing. For over fifteen years, RBCS has delivered consulting, outsourcing and training services to clients ranging from Fortune 20 companies to start-ups. Rex is also the immediate past President of the International Software Testing Qualifications Board and the American Software Testing Qualifications Board. Rex has published six books which have sold over 50,000 copies, including Japanese, Chinese, Indian, Hebrew, and Russian editions. He has written over thirty articles, presented hundreds of papers, workshops, and seminars, and given about fifty speeches at conferences and events around the world. Rex may be reached at **rex_black@rbcs-us.com.**

Greetings, and welcome to my quarterly column on software testing best practices. When I was asked to write this column, I had to choose the approach, the theme. The writers' aphorism says, "Write what you know." So, what do I know?

Well, if you know me and my consulting company, RBCS, you know that we spend time with clients around the world, in every possible industry, helping people improve their testing with training or consulting services, or doing testing for them with our outsourcing services. Our work gives me insights into what goes on, the actual day-to-day practice of software testing.

Now, not all of what goes on is good. There are bad practices, and we help clients fix those. But you don't need me to write about what not to do. Aren't there enough scolding bloviators in our business? With a click of your mouse, you can read these people's disdainful rants about testers they think are stupid, testers they think are in the wrong "school of testing," testers they love to hate. Lecture, scold, rant, bloviate. How tedious!

So, being a contrarian, I will do the opposite: With the exception of the paragraph above—where I poured well-earned scorn on people who write bad things about other testers—this column will be 100% good news. I will discuss testing best practices that my associates and I have observed other smart people doing. That's right. No negativity and no bragging about myself either. A simple theme: What other people do right when they test and why we love it.

I want to start with Agile testing when it works. No, I'm not recanting. Yes, I've written about the testing challenges of Agile, and I stand by what I wrote. Yes, I can talk about testing worst practices in some Agile teams, and I might in some future article—but not in this column. In this column, I focus on what's right about Agile. Here are five testing best practices

we've found in Agile done right:

Unit testing. Okay, it's true that most programmers, even Agile programmers, still have a lot to learn about proper test design. But if you're a professional tester like me, you have love hearing programmers talk about the importance of unit testing. We all know that unit tested software is easier to system test.

Static analysis. Not only do smart Agile programmers like unit testing, they like static analysis, too. Coding standards are hip again. Cyclomatic complexity is back. Writing more testable, more maintainable code: that'll make testers' lives easier in the long run.

Component integration testing. This under-appreciated test level exists—on properly run Agile projects. You can go years on sequential-model projects without seeing component integration testing. However, on a good Agile teams, people look for integration failures, and, because of continuous integration, the underlying integration bugs aren't hard to find.

Tools, tools, tools—and many free. All of this talk about unit testing, static analysis, and component integration testing would be just that—talk—without tool support. Fortunately, the Agile—err, what should we call it?—movement, revolution, fad, concept, pick your term, has brought with it a lot of tools to support these best practices, along with other best practices. For those of us without unlimited budgets—and isn't that all of us?—a lot of the best tools are free, too.

Tester and developer teamwork. At the beginning of our latest assessment, I had a great conversation with a test manager who works on Agile projects. Among areas of agreement: our shared joy at the death of a bad idea. The bad idea in question was this: the idea that the role of the test team is the quality cop, the enforcer, the Dirty Harry to the punks of the software team. "Seeing as I can refuse to approve the release, you gotta ask yourself one question: Do you feel lucky, programmer?" Instead, we see more people working together, collaborating for quality, and that's especially true on good Agile teams.

Just this morning, I spent three hours talking to two programmers—real seasoned professionals with years in the field—talking to them about testing. The testing that they did. In fact, it wasn't so much about testing, but testing as an essential tactical element in a larger strategy for higher quality code. They really knew testing, and they knew how the Agile approach and tools were helping them to achieve better testing and thus better code. At the end of our talk, I mentioned how much I enjoyed talking to programmers about good testing and good code.

He replied, "Yeah, we spend a lot of time around here talking to each other about that. How to be better craftsmen. How to test better. How to build better code."

Wow. If the entire methodology, the lifecycle, the tools, and every other aspect of Agile fades away, leaving behind only the habits of programmers serious about code quality, and testers working cooperatively with them to achieve it, that will be a signal achievement in the software engineering profession. Best practices, indeed.

So, there are the first five best practices. What comes next? That depends on which great practices my associates get to see in the next three months. See you then. ■

Metrics for Software Testing: Managing with Facts:
Part 1: The Why and How of Metrics

intermediate

Author: ***Rex Black***

About the Author:

With over a quarter-century of experience, Rex Black is President of RBCS (www.rbcs-us.com), a leader in software, hardware, and systems testing. For sixteen years, RBCS has delivered consulting, outsourcing and training services to clients ranging from Fortune 20 companies to start-ups. Rex has published six books which have sold over 50,000 copies, including Japanese, Chinese, Indian, Hebrew, and Russian editions. He has written over forty articles, presented hundreds of papers, workshops, and seminars, and given about seventy-five speeches at conferences and events around the world. Rex is also the immediate past President of the International Software Testing Qualifications Board and the American Software Testing Qualifications Board. Rex may be reached at

rex_black@rbcs-us.com.

Introduction

At RBCS, a growing part of our consulting business is helping clients with metrics programs. We're always happy to help with such engagements, and I usually try to do the work personally, because I find it so rewarding. What's so great about metrics? Well, when you use metrics to track, control, and manage your testing and quality efforts, you can be confident that you are managing with facts and reality, not opinions and guesswork.

When clients want to get started with metrics, they often have questions. How can we use metrics to manage testing? What metrics can we use to measure the test process? What metrics can we use to measure our progress in testing a project? What do metrics tell us about the quality of the product? We work with clients to answer these questions all the time. In this article, and the next three articles in this series, I'll show you some of the answers.

Why Should We Have Metrics?

Sometime I hear people asking why metrics are necessary, or worse yet disparaging metrics with smug comments like: "Not everything that you can measure matters, and not everything that matters can be measured." To me, such remarks are like questioning the value of literacy, or saying that reading is unimportant because you don't need to read to appreciate good art.

Metrics allow us to measure attributes. Metrics allow us to understand. Metrics allow us to make enlightened decisions. Metrics allow us to know whether our decisions were the right ones, by assessing the consequences of those decisions. Metrics are rational.

Plus, you really don't have much alternative to metrics usage. The other option is to base your understanding, decisions, and actions on subjective, uninformed opinions. This is not a sound basis for management.

You might be thinking; "I'm a reasonable person, and I make all sorts of smart and reasonable decisions in my everyday life without metrics." Well, maybe. First off, you might have grown so accustomed to all the metrics we have around us that you didn't notice them. When we drive, we refer constantly to a key metric: speed. When we shop, we mostly use the metric of price. In many situations, when you find yourself without the usual metrics, you might feel lost.

Second, when people make decisions or reach conclusions without metrics, based on what sounds reasonable, they can be wrong. My favorite example of this comes from the Greek philosopher, Aristotle. Aristotle was a smart fellow, and he said

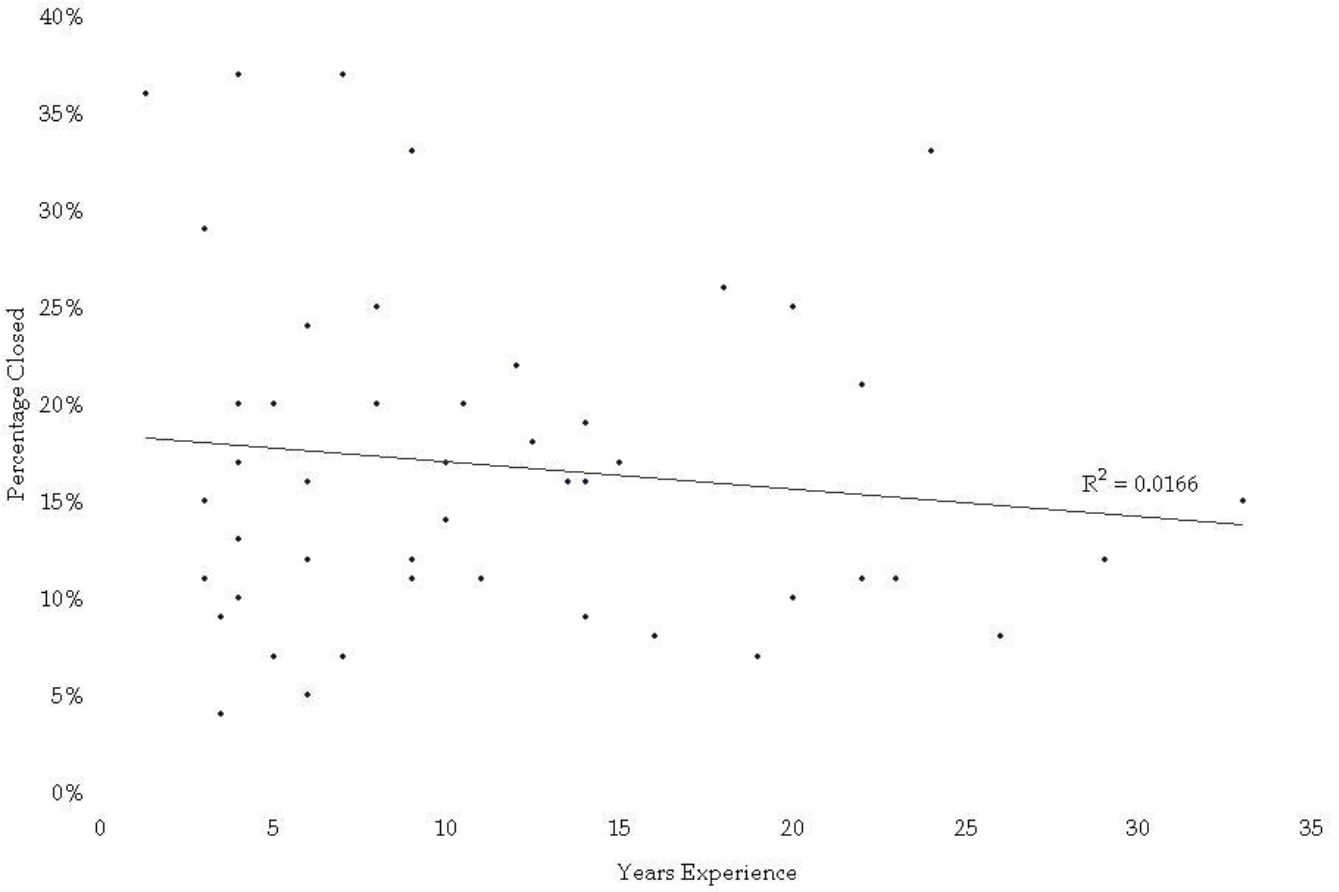


Figure 1: Data Disproves Widely-held Opinion

a lot of smart things. However, he also said that heavier objects fall faster than lighter objects.

That sounds reasonable, and anecdotal evidence like feathers and stones are all around us. Two thousand years later, though, Galileo dropped two cannonballs of very different weights from the Leaning Tower of Pisa. Both hit the ground at the same time. Simple experiment. Simple metric. Two thousand years of misguided thought overturned with a single thud.

In my consulting work, I often tell clients that the most dangerous kind of mistake is the mistake that sounds reasonable. Something that is wrong and that sounds stupid is harmless, because people will reject those statements out of hand. Reasonable-sounding mistake, that just might trick people. In fact, we've seen that happen.

Here's an example. We do a number of test process assessment engagements for clients, and one of these clients makes complex industrial-control systems that run oil refineries, pharmaceutical plants, and other critical equipment. In our assessment, we found that they had a very high rate of bug report rejection. Bug report rejection occurs when the report turns out to describe correct behavior, rather than the symptom of a bug. When the bug reporting process is working well, the bug report rejection rate should be under five percent. In this case, we found the client had about 20% of bug reports rejected as not due to faulty behavior. When I asked why the rate was so high, almost everyone believed that the reason was insufficient end-user experience using industrial controls.

It sounds reasonable, huh? People who don't understand complex systems might not draw the right conclusions about what constitutes correct behavior, right? Well, it turns out that the data easily disproved this reasonable—but mistaken—opinion. I created the scatterplot shown in Figure 1. The scatterplot shows the percentage reports rejected, on a tester-by-tester basis, versus the number of years of actual plant experience each tester had. As you can see, the R2 value—which measures the level of statistical correlation—is very close to zero. So much for reasonable.

Metrics are valuable whatever we are doing, but I think they are particularly important for testing. This is true because testing by itself, in isolation from the rest of the project, has no value, but it produces potentially valuable information. In order to obtain the value, this information must be generated and communicated effectively. That involves some form of testing metrics.

Effective communication is communication that serves a purpose. There are three fairly common goals of communication of test information, and all three are enhanced by metrics.

We might want to notify people of the status of testing. For example, we might want to make people aware of the bug backlog that has accumulated. In such a case, it's more appropriate and powerful to say, "We have 24 bugs remaining to close," than to say, "There are still bugs in the backlog."

We might enlighten people as to the impact of some attribute of the process. For example, we might want to help people understand that we are working inefficiently because many bug fixes fail confirmation testing. It's better to be able to measure and report the number of lost

person-hours resulting from confirmation test failures than to simply exclaim, “It’s very frustrating and inefficient to deal with all these lousy bug fixes that the programmers send us!” Of course it’s not always possible to evaluate an exact number of lost person-hours, however even an approximate value will shed some light on the situation we are facing.

We might also want to influence people to choose a particular course of action. Going back to the example two paragraphs ago, where we have a large bug backlog, we might show a breakdown of bugs by severity, and then propose a bug triage meeting to defer unimportant bugs in order to focus attention on the more important ones.

For our clients that are following best practices in their use of metrics, we see that some metrics are reported regularly as part of status reporting. These are sometimes called dashboards. They can be process, project, or product focused. Such dashboard metrics might have as their goals notification, enlightenment, and/or influence. Other metrics are reported as needed, after some analysis of a situation that has arisen. Such metrics would more commonly have enlightenment (why did the situation happen?) and influence (what should we do about the situation?) as their goals, than merely notification.

How Should We Develop Metrics?

So far, we’ve seen why metrics are useful, how metrics help to deliver the value of testing, and how metrics can serve specific communication goals. What metrics should we use, though? Is it enough to simply adopt the metrics that a tool like Quality Center produces? In my experience, such test management tool metrics are not sufficient, and in some cases are counterproductive. Such tools produce large amounts of very tactical metrics that can prove useful to test managers, but which are typically overwhelming and even misleading to people without a testing background. While test management tools are valuable to collect the raw data behind metrics, you should use a top-down approach for defining metrics, not a bottom-up approach.

By “bottom-up approach” I mean letting the tool define the metrics you will report. By “top-down approach” I mean starting with a clear picture of the objectives you are trying to achieve, and then deriving the metrics from that.

Identifying the objectives for testing and quality can prove challenging for some organizations, because not many organizations are used to thinking about what the testing and quality objectives are. I realize this statement sounds strange, but it is true. Ask yourself: Do you have well-defined, realistic, documented, agreed-upon objectives for your testing process? When we start working with clients, the answer is usually “no”.

Typical high-level objectives for the test process as a whole are:

• Find bugs, especially important ones

• Build confidence in the product

• Reduce risk of post-release failures

• Provide useful, timely information about testing and quality

You might have other objectives, and that’s fine.

Given a defined set of objectives, we can ask three types of questions about the degree to which we achieve those objectives:

• To what extent are we effective at achieving those objectives?

• To what extent are we efficient at achieving those objectives?

• To what extent are we elegant at achieving those objectives?

Let’s define what these concepts of effectiveness, efficiency, and elegance mean with respect to the achievement of objectives.

Effectiveness has to do with producing a desired result, in this case the objective. Efficiency has to do with producing that desired result in a way that is not wasteful and, ideally, minimizes the resources used. Of course, at some point, trying to increase efficiency starts to reduce effectiveness, as anyone who has driven a small, highly fuel-efficient vehicle knows.

What about elegance? Elegance has to do with achieving effectiveness and efficiency in a graceful, well-executed fashion. Elegance impresses. Elegance work resonates as professional, experienced, and competent.

Some people might ask, if we are effective and efficient, why should we care about elegance? Consider this example. Let’s suppose that you go into a café to get a cappuccino. You are in a hurry, and want to quickly get your caffeine fix and go. The line is short and the cost is low. The wireless signal is strong and free, so for the limited time you are waiting, you can get some work done. Within two minutes, you have your cheap cappuccino, it tastes excellent, and you are out the door with your to-go cup in hand. The café effectively and efficiently satisfied your objective. Are you happy?

Maybe not. What if the cashier was rude and lazy, almost overcharging you for the drink until you pointed out her math error? What if the man making your drink was dirty, smelled bad, and had long, greasy hair that was clearly shedding into people’s drinks? What if the place as a whole was not very clean or very pleasant to look at? In such a situation, you would not consider the place a very elegant way to satisfy a caffeine craving.

With the need for elegance established, let’s return to the issue of metrics. We should devise at least one metric each to determine the extent of our effectiveness, our efficiency, and our elegance. This metric should be something we can actually measure, of course. People having an elegant experience probably has a dopamine release in their brains, but that’s not likely to be something you can check easily. A better idea would be a stakeholder satisfaction survey using a Likert scale (e.g., asking satisfaction levels ranging from very satisfied to very dissatisfied).

In some cases, it’s very difficult to measure something directly. In such situations, you can use a surrogate metric. As an example, suppose I gave you a tape measure and sent you into a parking garage to weigh the vehicles in that garage. Could you do it? Well, you can’t directly weigh the vehicles, because you don’t have a scale. However, you could use the tape measure to calculate the volume of each vehicle. You could use the volume as a surrogate metric for weight, simply by making the simplifying assumption of relatively constant density.

In fact, if you were the owner of just one of the vehicles, you could use the owner's manual in the car to determine the actual weight. This would give you a known density for one vehicle, and you could then use that to calculate (via the volume) the weight in kilograms or pounds for the rest of the vehicles in the garage.

I'll give one example of each type of metric, direct and surrogate, in just a few paragraphs. In subsequent articles in this series, we'll see many examples of metrics, including direct and surrogate metrics.

It's important to say that it's not enough to just have a metric. We need to know what constitutes a good measurement for that metric. So, once the metrics are defined, we should set a goal for each metric. One way to set the goals is to measure where you stand now. (This is sometimes called baselining.) Another way to set the goals is to compare yourself to industry averages or best practices. (This is sometimes called benchmarking.)

One way not to set goals is to pick arbitrary, extreme values, though I'm afraid this does happen. For example, we have seen situations where tester were expected to find 100% of all bugs, while at the same time programmers were expected to have zero bugs in their code. Both of these goals were instituted in the testers' and programmers' annual performance evaluations, respectively. In this case, the managers had actually made two serious mistakes. They violated best practices for setting goals for metrics and violated the rule that process metrics should not be used for individual performance appraisal.

With proper goals in place, you should think about exceeding those goals. What improvements could you implement that would move the metrics towards higher levels of effectiveness, efficiency, or elegance? It's certainly true that at some point any process will have reached adequate levels of optimization, but it's also true that it's very rare for us to work with clients on metrics programs and find that everything is perfect the first time we baseline the processes, projects, and products.

So, we can summarize the process of deriving metrics as follows:

1. Define objectives.

2.Consider questions about the extent of effectiveness, efficiency, and elegance with which we realize the objectives.

3.Devise measurable metrics, either direct or surrogate, for each effectiveness, efficiency, and elegance question.

4.Determine realistic goals for each metric.

5.As appropriate, implement improvements that improve effectiveness, efficiency, or elegance as measured by the metrics.

With the process clear, let's look at two examples of metrics devised following this process[1].

First, let's look at one of the common objectives for testing, finding bugs. On a project, one of the key questions is whether we are finished finding new bugs. (This is often included as an exit criterion in test plans.) As a metric, we can plot the trend of bug discovery over time during test execution. An example of such a metric, at the end of the project, is shown in Figure 2. Our goal is to see the flattening of the cumulative bug opened curve (the upper line in the graph).

Let me point out that we also have another project objective, the resolution of known bugs, which is also shown in this metric (the lower line in the graph). It's not unusual to find ways to combine metrics on a single graph or table, and this combination can be quite useful to compare and contrast process, project,

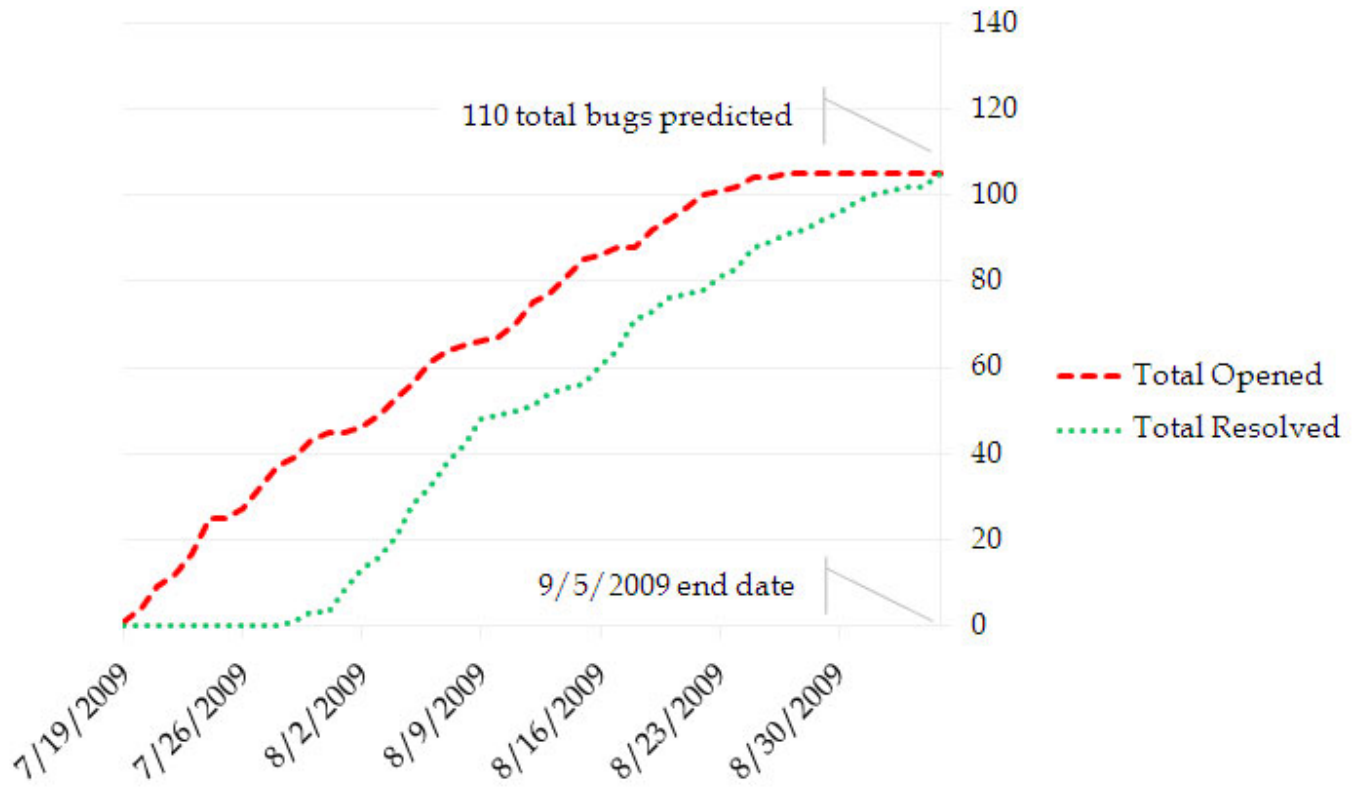


Figure 2: Bug open and resolution trends

Requirements Area	Currently Untested	Tested and Failed	Tested and Passed
Functionality	7%	3%	90%
Usability	25%	17%	58%
Reliability	0%	17%	83%
Performance	5%	10%	85%
Installability	7%	13%	80%

Table 1: Requirements Coverage by Area

Table 1: Requirements Coverage by Area

or product attributes that are illustrated by the two or more metrics shown.

This chart also helps nudge us towards two obvious improvements. If we were to find (and resolve) bugs earlier, we could finish the project earlier. So, how can we shift the bug opened and resolved curves left, towards the start of the project? How can we shift the total number that we discover downward on the vertical axis? Fewer bugs, found and resolved earlier: that sounds smart, doesn't it?

1 For a complete discussion of this process, you can read my chapter in the book Beautiful Testing

Next, let's look at another common objective for testing, building confidence. We would want to achieve a significant level of confidence prior to releasing software to our customers. However, how can we measure confidence directly, as confidence is a state of mind? For confidence, we can use coverage as a surrogate metric: the more thoroughly tested the product is, the more confident we can be that the system has no surprises in store for us (or the customers or users) after release. Now, coverage is a tricky concept, because coverage has multiple dimensions, including code coverage, design coverage, configuration coverage, test design technique coverage, requirements coverage, and more. Certainly, for higher levels of testing such as system testing and acceptance testing, a key question is whether any requirements have identified failures. So, our metric can include three elements:

• How many requirements are completely tested without any failures?

• How many requirements have failures?
• How many requirements are untested?

These are typically measured as percentages, as shown in Table 1. At the end of testing, the goal is to test 100% requirements, with no known must-fix failures at the end. As an improvement, we can look at ways to reduce the percentage of requirements that fail in testing when first tested.

What Else is True of Good Metrics?

Lots of organizations have metrics programs, but the metrics are not always very good. This is not always due to a failure to follow a good process for developing metrics, such as the one outlined above, though certainly bad metrics-development processes are a major contributor. What else can we say about good metrics?

Certainly, we want our metrics to be simple and effective. Simple means not just simple to gather and calculate, but also simple to understand. Effective means that the metric is obviously and actually connected to parts of the software process in such a way that we know what actions to take to move the metric in the desired direction. This property is something one of my clients refers to as the "so what?" question for metrics.

Since we were just on these topics, metrics should be efficient and elegant, too. Efficient means that we can produce the metric without an excessive amount of work; the effort required to produce an efficient metric is repaid by the value we receive from that metric. Elegant means that the metric is seen by the audience as a pleasing and smart way to present the information.

So, what is true of metrics programs that have simple, effective, efficient, and elegant metrics?

Such a set of metrics are useful, pertinent, and, especially, concise. While it can be tempting to measure absolutely everything, you should avoid too large sets of metrics. Such metrics will prove too difficult to measure in practice (and thus inefficient) and usually very confusing to participants (and thus inelegant). To be clear, there is value in considering a large variety of metrics when first setting up your metrics program; however, once implementation and regular measurement starts, you should settle on a limited number.

That said, it's also important that the metrics be sufficient in number and diverse enough in perspectives to balance each other. For example, consider again the bug trend chart shown in Figure 2. I said that we want the cumulative opened curve to flatten, and for the cumulative resolved curve to intercept the cumulative opened curve, as we get to the end of testing. That's true, but by itself is out of balance, because we might have stopped finding new bugs when our testing is completely blocked. In such a case, notice that the requirements coverage metric I mentioned as the second example balances this bug metric, because the requirements coverage metric will be stuck below 100% tested and passed.

To make the metrics simple to gather, calculate, track, and present, you must consider the implementation of the metrics. Automated tool support can be very helpful in this regard. However, be careful, because it's easy, once the tools get involved, to let the built-in metrics of the tool determine what you will measure (which is back to the "bottom-up" mistake I mentioned earlier).

Implementation of the metrics should also consider the proper way to track and present a given metric, because proper presentation is a major factor in making a metric simple and effective. You have three general options. Metrics can be

presented as snapshots of status at a moment in time, as shown in Table 1. Metrics can show trends emerging over time, as was the case in Figure 2. Metrics can also show the analysis of causes and relationships between factors that influence testing and quality outcomes, as we saw in Figure 1. The formulation of clear objectives and questions related to them, as discussed earlier, should help you make the choice here. However, if in doubt, try various options and see which one suits best to your process.

Making the metrics simple to understand is not likely to happen without some education. Part of a successful metrics program is ensuring uniform, agreed interpretations of the metrics. Clear understanding of what the metrics tell us helps to minimize disputes and divergent opinions about various measures of outcomes, analyses, and trends that are likely to occur when we measure projects, processes, and products. Remember that reporting of metrics should enlighten management and other stakeholders, not confuse or misdirect them.

So, when presenting metrics, be sure to provide objective analysis, tempered with appropriate and balance subjective interpretation. This is especially true when trends emerge that could allow for multiple interpretations of the meaning of the metrics. Of course, we want to avoid complex and ambiguous metrics that tend towards such confusion, but the problem is not merely one of metrics design and stakeholder education.

When using metrics, we have to be aware of and manage the tendency for people's interests to affect the interpretation they place on a particular metric. Three psychological dynamics tend to create problems in the use of metrics here. The first is confirmation bias, which is the tendency to accept facts and opinions that confirm our own existing opinions, and reject other contradictory facts and opinions. For example, the project manager who is sure the product will release on time (and whose bonus depends on it) will have some significant confirmation bias with respect to test results showing a large and growing backlog of bugs.

The second is cognitive dissonance, which are the feelings of confusion, anxiety, frustration, and even anger that result from trying to simultaneously have incongruous beliefs, attitudes, and understandings. The project manager who starts to understand the implications of the bug backlog will soon experience cognitive dissonance.

This leads to the third psychological dynamic, which is transference. In transference, a person transfers how they feel about some particular situation onto someone or something else. In this case, the project manager might transfer their anger over the delay in release onto the test manager who is reporting the test results, since it was that test results which made the project manager unhappy. Confirmation bias, cognitive dissonance, and transference are all common human psychological dynamics, and you certainly cannot change human nature. However, you should be aware of how these psychological dynamics will affect people's response to metrics.

When thinking about using metrics for reporting purposes, keep the goals in mind: good testing reports based on metrics should help stakeholders and managers improve processes, guide the project to success, and manage product quality. You should check with the people who are using the metrics to make sure that the metrics are working for them. I recommend to use the Likert scale survey that I mentioned early to assess the usefulness of the metrics for the stakeholders.

Moving on to the Process

In this article, I offered a number of general observations about metrics. We've seen the importance of using metrics to manage testing and quality with facts. We've looked at the proper way to develop metrics, top-down starting with objectives rather than bottom-up starting with tools. We've seen two examples of metrics for testing. We've also looked at some rules for recognizing a good set of metrics.

In the next three articles in the series, we'll look at specific types of metrics. We'll start with process metrics, because these metrics are the least used and least understood of the three types. See you in the next issue! ■

Validation Testing (the good “Happy Path”), Falsification Testing (the Bad) and a word about TDS Test Design Specifications.

intermediate

Author: ***Yves Souvenir***

About the author:

Yves Souvenir serves as an test management consultant since 2004 with contributions to the testing departments in the banking industries, and the establishment of independent testing department at the European institution. He holds ISTQB certificates and a master of science of the University of Brussels in Aerodynamics since 1991.

Introduction

I do not like when I hear the comment, “ A user would never do that”, it is all a matter of time and it will happen! Remember the Airbus Crash leaving from Rio to Paris flight AF 447? Four independent speed indicators failed to operate due to high altitude icing during the climb of the aircraft. A problem know since 1947 before even jets where introduced. Massive Ice crystals manage to block the pencil shaped airspeed indicators on the aircraft who are hot as hell.

It was not a user scenario so it was not tested and pilots were not trained in the simulator to handle the situation. Nevertheless they would be able to handle the emergency situation if software did not ‘throw in the towel’ without having the speed readings available and, if only the pilots were able to override the system like it is possible in the Boeing aircrafts. A situation which doesn’t make life easier to the pilots.

One alarm after another lit up the cockpit monitors. One after another, the autopilot, the automatic engine control system .., just like a Christmas tree. Consequently the flight computers shut themselves off, because of the Unknown situation. The final minutes of flight AF 447 had begun. Four minutes after the airspeed indicator failed, the plane plunged into the ocean, killing all 228 people on board.

We as tester should think beyond the box and write down every possible non-scope item (Falsification Test), and never think about the above Sentence which I was several times remarked by a developer.
Test cases must include both Validation tests (the Good, (the happy path)), test that verify functionality using expected input and Falsification tests (the Bad), tests for user unexpected data to see whether the program handles that data appropriately.

Verification tests are necessary to prove that the application works as intended but the falsifications tests are More Important.

Systems need to be Robust and handle bad Data without Error.
One of the great things will be with the enormous emphasis on security by using Falsification test; defects found, are resolved as ‘Customer would never do that’.

The Happy path should always Pass.
One day I arrive at the office and found a mail of Regis a colleague developer in my inbox, telling that he worked on a new component for a several days and

wanted me to do the testing as soon as the build came out.

My schedule was tight but I was excited about finally being able to test the component. In fact I wrote already the test objectives and the purpose why the test exist on a discussion and a presentation Regis gave some weeks ago about the new component.

Later that day indeed the new component was installed in DEV, and I immediately went to the menu bar on the left side with the new component, entered some common simple data and .. it didn't work!
Simple inputs should always work I thought; simple inputs are the "happy path" to me.

Because to me the Happy path must always work I immediately assumed that I must have done a mistake somewhere. (I knew I went through steps a little too fast and probably mist some selections. So I tested it on another machine and did is slowly. Unfortunately I had the same results.

I tested some more times and but had always the same result. So I called Regis with the bad news.

When I described to him for the last then minutes, he said hmmm, I made a change just before checking the files in, but didn't think that it would make a difference.. I guess I was wrong. It had a downstream effect this change in the files.

Coming at this point, I wasted more than an hour and felt irked and Replied to Regis: "Seriously Regis the happy path should always pass."

I remember and repeat this phrase every time something that should work doesn't!
About TDS (Test Design Specifications).
The Process of designing test is as equal important as the act of designing end-user software.

TDS, Test Design Specification is applicable for both manual and automated tests, an typically has the same review process as other documents such as specifications and design documents used in the software engineering process.

Because TDS describes both the approach and the intent of the testing process, it becomes an integral part of the testing process throughout the entire life of the product, especially during the post ship phases of the software's life when a sustained engineering team might own the product support.

TDS Test Design Specification for the Falsification and Verification Tests.

• Overview/ the goal of the test and why it exists!

• A strategy: this is high level approach, it is risk based and proceeds from the risk anaylsis.

• Internationalization and globalization testing; act like the test will be read by the world.

• Functional testing

• Component testing

• Integration/system testing

• Interoperability testing

• Compliance and conformance testing

• Performance Testing

• Security Testing's

• Setup /deployment testing

• Dependencies

• Metrics.

Conclusion

The starting point as we all know must be as early as possible.

Start in an early stage with the Falsifications Test, Write the test objectives and let them approve by the Test Manager.

He will be very grateful of the combinations of the selected test strategy, and he describes the use of tolerances in the system about the test execution with the client.

The Normal data must always work! ■

Data-Driven Testing with Selenium

intermediate

Author: ***Jacek Okrojek***

About the author:

Graduate of the Faculty of Technical Physics, Computer Science and Applied Mathematics at Technical University of Lodz, specializing in Network and Telecommunication Systems, tester, test leader, freelance developer, for more than 6 years involved in testing and developing software in the Ericpol Telecom and as an independent consultant, participated and supervised the tests at basic, functional, and system integration level , conducted training software testing,

contact: **jacek.okrojek@gmail.com**

Introduction

Testing applications, as we all know, is a time consuming task. We need to test various scenarios and input data. It is very common that many test scenarios are the same, and the only difference is the test data set. We can make our work more efficient in such cases with a Data-Driven Test (DDT) approach. In this issue, I will present this concept and demonstrate how it can be used with Selenium and Python.

Data-Driven Testing concept

Imagine you have a web application with user access control. Users can be assigned to different user groups and access different sets of functions. Your goal is to test the user authorization module. The Selenium IDE allows you to record actions needed for user authorization and create general test scenarios. You can export this test to Selenium's supported programming languages. You can copy and paste it for use in scenarios with different input data. You can also add assertions and check if the user can access required functions and system behaves correctly. This collection of tests can be run using Selenium RC. There are a few advantages of this approach, but, for me, only one is really important – the fact that you can run it during the automated integration test phase.

What happens if a new user group or new functions are introduced to the application? You will need to search all your tests and update them where required, or add new tests in the same way as earlier. This is repetitive, error-prone and inefficient. Your life would be much easier if you could separate the test data details from the code. Since the test scenario is always the same (logging in), it would be good to introduce one parameterized test method and execute it for different test data sets.

This approach is called Data-Driven Testing, so named because it describes the key concept: the separation of test data from test logic. It eases manipulation of test data, e.g. when updating or adding features to the application under test. The implementation of the test logic should be as flexible as possible. Test data can be stored in any desired format: text, CSV, spreadsheet files, or in a database.

As well as faster test implementation and easier test data maintenance, this approach gives us one more advantage: test data can be prepared by domain specialists, which is important when the system under test (SUT) has very complicated or highly specialized logic.

Problem generalization

The most interesting aspect of web applications, from a tester's perspective, are web pages with forms. They are usually the main way in which users pass data to the server. In general, we can describe testing of those pages as a three-step process:

- Opening the web page.
- Filling form fields and submitting form.
- Checking that answer page contain expected results.

Since we would like to have as general a framework as possible, we need to design it to be close to this general procedure. We are going to use a spreadsheet as a test data source, with the data organized as shown on Figure 1.

The first two columns are for administrative purposes and help us to identify which tests are failing or passing. The third column is a web-page URL.

The next columns are the user data (i.e. the data that is coming in via the form). The headers of these columns let us identify form fields. Depending on the situation, it could be a plain field name or its XPath designator.

Finally, we have the user action and the expected result. It is very important that these are the last two columns, as our code later will rely on this positioning.

By organizing data in this way, you are able to adjust the framework to the tested web page. You can add or remove columns, and therefore change the number of input fields to exercise. If we remove

ID	Title	URL	UID	Pass	Action	Result
1	Correct logging-admin	/auth	admin	admin	submit	Hello Admin
2	Correct logging-accountant	/auth	acc	acc	submit	Hello Acc
3	Correct logging-salesman	/auth	sal	sal	submit	Hello Sal
4	Incorrect logging	/auth	acc	sall	submit	Login failed

Tabela 1. Example spreadsheet.

all columns between the URL and Action column, we can also test formless web pages that have only links.

A typical implementation of a test procedure will look something like Listing 1.

```
1  sel = selenium(seleniumHost, seleniumPort, browserStartCommand, browserURL)
2  sel.start()
3  sel.open(...)
4   for k, v in actions.iteritems():
5       sel.type(k, v)
6  sel.click(...)
7         sel.wait_for_page_to_load(timeout)
8  results = result.split(";")
9  for r in results:
10     assert sel.is_text_present(...) == true
11  sel.stop()
```

Listing 1. Example implementation of data-driven test procedure.

Lines 1-3 open the web page to be tested in a web browser. Lines 4-5 write the test data to the form using the type method. The name of the form is stored in the k variable, taken from the first row of the table in Figure 1. In line 6, the click method is used to perform the required action, and the script then waits for a server response in Line 7. In the final lines 8-11, we are checking if the result page contains the expected text specified in our test data table. We can check for multiple text elements by separating them with the ";" sign.

Getting data

I recommend storing your test data in an Excel spreadsheet. The advantage of this approach is better visibility and organization of data. For example, you can store data for different web pages in separate tabs, helping you keep your data in order, and easing future maintenance. You can also use Excel to calculate expected results, in some cases, which saves you time.

To access the Excel data, we are going to use the Python library xlrd. In listing 2 below, you can find the part of the script which is responsible for this task. In this script, you open the file and specified by index tab, then, in a loop, we are collecting all data from all cells. Due to way we organize data, the values from the first row correspond to the header list, and values from the other rows correspond to actual data.

```
1  headers = []
2          book     =     open_workbook(xlfilename)
3     sheet   =   book.sheet_by_index(xlsheetIndex)
4   cols, rows = sheet.ncols, sheet.nrows
5  data = [[None] * cols for i in range(rows-1)]
6        for    row_index    in range(rows):
7            for  col_index  in range(cols):
8           if row_index == 0:
9         headers.append(sheet.cell(row_index,col_index).value)
10          else:
11           data[row_index-1][col_index]   =sheet.cell(row_index,col_index).value
Listing 2.
```

Test Parametrization

P Our test procedure should be executed for different data sets. If we use the unittest library and loops, you will end up with quite a serious inconvenience. If, during the test, you encounter an unexpected result, the whole process will be stopped. To continue with further tests, we would need to analyze the fault and resolve it.

By using py.test, we can overcome this problem. You can find sources and documentation at [2]. In py.test, all functions with names starting with "test_" are treated as test functions, and executed by the framework during testing. Let's put code from listing 1 into these test functions.

To prepare data for test framework requires implementation of py.test_generate_tests function. At the beginning, we will get data from the Excel file in Listing 2. Next, we are going to combine data with header into dictionaries. By executing metafunc.addcall(funcargs=dict(actions=d)) we are adding next data set. This data we can access in test_actions function by action variable.

```
1       def    pytest_generate_tests(metafunc):
2       ...
3       for r in range(rows-1):
4           d = {}
5           n = 2
6                    for  i  in headers[2:]:
7                d[i] = data[r][n]
8               n = n + 1
9     metafunc.addcall(funcargs = dict(actions = d))
10
11  def test_actions(actions):
12       ...
Listing 3.
```

Summary

Under http://coremag.eu/, you can find the complete code for the proposed solution. It has some limitations, so I suggest you modify it and adjust to your needs. The main aim of this article is to demonstrate how to implement DDT with Selenium. In next articles, I will present more widely applicable and complex solutions.

Acknowledgments

I would like to thank Mr. Justin Megawarne for proofreading and comments.

References

[1] Xlrd http://www.python-excel.org/
[2] Py.test http://codespeak.net/py/dist/test/index.html ■

Special gift from c0re and testerzy.pl

Our partner, testerzy.pl, created an application for All pairs generation.

The algorithm has been designed and tested by testers cooperating with testerzy.pl

The application is free for non-commercial usage and available for all users registered on forum.testerzy.pl and on c0re website.

To download the application: register on http://forum.testerzy.pl/or register / login on www.coremag.eu and check our Download section.

OpenSta – OpenSource for Web Load, HTTP Stress & Performance testing

intermediate

Author: ***Łukasz Smolarski***

About the author:

Łukasz Smolarski :
Graduated from Higher School of Business-National Louis University – faculty: "Computer Science" and Leon Koźminski Academy – faculty: "Management". During his studies he won a scholarship for leaders funded by GE Foundation and Institute of International Education. He currently works for Gtech Polska on the position of Quality Software Engineer as a Team Leader and person responsible for test automation. In 2007 he passed ISTQB Foundation Level, and in 2010 become AIS Certificated Specialist in HP Mercury Quality Center and Mercury QuickTestPro. Member of SJSI. **Contact: smolar2@op.pl**

Introduction

Every single day in our work we can notice that all types of tests are run repetitiously – which obviously is wasting a lot of valuable time. It's a hard work when every single scenario has to be repeated several times. There's a selection of free tools which can help in test automation process – in this article I will describe one of them. The tool named OpenSta is mainly designed for measuring performance testing, however it can also be used for other purposes, such as automation of certain actions performed in testing activities.

OpenSta is continuously developed free tool which can be downloaded from http://opensta.org website. Moreover, in case of problems or questions, we can use the forum for users of this tool and search or ask for help.

First steps with OpenSta

After installation, OpenSta Commander should be run. It's the main screen of the application and contains a tree, which is our repository and place where we can keep our scripts and tests.

The structure consists of three folders: Collectors, Scripts and Tests. We will focus on two folders. The first one is Scripts, where, using SCL programming language, we can create our scripts. The second one is Tests, where, using scripts created earlier, tests are created. Additionally, in the upper part of the application, there's a menu containing several useful options, as well as extensive Help.

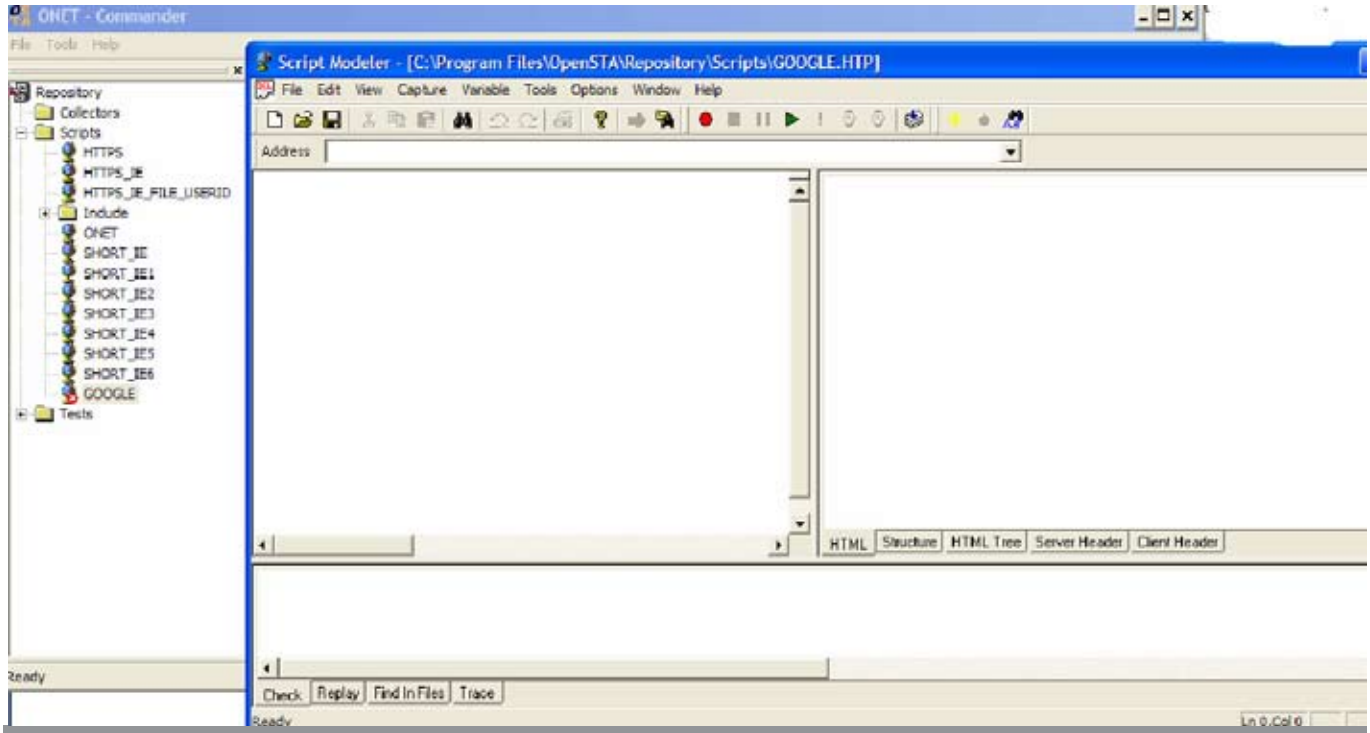
To create a new test, first we have to create new scripts – it can be made by choosing **File ->New Script** from top Menu. When we double click on the script, a recording window will open.

OpenSta supports both HTTP and HTTPS protocols what distinguishes it from other tools – like JMeter - and is one of its main advantages. If we are using Proxy server for internet connection, we have to properly configure the browser. In order to do it, click **Options -> Browser** from the menu (applies to IE 8).

If we want to connect to remote Server, it is possible by using settings from **Options->Gateway** menu. Unfortunately, you have to set up Proxy like shown on picture 3. In other case OpenSta will not be able to run.

Another function which is worth to mention, is the possibility to declare variables from **Variable ->Create menu.** It allows to prepare variables used for storing important values before recording script will take place. In order to start recording, click red button or choose **Capture->Record** from the menu. After that a browser will open and we can go through our planned test scenario. To finish recording, simply close the browser or click on the Stop button.

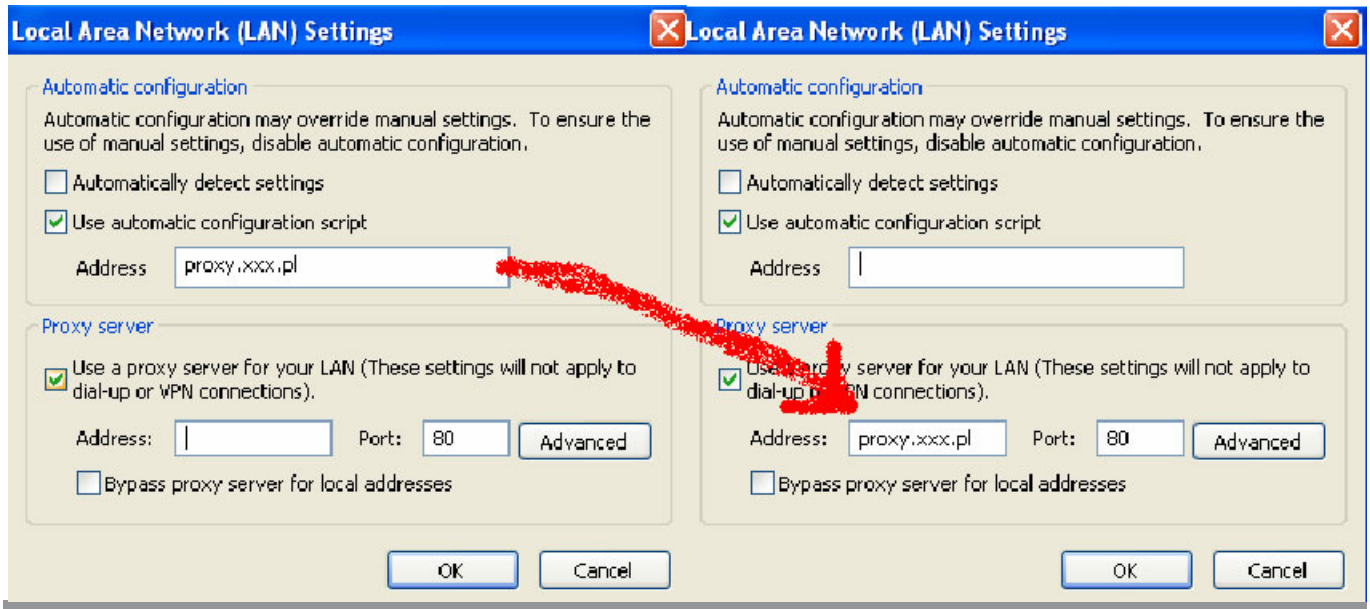
Picture 1 Main Screen of OpenSta



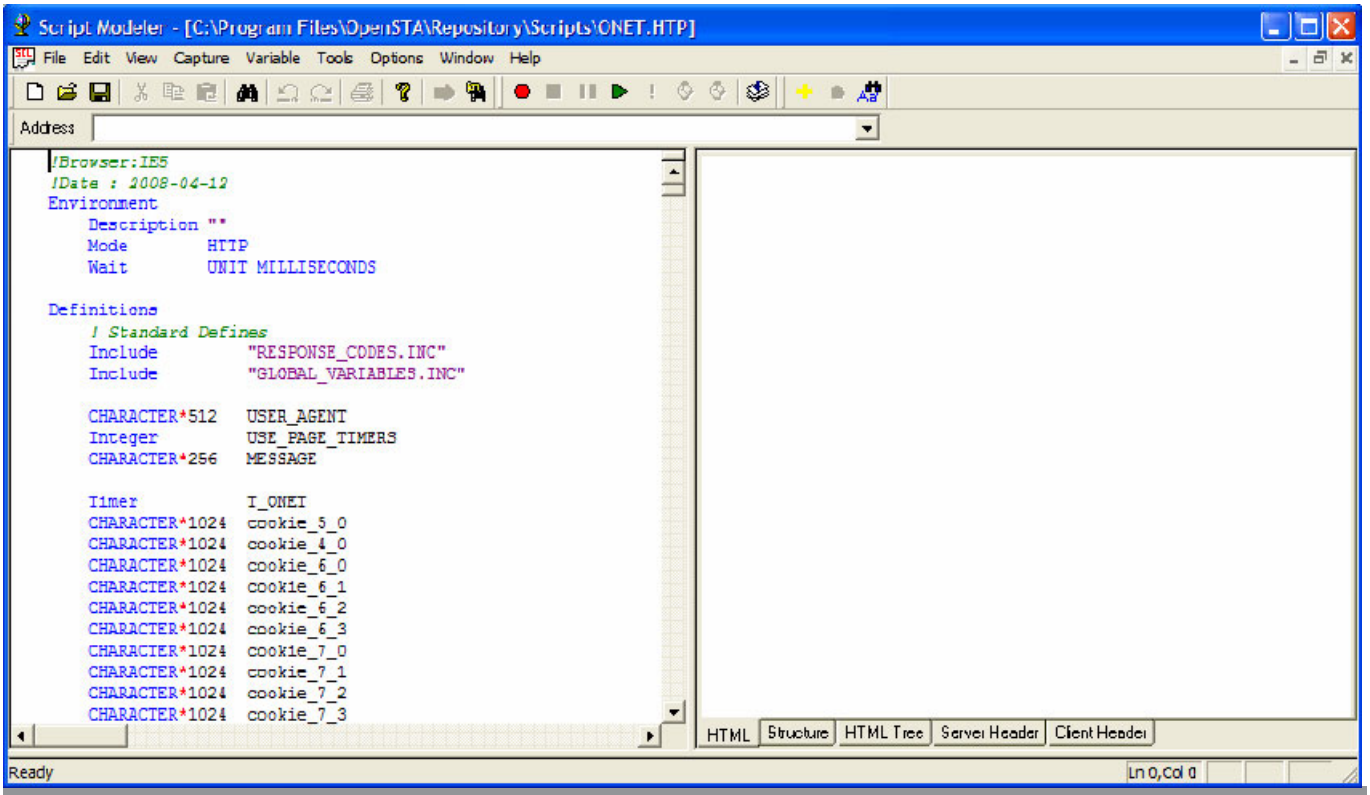
Picture 2. Creating Script in OpenSta

On the left side we can see the source code of recorded scenario, declared variables, environment information and other data captured by OpenSta. As it was mentioned before, source code is written in SCL (Structured Control Language). If any modifications need to be made after recording, it's possible by modifying the script. After finishing the code must be compiled, and, if there are no errors, we can run the script by clicking green **Play** button.

SCL language is not the simplest or easiest one, but if we take a closer look, we can notice some dependencies, such as adding data to variables. On the next picture we can see a piece of code which was modified in order to retrieve data from HTML generated by JavaScript – OpenSta has recorded it as a permanent value although it is generated dynamically. Running the script caused errors as the "old" value does not conform with the current one, generated by the page – hence it's necessary to put the value as variable.



Picture 3. Proxy configuration



Picture 4. Source code view in OpenSta commander

If the script contains no errors, the following message should be displayed:

After compilation, we can see „Get" function in the code – it can be highlighted using cursor – and when yellow arrow appears in top menu, click on it. Then the preview of recorded page will be shown.

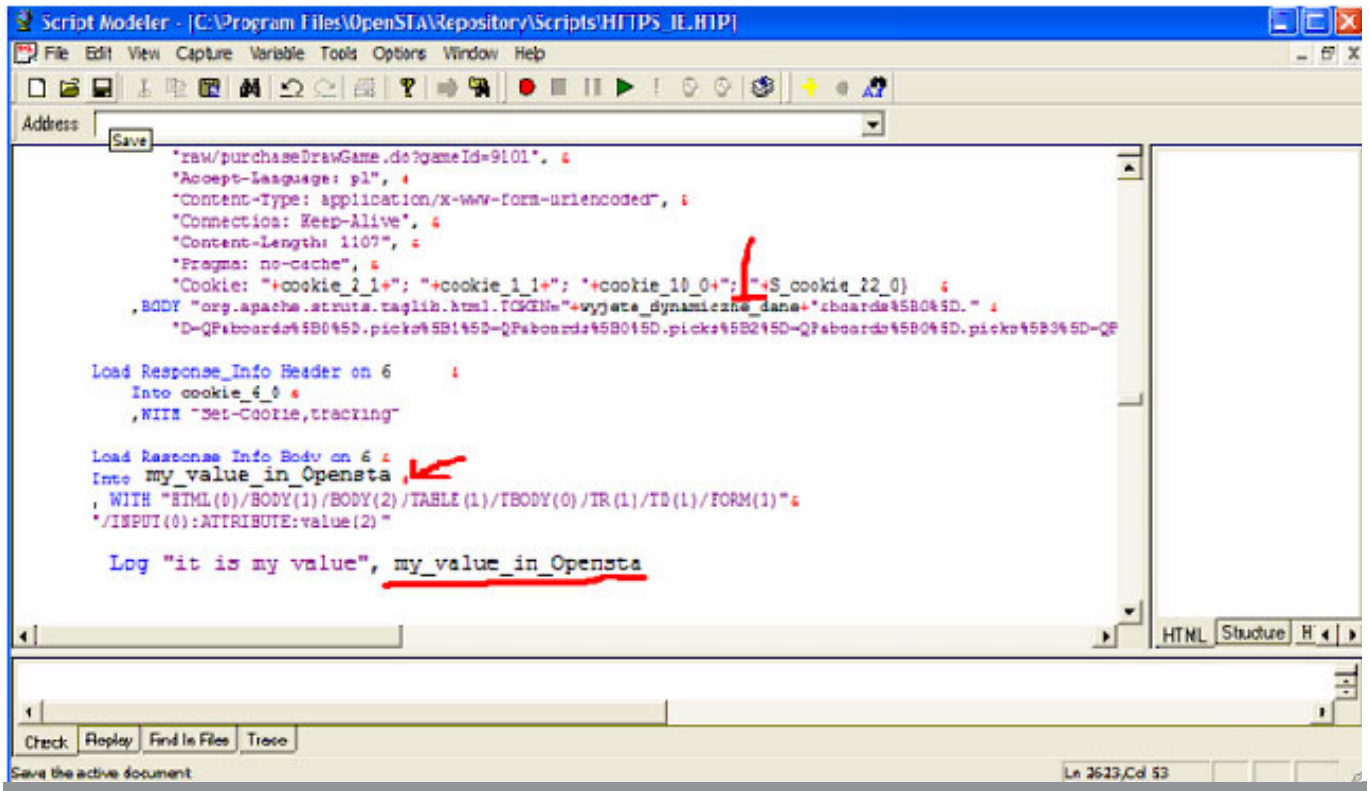
Now we can see HTML structure, server data and other useful information. We can also go to HTML code by right clicking on the interesting part of code and create variables, which can be used to retrieve data from DOM module (it can be used to retrieve information such as details created by JavaScript).

By clicking on structure tab, we can preview the page to see the entire structure and all related elements and their values.

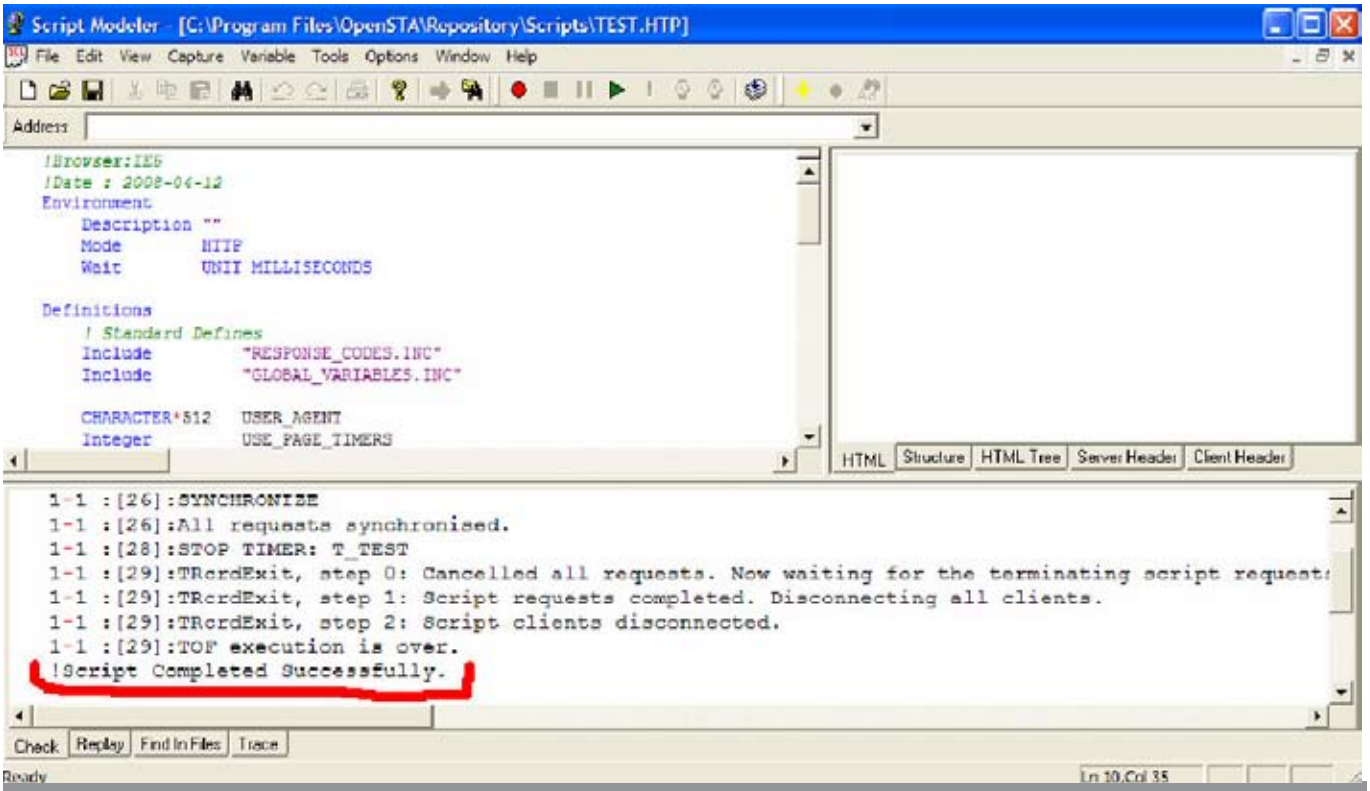
Additionally, by clicking HTML tab, we can find a specific value in the code, and after that, with right click, create variable containing this value and put it in the source code.

Now let's focus on test creation. Return to main page and choose File-> New Test-> Tests from the top menu. After that double click on the "Test" icon shown in the tree - testing menu will be shown.

Next, choose the script you have just created and drag it to the Task area. This area is split to columns, where we can add several scripts. It means that if we



Picture 5. Modyfing source code

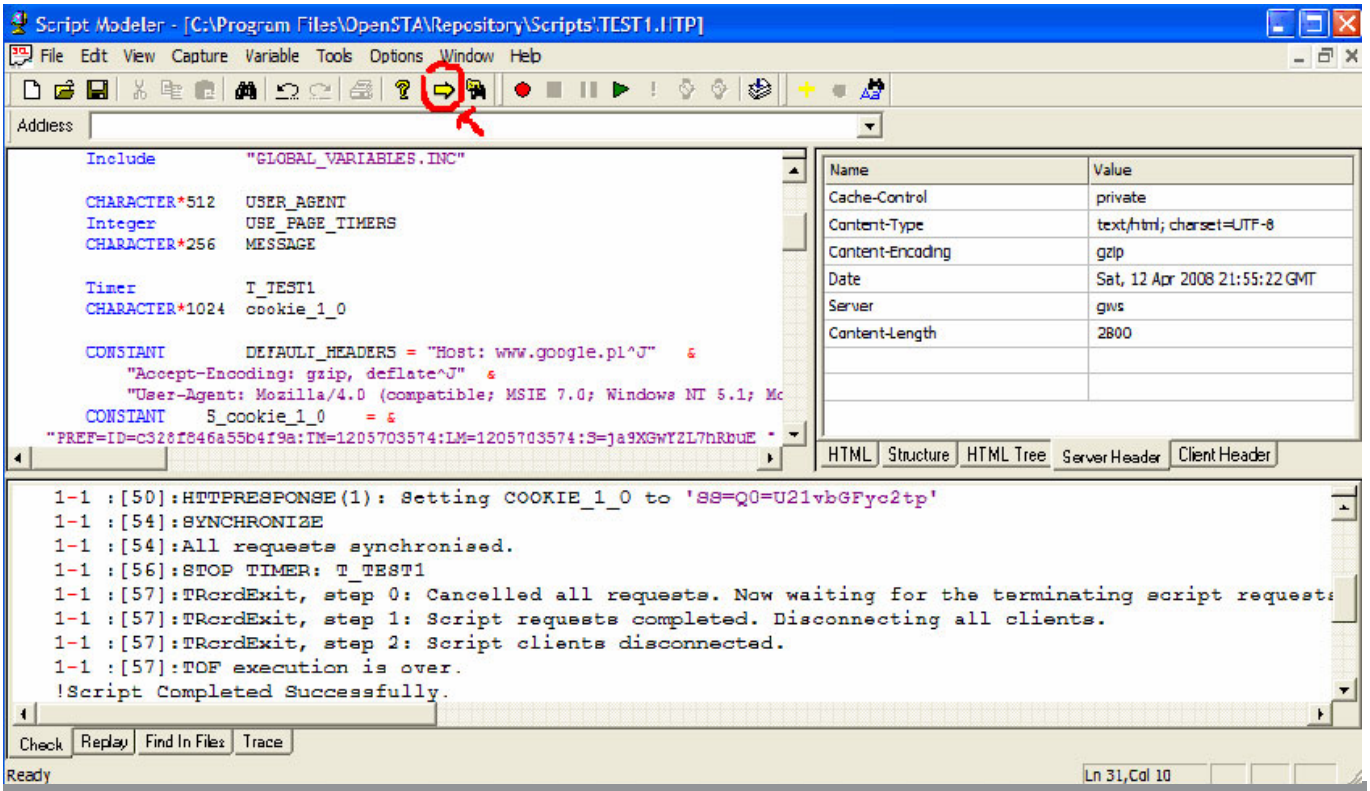


Picture 6. Compiling code

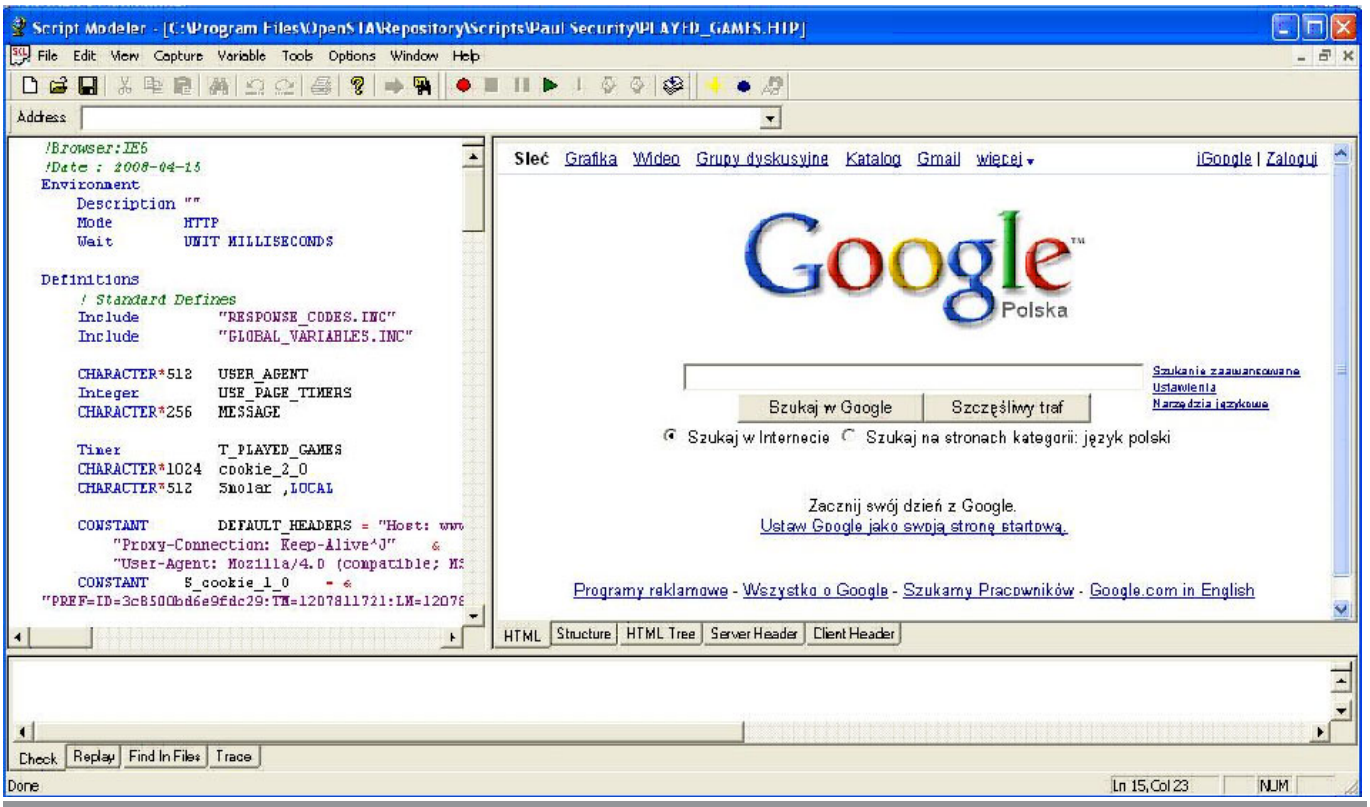
add some scripts to the same row, but to different columns, they all will be executed at the same time. If we add some scripts to different rows, they will be executed in certain order (Ascending).

In Test configuration there are some options, which can be controlled. Start option is used for setting begin time of executed tests (Immediate, delayed and planned). Next setting is the number of virtual users set for every single task. Moreover we can split users in categories, i.e. Total amount of VU =1200, but 2 of them are assigned to “Timer Results” and to “HTTP results” respectively.

There is also possibility to define users directly in the source code (using loop) – i.e. for the purpose of creating account via bank website and checking how long does it take to accomplish the task. We can check this data in some reports which will be described later. Now click on the Run button and the test will start.



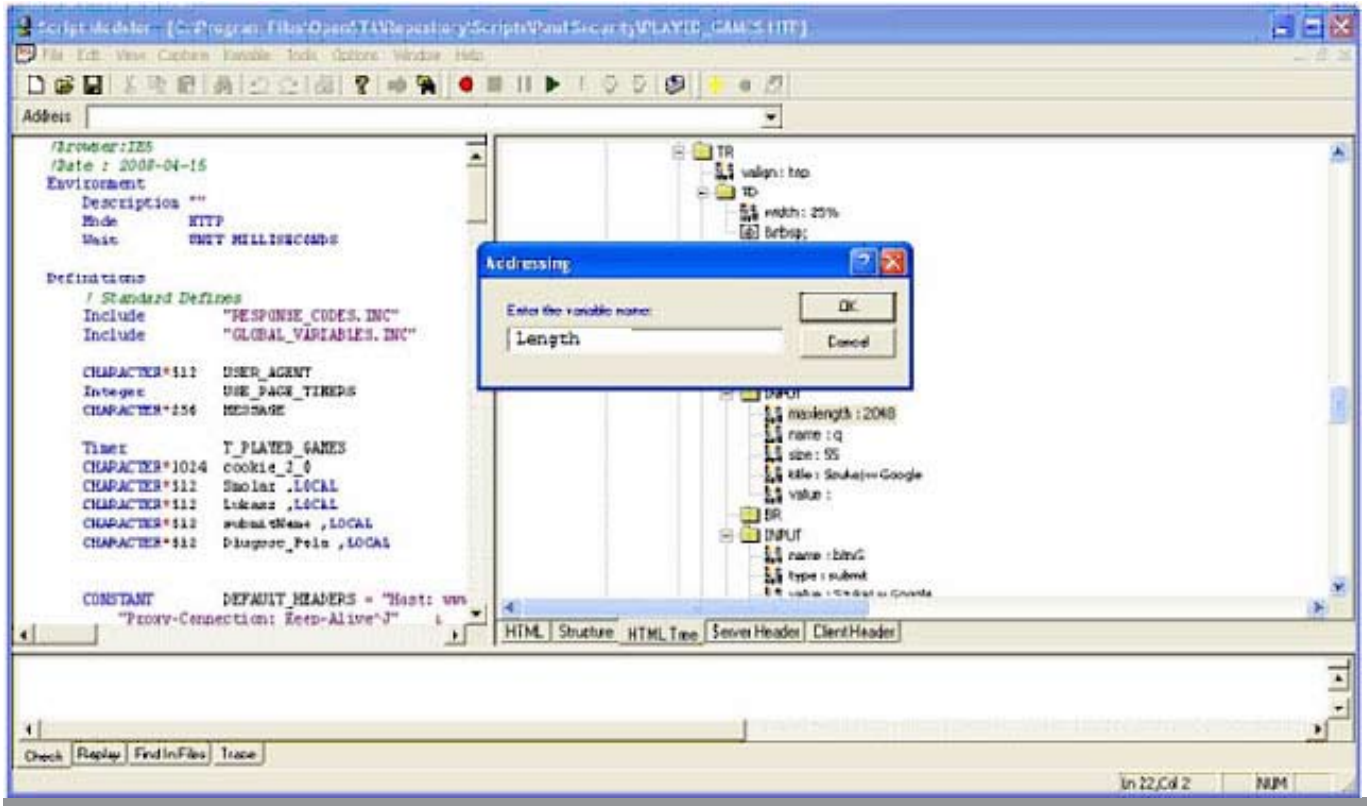
Picture 7. Get function



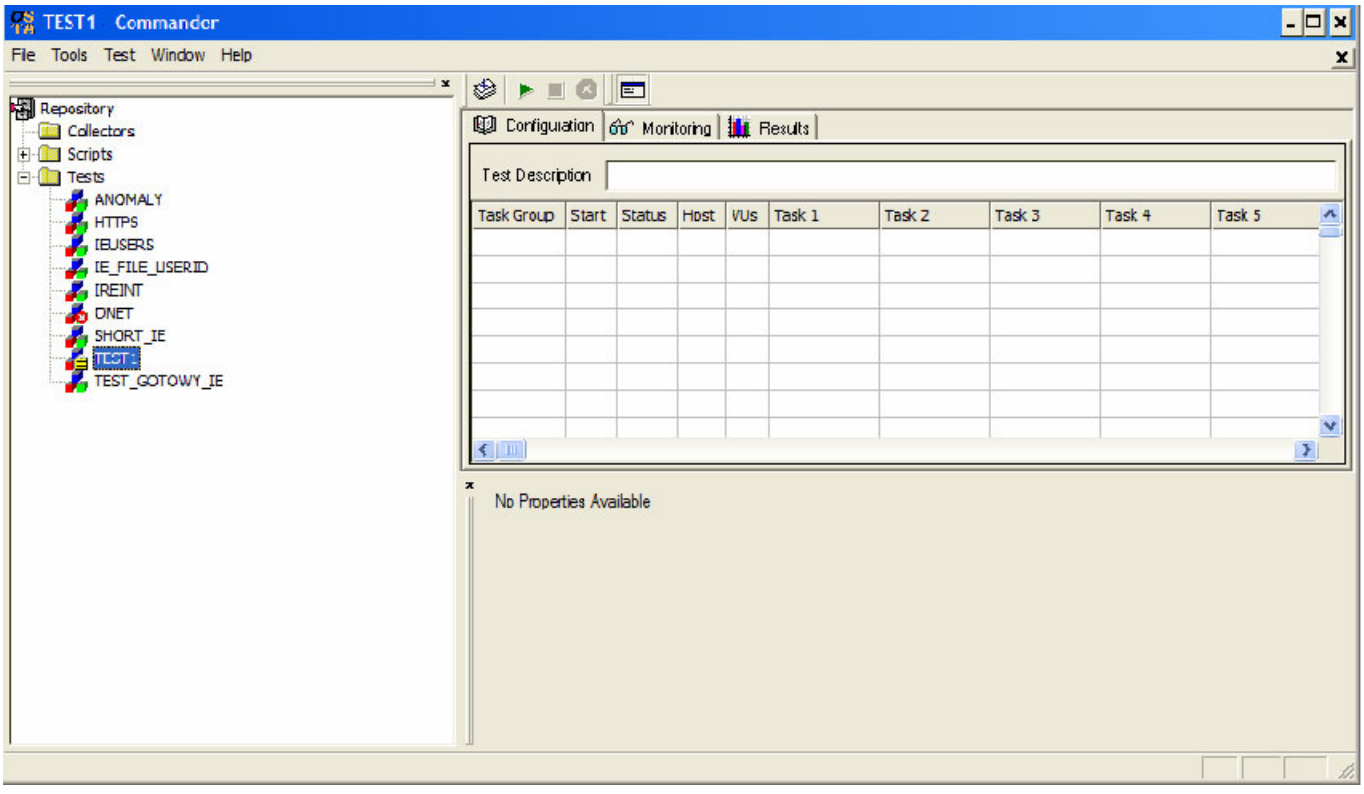
Picture 8. Page preview

During test execution we can monitor the progress and observe what is happening. This can be done via Monitoring tab. We can also check Summary tab and see how the test has been performed. (Unfortunately these reports are not user friendly since the information is not clear enough). In addition we can check whether any errors occurred during test execution.

The functionality described above is just a description of creating simple test scenario. My intention was to show only simple example of using OpenSta. As you know there can be complex test scripts created with many functions inside. Moreover, we can set up many configuration options and – as the result - tests will be run on different environments. During test execution we do not



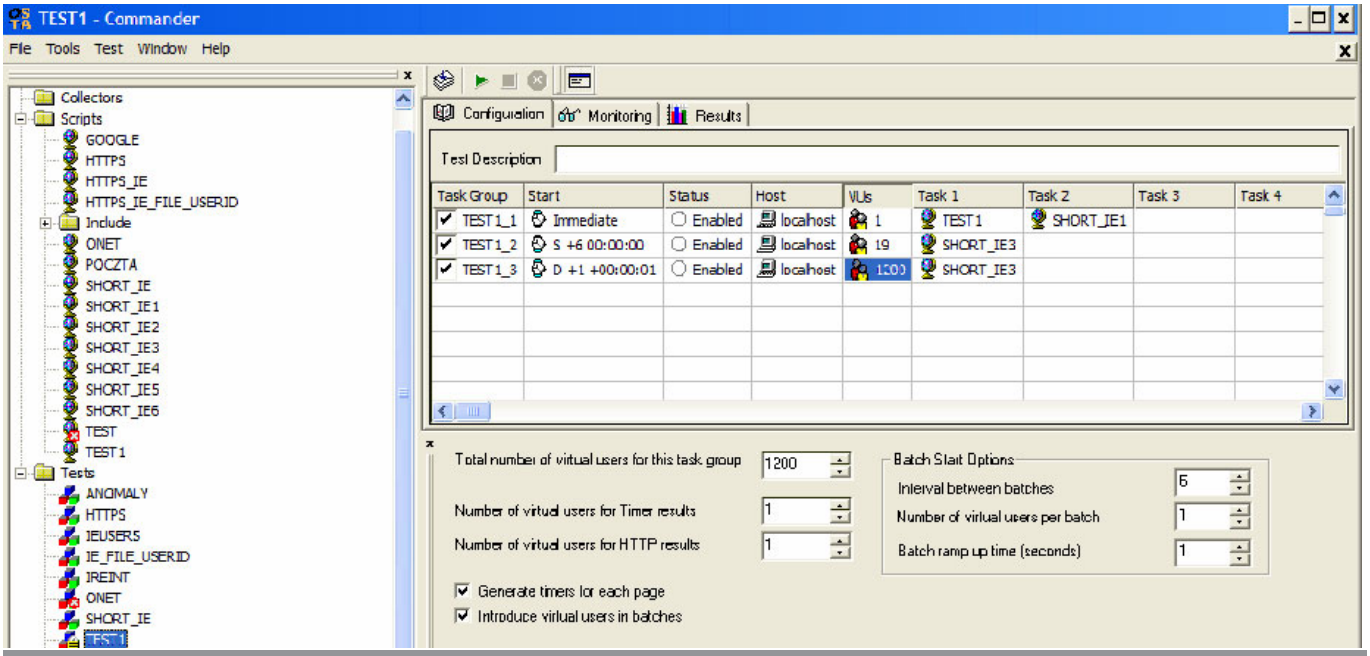
Picture 10. Creating variable with value taken from HTML tree



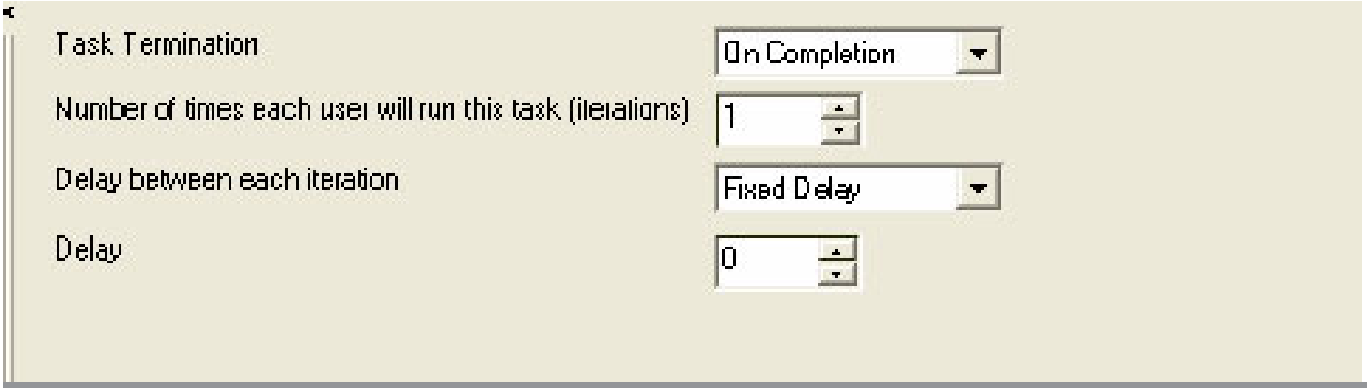
Picture 11. Testing menu

see how the script is going through web pages. All operations are executed in the background.

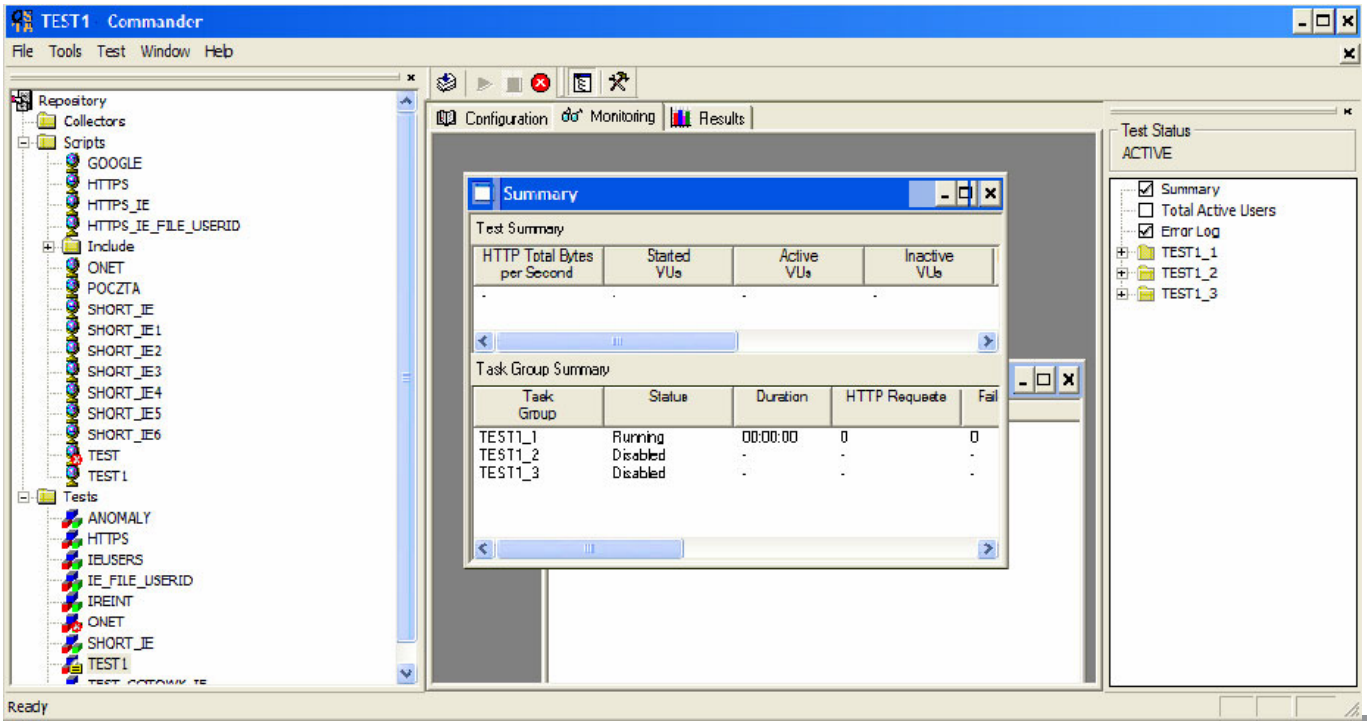
Like all other applications, OpenSta has also some disadvantages. I would like to describe some of them. One of them is the fact, that recording via HTTPS sometimes doesn't work - in this case we need to record on HTTP and modify source code to adjust it to HTTPS. This problem is caused by some errors in the OpenSta application and should be resolved in new releases. Another problem with OpenSta is that when we launch the browser, starting page sometimes does not appear. To solve this problem, proxy server needs to be set up again – exactly like shown in the pictures above. Another error which I've noticed is the problem with the length of characters while recording a script. Due



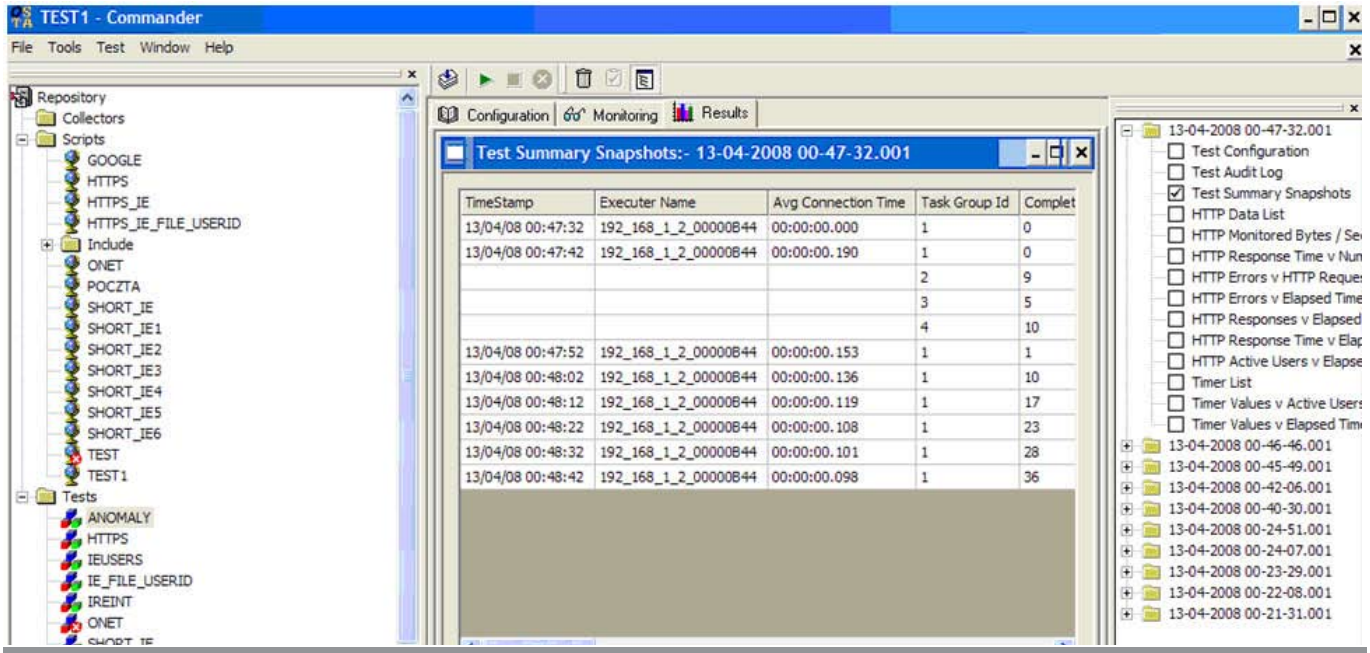
Picture 12. Test configuration



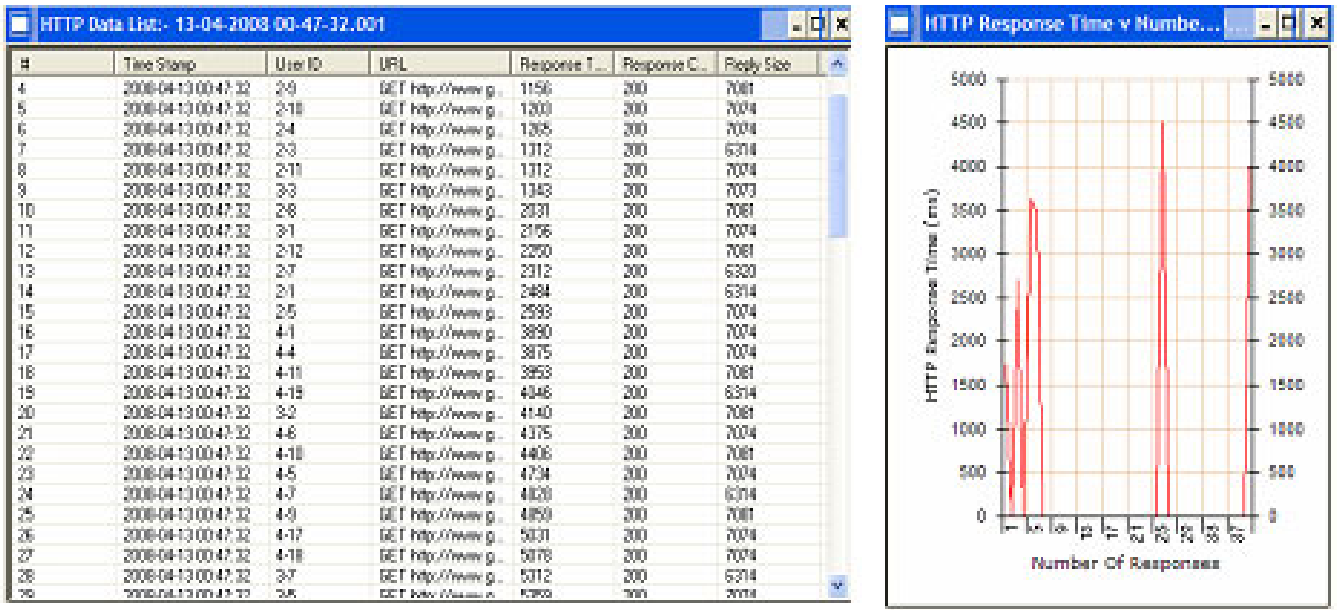
Picture 13. Task settings



Picture 14. Report visible during test execution



Picture 15. Results after test execution



Picture 16. Sample reports

to this fact compilation was failed. Fortunately, if any error occurs, we are able to see in which line of the code it happened . In my case I reduced the length of string and it started to work properly. (problem with browser , cut to IE7 – String “User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) ; .NET CLR 1.1.4322; InfoPath.2; MEGAUPLOAD 2.0; .NET CLR 2.0.50727)”).

Summary

Summarizing, I think that OpenSta it is a great free tool for performance testing. Despite some disadvantages, it has many useful functions allowing us to easily check web load, stress and performance of our application. Moreover this tool is continuously developed and maintained. It’s possible to join forum for OpenSta users and ask for help in case of any problems. I think nowadays it’s worth to check and try new tools available on the market as they can support our work and optimize it. If you’d like to get more knowledge about OpenSta, please refer to the following resources:

http://www.opensta.org/
http://portal.opensta.org/ ■

Requirements
Engineering
Qualifications Board

REQB
show your competence become certified!

Many problems in the field of Software Quality occur because of bad requirements. REQB is setting international standards for Requirements Engineering to change that!

www.reqb.org