*Hans van Loenhoud*
*Erik Runhaar*

# The relation between requirements and testing in Agile projects

## Introduction

In Agile Scrum projects, requirements are documented in user stories with their related acceptance criteria. The user stories are initially developed by a product owner, put on a product backlog, and then selected for further refinement and elaboration in individual iterations.

In practice, we see that these initial user stories are concentrating on high-level functionality. Acceptance criteria add some details and non-functionals to it, but often do not provide a sufficient basis for testing the software. A user story mostly focusses on the functionality of an individual item of software; end-to-end integration of the item into a working system-of-systems seldom receives sufficent attention.

(More) Involvement of the testing expertise in the creation and elaboration of the user stories will improve their testability and thereby mitigate business and integration risks.

## Requirements

Requirements can be seen as the substantiation of demands, wishes, expectations of the business that desires some support for their operation from a software system. In Agile projects, it is usually the product owner, being a representative of the business stakeholders, that collects these demands and consolidates each of them into a single statement: the user story. The format of the user story itself entails a, often too specific, focus on the functional requirements: **'As** [a person in a certain role], **I want to** [perform a certain action / obtain a certain result] **so that I** [reach a certain goal / get a certain benefit].' It is about what the system is supposed to do, for instance, 'As a sales manager, I want to print an invoice for the product sold, so that I can collect money from my customer.' The complete set of user stories is set up to describe the functionality of the system from a business perspective.

As soon as it is established what the system should do, the focus shifts to how the system should do it, primarily the non-functional requirements. These non-functionals are documented in the acceptance criteria for the user story. A common format is **'Given** [certain preconditions], **when** [a certain action is carried out], **then** [a particular set of observable consequences should obtain].' For instance, 'Given that I sold a product, when I enter the sale in the system, then the invoice is available within 15 seconds.' [performance].

A third type of requirements that must be taken into account when developing a system are the constraints. Constraints are mostly technical in nature and limit the solution space within which the system is to be developed. It may be about the infrastructure on which the system is to be implemented ('The system will work on iOS and Android devices'), or about the architecture ('The system must fit into the existing IT-landscape'). Sometime constraints pertain to legal issues, industry standards, or cultural aspects.

In Agile projects, the initial attention goes to user stories ('epics') on large chunks of functionality. They arise from direct contacts between the product owner and his immediately surrounding business stakeholders, and are collected on the product backlog at the start of a project. In release and sprint planning, and in grooming sessions, sprint teams frequently discover that these kinds of user stories are too high-level to be realized in a single sprint. They have to be split up into more detailed user stories, containing smaller pieces of functionality, before they can be added to the sprint backlog of a certain sprint. By the nature of their specific format, user stories tend to concentrate on functional solution aspects. Acceptance criteria mostly regard additional detailed functionality or obvious non-functional requirements from a direct business view; constraints tend to be overlooked, underestimated or taken for granted.

As a consequence, an Agile project usually starts with an imperfect and volatile set of core requirements, which is gradually detailed and upgraded during the course of the project as more information becomes available in the sprints. User stories are refined to describe them in more detail and new acceptance criteria are added on non-functional requirements and constraints. This is the consequence of intensified contacts by the whole sprint team with a broader group of stakeholders (exceeding the direct business stakeholders) and the feedback received from them. Another common feature is that inconsistencies and conflicts between requirements and between stakeholders come to the surface, which first have to be resolved before a certain piece of software can be developed. In addition, the integration of new parts into the existing system may lead to the discovery of new requirements. Taking it all together, during every sprint the team learns more about the system in

an exploratory way and will discover necessary additions and changes on requirements; new user stories and acceptance criteria are added to the product backlog and the sprint backlog of the current sprint is de-scoped to account for these changes.

Agility in its very nature is to be open for changes. The Agile manifesto explicitly welcomes them during a project, as it allows the team to fine-tune the system to the ever-changing environment instead of developing a system on a fixed situation at the start of it. However, change and instability during a single sprint will threaten the success of working software at the end and reduce the velocity of the team. Therefore, the quality (complete, clear, consistent, agreed, …) of user stories and acceptance criteria should be assured before they can be selected for elaboration in a certain sprint, at least to a level that allows for concrete planning and task definition.

## Testing

The requirements defined in user stories and acceptance criteria serve as the basis for design, coding and integration of the software to be developed and the user processes to be supported. At the same time, they are used to develop test cases for the verification and validation of the software and for preparing the product demo.

Ultimately, testing is about gathering information on the quality of a software system. Testing concentrates on two quality aspects: 'Is the system built right?' (conformance to specifications) and 'Is the right system built?' (fitness for use). Testers use the requirements as an input to investigate both questions and execute their tests for providing the answers. With these answers, the business can mitigate risks concerning the actual use of the software in production.

More detailed and more elaborated requirements permit the tester to develop and execute tests that provide better information on the quality and more accurate risk mitigation for the stakeholders. In an ideal world, the starting point for the tester would be a single complete and consistent set of requirements, at the same level of detail and abstraction, consisting of user stories with the functional requirements and related acceptance criteria with additional details on functionality, non-functional requirements and constraints. In combination with a risk assessment, the tester then can prioritize and develop test conditions and test cases to investigate the relevant quality aspects of the system.

In Agile projects, documentation is lean ('just enough') and time is short. If, at the beginning of a sprint, requirements are incomplete, unclear, inconsistent or not agreed between stakeholders, testing may be unable to provide enough information on the quality of the software in time. Sprint teams try to avoid this by repeated grooming the product backlog to improve the quality and by carefully selecting the user stories for the next sprint during the sprint planning phase, but might fail in doing so. If testing starts from the assumption of high quality user stories, defects in the software will be discovered during the sprint, but flaws in the users stories may be overlooked.

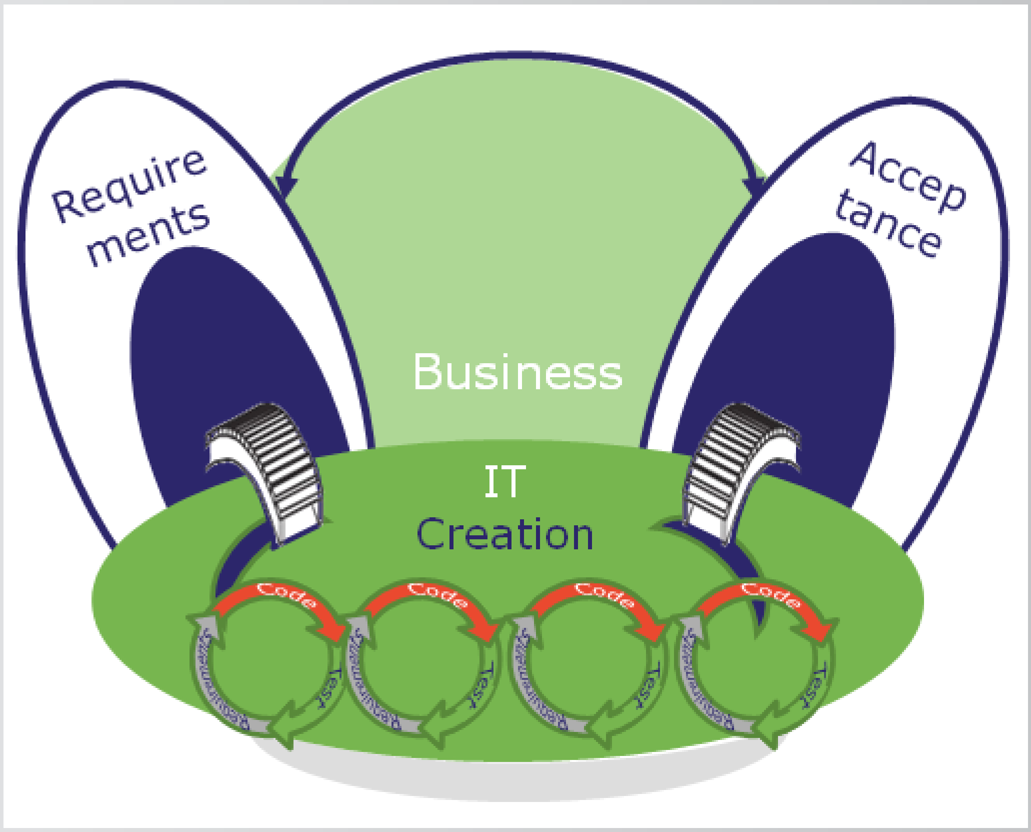
The same user stories and acceptance criteria serve as an input for the developers who build the software. They are confronted with the same flaws in the user stories, but they are in a position to 'fill in the gaps' by adding their own interpretation of detailed requirements. From the focus of the developers on delivering working software, this kind of additional requirements often relate to detailed functionality for end users and to technical constraints. When sprint teams work from the idea of a fixed sprint backlog, these additions will not be made explicit in updates of user stories and acceptance criteria, and are easily missed in testing.

## The "Frog" model

In order to describe te relationship between Requirements, Creation and Acceptance, we developed the 'Frog'-model to illustrate the development life cycle in an Agile context.

At the left hand side, we discern the Requirements part, in which a set of users stories and acceptance criteria is established and collected on the product backlog, including a requirements setup and prioritisation of the overall user stories in an end-to-end or Release theme. The definition of these requirements on all levels, and the tracking and tracing of it, is the responsibility of the business, represented by the product owner. In large organizations this will be a challenge, often assigned to a team of architects.

The right hand side is the Acceptance part, where the business decides on the use of the developed software. Once again, the business is reponsible for it, with the prod-



uct owner in the role of representative of the sprint team, 'selling' the solution. Testers facilitate this part, as the business decision will heavily rely on quality information provided by them. In every sprint, there is a demo, in which the product owner is responsible for the acceptance of the final result of the sprint at hand. Apart from and exceeding that, a group of users may do their own acceptance itself, in the form of a test activity, showing that the delivered system adeqately supports their work. This might be done as part of the iteration, but that is not always possible, for instance in the case of purchased standard software. Paramount, the end-to-end and overall non functional issues, like performance, security, et cetera, are best tested in a more stable, overall end-to-end / system-of-systems environment. This is what testers should facilitate and draw the product owner's and architect's attention to.

At the bottom of the model, the Creation part is about the (technical) realization of the software, based on the requirements. This is a sequence of Agile iterations (or sets of iterations, SCRUM-of-SCRUMS) that lead to working software products that can be demo-ed. This creation activity is the responsibility of IT as delivery. Since each sprint team is responsible for their own iteration result, the final solution is usually system integration- or acceptance tested as a separate activity. In many organizations this is an unexplored part of system development.

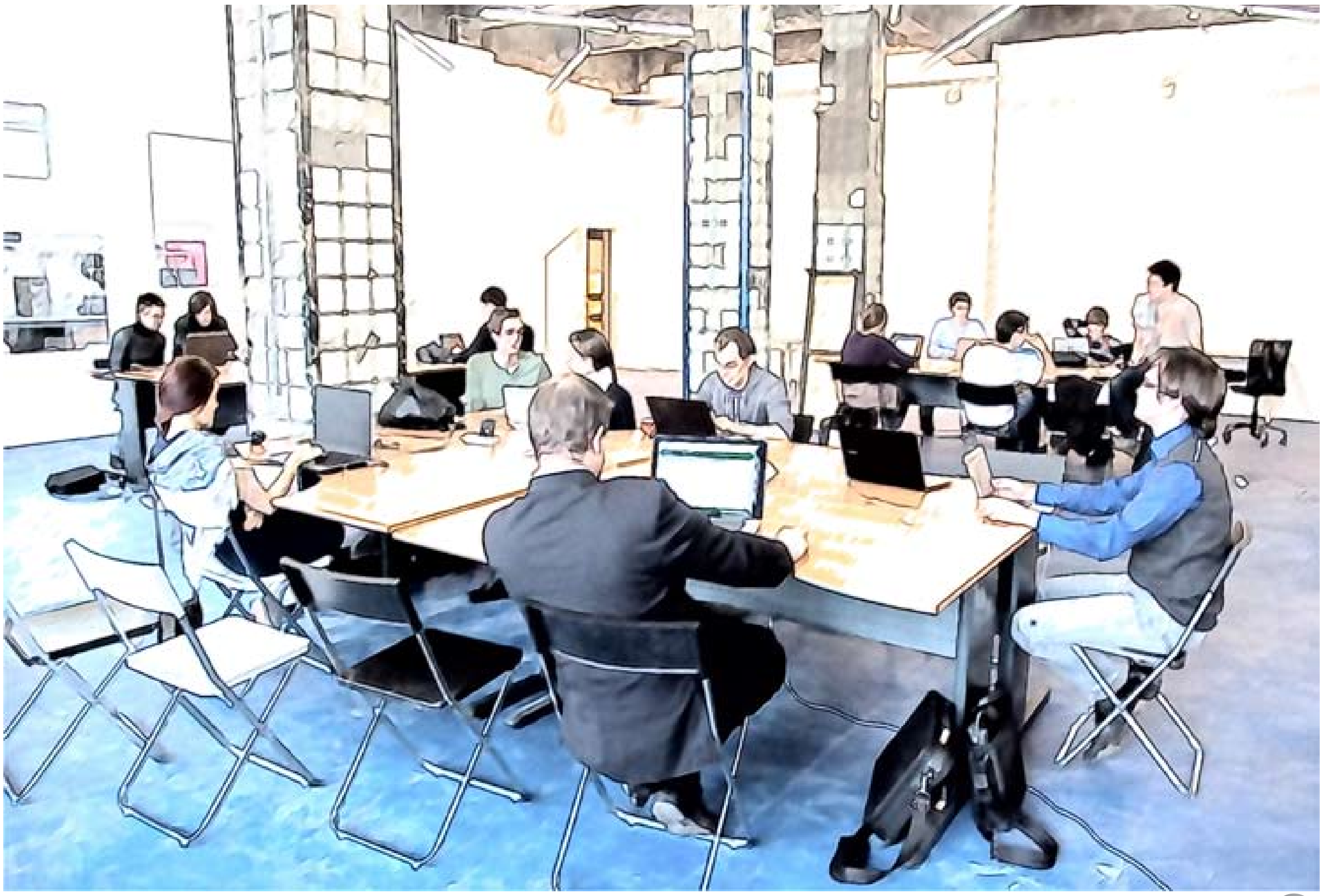
## Tester's involvement

In many Agile projects, team members with a testing background serve as the quality conscience of the team. Their geatest value lies in their broad and independent view on the quality of the system as a whole and their ability to demonstrate this quality or the lack of it.

In the Creation part of the Frog-model, testers participate in sprint teams, supporting the team in component and integration tests, and developing and executing system and regression tests. In the Acceptance part, they support and guide end users in acceptance tests, develop and execute end-to-end tests, participate in preproduction tests, et cetera. This may be done as part of the sprints, or as a separate track apart from these.

Since the initial requirement setup in the creation of user stories is done by the product owner, the involvement of testers in the Requirements part is usually limited. In subsequent grooming, defining and redefining during the Creation part, testers do participate, but as stated before, the requirements at that stage are heavily focused on functional aspects of the chosen solution. The quality of the user stories and acceptance criteria at the start of a sprint then will be suboptimal, which is inherent to the nature of agile and may entail the issues mentioned.

Earlier involvement of testers in the Requirements part will assure the quality level of users stories and acceptance criteria as an input for the Creation part, thus enabling IT to efficiently develop the software without unnecessary disturbances underway, and guaranteeing a smooth acceptance.



Testers can contribute to the Requirements part in several ways.

- Requirements sources

Testers have contacts with a much broader circle of stakeholders than the product owner does. While the product owner has a focus on direct business contacts, testers will identify additional requirements sources from IT, competitors, customers, governmental organizations, adjacent systems, legislation, et cetera.

- Level of detail and abstraction

Testers will recognize differences in level of detail and abstraction within a set of user stories. They are able to propose a suitable hierarchy.

- Consistency and agreement

Testers can identify gaps, overlaps and inconsistencies within a set of user stories, and may notice (hidden) conflicts between stakeholders and within requirements, that must be resolved before they can be realized in one and the same system. They can help to harmonize a collection of requirements from different sources into in single consistent set.

- Non-functionals and constraints

Testers will pay proper attention to a broad pallet of non-functional quality characteristics and constraints, leading to a complete set of detailed users stories and acceptance criteria at the start of the Creation part.

- Testability

In their own interest, testers will check user stories and acceptance criteria for testability. Good testability will make it easy to demonstrate the quality of the delivered software.

## Conclusion

The Requirements part is the most challenging part of Agile projects. Usually, the requirements are derived by a business representative in the role of product owner and collected on a product backlog in the form of user stories. Biased focus and lack of requirements engineering skills may results in an initial backlog with flawed user stories. Typically, such user stories concentrate on single sprint functionality and neglect non-functionals, non-technical constraints and end-to-end integration issues. This causes issues and delays during the subsequent (technical) Creation part and controverses in the Acceptance part.
Agile projects will benefit from the involvement of testers right from the start in assuring the quality of the requirements. This was one of the critical success factors in the original Waterfall model. It is still important in the Agile situation, especially when working in an Agile way on large projects. In an independent role, testers can act as a bridge between business and IT, and between business stakeholders and other concerned parties, improving communication and assuring overall quality.
An experienced tester with sufficient knowledge of requirements engineering can support the development and growth of a complete set of clear, consistent, and agreed requirements that serves efficient development of effective IT systems, thus guaranteeing the cohesion between the Requirements, Creation and the Acceptance parts of the Frog-model.

AUTHOR

**Hans van Loenhoud**

As a senior consultant at Polteq, the Netherlands, Hans van Loenhoud provides test consultancy to customers all over the world and acts as a trainer in many courses on software testing, especially the courses of the ISTQB. He wrote several books and articles on software testing and presented at testing conferences in the Netherlands and abroad.

Hans is also engaged in requirements engineering, as requirements are the starting point for all testing activities. He developed a Foundation Level course based on the IREB body of knowledge and helped dozens of students to pass the CPRE exam. Hans is chairing the international workgroup for the IREB Advanced Level Elicitation and Consolidation syllabus.

**Erik Runhaar**

Erik Runhaar is working on testing since 1996 in different types of organizations, ranging from large multinationals in ERP to smaller companies in health care and health insurance in the Netherlands.

Erik is a senior consultant at Polteq. He is accredited trainer for ISTQB Foundation (including Agile Tester Add-on) and Advanced level courses as well as for the iSQI Certified Agile Tester. Erik is also involved in development projects, and in coaching and mentoring people in testing activities. Erik shared his ideas on testing in articles ranging from Test strategy to Agile testing issues in several testing papers.

*Seretta Gamba*
*Dorothy Graham*

# Process Patterns in Test Automation

## Introduction

It's now a couple of years that we have been collecting test automation patterns in a wiki [testautomationpatterns.wikispaces.com. The wiki is read only. To contribute please ask for an invitation.]. We have classified them as process, management, design or execution patterns. We have also given quite a few tutorials about them at various conferences. Generally we end up speaking only about management or design issues or patterns. Execution issues are usually a side effect of poor automation design, so in this paper we want to take a closer look at process issues and patterns.

## Process patterns

Process issues are often underestimated: it is often difficult to recognize them from the inside of a project, particularly if the company culture does not support test automation or processes that would help automation. Test automation works best as a team effort with testers, developers and automators working hand in hand. A typical process issue is missing or poor communication between different departments (INADEQUATE COMMUNICATION). Other problems arise when nobody cares about data or script reuse (DATA CREEP, SCRIPT CREEP), documentation (INADEQUATE DOCUMENTATION), revision control (INADEQUATE REVISION CONTROL) and so on. These kinds of issues can kill even a well-designed and well managed automation effort.

Starting with INADEQUATE COMMUNICATION, let's examine the most important issues more in depth.

**Issue Summary**
This issue covers two frequently recurring problems:

- Testers don't know what automation could deliver and the test automation team doesn't know what testers need
- Developers don't understand, don't know or don't care about the effect of their changes on the automation

**Category**
Process

**Examples**
1. Test cases that should be automated are written very sparingly because „everybody knows what you have to do"... only automators do not
2. Automators need help from some tester or specialist, but that person is not available or doesn't have time
3. Testers do a lot of preparations to do manual testing that could be easily automated if only the automators knew about it
4. Testers, developers and automators work in different buildings, cities, time zones, or countries
5. Developers change the Software Under Test (SUT) without caring if it disrupts the automation or makes it harder

**Questions**
- Are testers and automators on the same team? If not, why not?
- Do developers notify automators when they want to use new components?
- Do automators report to development which components they cannot drive?
- How often do team members meet personally? How often in telephone conferences / live meetings?
- Do team members know each other? How about time or language differences?
- Do team members with the same role have the same experience / know-how? Do they speak the same „language"?

**Resolving Patterns**
Most recommended:

- SHARE INFORMATION: this pattern is a no brainer for big and small automation efforts. Use it!
- WHOLE TEAM APPROACH: if your development team uses an agile process and you apply this pattern, you will not encounter this issue

Other useful patterns:
- GET ON THE CLOUD: This pattern is especially useful if you are working with a distributed team

**Pattern Summary**
Testers, coders and other roles work together on one team to develop test automation along with production code.

**Category**
Process

**Context**
This pattern is most appropriate in agile development, but is effective in many other contexts as well. This pattern is not appropriate if your team consists of just you.

**Description**
Everyone on the development team collaborates to do test automation along with production code. Testers know what tests to specify, coders help write maintainable automated tests. Other roles on team also contribute, e.g., DBAs, system administrators.

**Implementation**
If you are doing agile development, you should already have a whole-team approach in place for software development, testing and test automation.
If you are not doing agile, it is still very helpful to get a team together from a number of disciplines to work on the automation. In this way you will get the benefit of a wider pool of knowledge (SHARE INFORMATION) which will make the automation better, and you will also have people from different areas of the organisation who understand the automation.

**Potential problems**
If people are not working on the automation as a FULL TIME JOB, there may be problems as other priorities may take their time away from the automation effort.

If your developers are using an agile development process the pattern to apply to solve such a problem is definitely WHOLE TEAM APPROACH.

Developers should already be writing automated unit tests, so they should also be open to help automate the system tests. Also being on the same team will spare you problems like when developers change something that disrupts the tests and you find out only when your automated tests suddenly all fail. Another advantage is that you can find out at a very early development stage if some used component is not supported by your automation tools. In this case you will be able to find some solution:

- You convince the developers to change the component
- You find a new tool that can support it
- Together you find a way to work around the problem so the automation can use the component

The pattern SHARE INFORMATION is good in any type of context.

## SHARE INFORMATION

**Pattern Summary**
Ask for and give information to managers, developers, other testers and customers.

**Category**
Process

**Context**
This pattern is appropriate when you have to communicate with management, testers or developers, and when you have new people coming onto the team.
This pattern is not appropriate when you are working alone on issues that you have already mastered completely.

**Description**
There are many people who are involved with test automation, and they have different needs for what they need to know. But they won't know about things unless they are told, so you need to share relevant information with them at appropriate times.

**Implementation**
Some suggestions:

- Keep management informed on the progress of the test automation project. Find out what metrics they need, explain which can be easily collected and which not, and provide regular overviews in a format that is most appropriate for them
- Have managers tell you what they specifically expect from test automation. In this way you can notice quickly if they have UNREALISTIC EXPECTATIONS and can inform them accordingly
- Speak with other people about what you are doing: explaining something often leads to new ideas, yours or the people you are talking with
- ASK FOR HELP when you have a problem or a question: you should never ponder too long on some issue, other people may have already solved just the same question
- Listen to testers or developers. Ask why they do something and why they do it as they do. If you find out what they really need, you can support them even better than you were planning
- Ask developers to keep you informed when they make changes to the Software Under Test (SUT) that affect test automation
- After you have obtained some concrete results CELEBRATE SUCCESS
- Speak also about your failures: people will be thankful if in that way they can LEARN FROM MISTAKES

Communication also includes reports, demonstrations, Wikis, billboards etc. Use what is best known in your company.

**Potential problems**
Communication can easily be misinterpreted, especially emails.

Communication needs to be at the right level for the recipient and tailored for the audience, or it will be ignored or worse.

Actually the pattern SHARE INFORMATION is not only valid for test automation! It would be useful in a pure development or exploratory test context. It would give also good value at Christmas time with your family!

Now let's explore some other important process issues, DATA CREEP and SCRIPT CREEP. The amount of data or scripts keeps growing mainly because instead of reusing them people create new ones all the time. Note that the problem here is not the amount of data or scripts, it's not reusing them and writing doubles instead!

## DATA CREEP

**Issue Summary**
There are countless data files with different names but identical or almost identical content

**Category**
Process

**Examples**
1. Nobody knows what is being used and where, so nobody wants to be responsible for deleting eventually needed data
2. To edit or remove the data files is too much work: one would have to look up all the places where they are used and change the referrals. If files are similar rather than identical, a unified file would have to be created

**Questions**
- Is the data documented?
- Are there standards regarding naming and documentation?
- Who creates the data? How? Who uses it?

**Resolving Patterns**
Most recommended:

- GOOD PROGRAMMING PRACTICES
- MAINTAINABLE TESTWARE
- MAINTAIN THE TESTWARE
- MANAGEMENT SUPPORT: You will need this pattern to be able to change the current bad behaviour
- REFACTOR THE TESTWARE

You should already be applying these patterns. If not, do it!

Other useful patterns:
- GOOD DEVELOPMENT PROCESS: apply this pattern if you don’t have a process for developing test automation. Apply it also if your process lives only on paper (nobody cares)
- LEARN FROM MISTAKES: apply this pattern to turn mistakes into useful experiences
- KILL THE ZOMBIES: Apply this pattern for a start
- DEFAULT DATA: use this pattern if your tests use a lot of common data that is not relevant to the specific test case
- DOCUMENT THE TESTWARE: you should be already applying this pattern. Retro fixing documentation is quite an effort. Do it in the future for all new projects and every time you have to update something old
- KEEP IT SIMPLE: Always apply this pattern!

**Issue Summary**
There are too many scripts and it is not clear if they are still in use or not.

**Category**
Process

**Examples**
1. It takes so much time to check if a script is already available that testers or automators would rather write a new one instead. This means that there are a lot of very similar scripts.
2. Nobody „refactors" the scripts so that after a time some fail consistently and are not executed any longer.
3. It isn't possible to check which scripts are actually in use.

**Questions**
- How are scripts documented?
- Are there standards regarding naming and documentation?
- Who writes the scripts? Who uses them?
- Is anyone charged with reviewing the relevance and usefulness of the scripts at regular intervals?

**Resolving Patterns**
Most recommended:

- GOOD PROGRAMMING PRACTICES
- MAINTAINABLE TESTWARE
- MAINTAIN THE TESTWARE
- MANAGEMENT SUPPORT: You will need this pattern to be able to change the current bad behaviour
- REFACTOR THE TESTWARE

You should already be applying these patterns. If not, do it!

Other useful patterns:
- GOOD DEVELOPMENT PROCESS: apply this pattern if you don't have a process for developing test automation. Apply it also if your process lives only on paper (nobody cares)
- LEARN FROM MISTAKES: apply this pattern to turn mistakes into useful experiences
- KILL THE ZOMBIES: Apply this pattern for a start
- DOCUMENT THE TESTWARE: you should be already applying this pattern. Retro fixing documentation is quite an effort. Do it in the future for all new projects and every time you have to update something old
- KEEP IT SIMPLE: Always apply this pattern!

Did you notice that these issues are almost identical? That they suggest almost exactly the same patterns as solution? The reason of course is that the underlying causes are the same in both cases:

- There are no conventions or standards on how to name either scripts or data (or if there are they are not applied).
- There is no rule where to save the data or the scripts to make finding them easier (again if there is, nobody seems to care)
- There is no standard template within a document for describing data or scripts so that the information is easily searchable
- There is no way to find out quickly where they are being used when you would like to change something and are not sure of the possible side effects

Usually nobody has time and so it's much quicker and easier to create something new than to look if it's already there, try to understand if it could be reused, and adapt it (risking disrupting something already running…).

This behaviour is building up a workload, often referred to as "technical debt", which if not addressed, can bring down an entire automation effort, As with financial debt, if you don't keep it under control, it can ruin you!

It is quite difficult to change such behaviour once it has taken root. Also often it is associated with some old hand who knows exactly where to find things all along. New team members are discouraged to try to document or reuse data or scripts out of fear of disrupting some existing tests. Finally when the old hand retires all that stuff will probably get thrown away and the team starts again from scratch. To avoid this you will definitely need to apply the management pattern MANAGEMENT SUPPORT.

## MANAGEMENT SUPPORT

### Pattern Summary
Earn management support. Managers should only support sound and well-reasoned activities, so we need to work at selling the idea initially and then keep them up-to-date with progress and issues.

### Category
Management

### Context
This pattern is applicable when test automation is intended to be used by many people within an organisation.
This pattern is not applicable for one person beginning to experiment with a tool to see what it can do.

**Description**
Many issues can only be solved with good management support.
When you are starting test automation, you need to show managers that the investment in automation (not just in the tools) has a good potential to give real and lasting benefits to the organisation.

In an ongoing project, inform regularly on the status and draw special attention to any success or return on investment. You still need to have good communication and a good level of understanding of current issues from management.

Sometimes a single incident can be more convincing than a large set of numbers, for example if a recurring bug is found by an automated regression test for a user that had complained about this same bug twice before.

**Implementation**
Some suggestions when starting (or re-starting) test automation:
- Build a convincing TEST AUTOMATION BUSINESS CASE. Test automation can be quite expensive and requires, especially at the beginning, a lot of effort.
- A good way to convince management is to DO A PILOT. In this way they can actually "touch" the advantages of test automation and it will be much easier to win them over.
- Another advantage is that it is much easier to SELL THE BENEFITS of a limited pilot than of a full test automation project. After your pilot has been successful, you will have a much better starting position to obtain support for what you actually intend to implement.

Some suggestions for on-going test automation:
- If you have INADEQUATE SUPPORT you may have to free some people from their current assignments.
- If you have INADEQUATE TOOLS you may need to invest in new tools or build or revise your TEST AUTOMATION FRAMEWORK.
- In these cases you may need to SELL THE BENEFITS in order to convince management that the investment will be worthwhile.

**Potential problems**
It is almost equally important to set realistic expectations about what the test automation project can deliver. UNREALISTIC EXPECTATIONS can lead to disappointment and frustration and you can lose management support just when you need it most.
Another problem that can arise is that the manager talks about supporting you and claims to support your efforts. But when you need to take some additional time or use additional resources, then „sorry, they are not available". This is not true support, but „lip service".

It is also possible to inadvertently set UNREALISTIC EXPECTATIONS by being overly enthusiastic about what can be accomplished early on in automation. It can be easy to show good results when you haven't yet encountered any of the problems that will occur later, such as the cost of maintaining the automated tests when the software under test is changed.

## Next steps

Having management support will enable you to apply patterns like KILL THE ZOMBIES that tells you to remove all data or scripts that are not in use, or REFACTOR THE TESTWARE which gives suggestions about eliminating doubles, documenting etc. These activities require quite a lot of effort and without support from management you will not be able to get the necessary resources or time. Also, with support, you will be able to introduce the GOOD PROGRAMMING PRACTICES and GOOD DEVELOPMENT PROCESS that will help you avoid the same issues in the future.

AUTHORS

**Dorothy Graham**

Dorothy Graham has been in software testing for 40 years, and is co-author of 4 books: Software Inspection, Software Test Automation, Foundations of Software Testing and Experiences of Test Automation. She has been on the boards of conferences and publications in software testing, was a founder member of the ISEB Software Testing Board and was a member of the working party that developed the ISTQB Foundation Syllabus. She was awarded the European Excellence Award in Software Testing in 1999 and the first ISTQB Excellence Award in 2012. She is currently working on the Test Automation Patterns wiki with Seretta Gamba.

**Seretta Gamba**

Seretta Gamba has over 30 years' experience in software development. As test manager at Steria Mummert ISS GmbH she was charged in 2001 with the improvement of the test automation process. After studying the current strategies, she developed a kind of keyword-driven testing and a framework to support it. In 2009 the framework was extended to support also manual testing. Seretta referred about it at EuroSTAR and got the attention of Dorothy Graham that subsequently invited her to contribute with a chapter in her new book (Experiences of Test Automation). On reading her bonus book Seretta noticed recurring patterns in the solution of automation problems. After gaining Dorothy's support, she is currently intent on cataloguing Test Automation Patterns.

*Bogdan Bereza*

# Two sister acronyms: QA and BPR

## What we have in common

Business process re-engineering (BPR) is an interesting discipline for QA engineers. For example, it has many crucial activities in common with business analysis. Actually, business process change, is what should often take place after business analysis, instead of system development. If, due to customer's misconception, they take place in parallel, this leads to many interesting phenomena, including the notorious scope creep.

Poor business process may destroy the best efforts of software engineers, when a potentially good software cannot be used properly in hostile business environment. For example, the stock-market internet bubble crash at the beginning of this century, was not caused by using XP end other not-so-concerned-about-the-requirements development methods, but because the need for these methods had been created by bad, cowboy, irresponsible business approaches.

Last but not least, testing a business process, and then "debugging" it, that is looking for the reasons why it fails, has much in common with software testing.

The story I describe here, happened to me almost symbolically the day after I had taught a three-days training course in BPR in Rome. It is a wonderful example of how business processes fail, as well as of how social, cultural and economic environment create conditions in which bad business practices can thrive.

So, after teaching this course, I was to go back to the airport and fly back home. What happened then, was a brilliant show why BPR is necessary, why it is not the same as introducing IT / Web support, and why it is so very interdisciplinary.

## Bad UX, or user experience

Already on arrival some days earlier, I was rather taken aback when my shuttle bus stopped somewhere in the middle of the rather crowded and badly-lit street, at a place not easily recognizable as any bus stop, except for a long queue of nervous-looking people with suitcases, and I heard a laconic info from the bus driver "Termini", the name of the main and biggest railway station in Rome. No railway station was there anywhere to be seen, but – thanks heaven for Google Maps and its Street View! – I managed to recognise a rather morose and ugly wall of a huge building as the station building. Naturally, this experience made me somewhat apprehensive before my return journey.

## Reassuring second experience

To feel safer, I searched the web for ideas and found a professional-looking web site of a shuttle bus company "Terravision" (http://www.terravision.eu/).

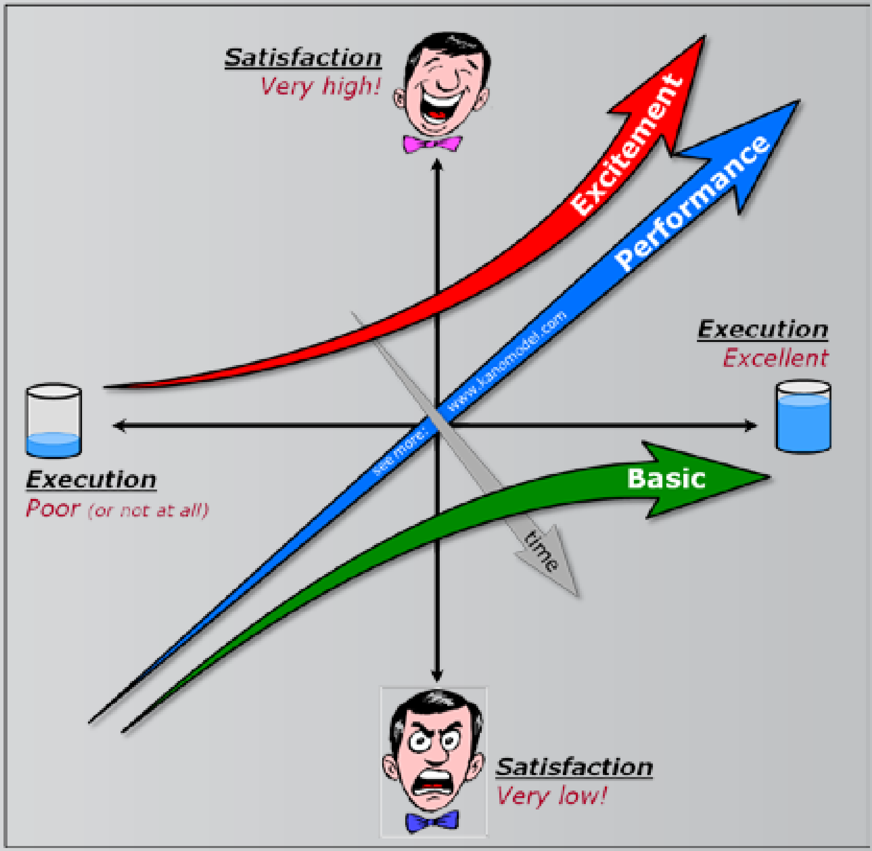
Wow, I could really buy now my bus ticket on-line, thus avoiding the scary prospect of perhaps having to buy my ticket from the bus driver or his assistant. Why was it scary to me? As the bus company's personnel did not wear any uniforms, which I had learned already on arrival, I was afraid I would not be able to recognize the right person easily (now I know I could – the assistant by shouting, the driver by smiling sarcastically at the stupid people attempting to board his bus).

So, until then I had already experienced a number of seriously "broken windows" (see Michael Levine, http://www.amazon.com/Broken-Windows-Business-Smallest-Remedies/dp/0446698482), seriously damaging my user experience. Lack of recognizable uniforms. Unprofessional behaviour – the assistant smoke a cigarette while talking to passengers. Badly lit location/venue/bus stop. No markings on where I was, lack of helpful information, the sight of obviously stressed people, queuing. Too bad!

Back to Rome. I was then still ready to revise my first negative impression, and I was well on my way to do it, when I found Terravision's sensible and well-organized web site, and could buy their ticket in advance without any unnecessary hassle, which I had by then learned to expect from shuttle buses' notorious customer interface. A warning sign, though: the necessity to exchange my ticket for a separate boarding card before being allowed to take the bus looked like a rather crazy and unnecessary complication, as the ticket I bought

This is the first interesting lesson of the story: the effect of an even slightly bad first experience has very devastating and lasting effect of how a product or a service is later seen. So, spiral development and prototyping in all respect, beware of creating such a lasting impression by demonstrating a very bad first system version to customers too early.

Product quality, whether a product is a service or a physical entity, does not hang in empty space, it is tightly connected to everything around it. Therefore, QA in general and testing in particular, may easily pay too much attention to technical details, instead of the complete user experience, or UX. The conclusion is **not** that technical, functional and extra-functional (yes, I hate the misleading and stupid term "non-functional") quality is not important. Yes, it is a **necessary, but insufficient** precondition of high UX.



www.kanomodel.com

Radek Hofman conducted a number of very revealing experiments in this area. They show clearly, and in a statistically significant manner, which is a rare occurrence in anecdote-prone software quality engineering, two important phenomena:

- "Software quality perception" (http://www.academia.edu/5515172/Software_Quality_Perception) tells you how simple rumours (oh so easy in the age of Facebook, Instagram, twitter and other rumour-spreading and brain-killing social media) can dramatically influence the judgement passed on quality by professional testers.
- "Behavioral economics in software quality engineering" (http://www.academia.edu/5515175/Behavioral_economics_in_software_quality_engineering) tells you how "history effect" – the influence of your first bad experience of a product, will stubbornly bias your perception of it.

Finally, if you'd rather have a model to see similar effects on a diagram, welcome to Kano model.

was valid for a specific hour, but what the hell, I thought, you must not expect perfection when you buy a 4-euro service, not a Rolls-Royce.

When I arrived next morning, on 20 November 2014 – well before the assigned time, to be sure – at the well-advertised Terravision Café at Termini, I was rather shocked again, when I saw a long communist-style line of nervous-looking people queuing to buy their tickets, thoroughly mixed up with people wishing to exchange their tickets for boarding cards. Obviously, the whole system of tickets and boarding cards was extremely clumsy, totally unnecessary, and awkward for everybody, both customers and for the rather angry-looking (tut-tut! Too sure about their jobs, perhaps?) girls inside the ticket booth. Yes, there was an A4-format paper telling those with tickets to "jump the queue" before those wanting to buy tickets. An obvious failure of localization: an attempt to impose a rather peculiar Italian habit on pre-dominantly international customers.

The whole queue thoroughly blocked the only entrance to Terravision Café inside. I started expecting the worst, but the exchange process ticket-for-boarding-card went surprisingly painless for me. I could not help overhearing, however, an elderly Swedish couple enquiring about the possibility of ensuring tickets for the next day, only to be told – in a rather brusque and unfriendly manner (tut-tut!) – by one of the girls behind the counter – that it was not possible longer than 30 minutes before bus departure. She did not mention the possibility to use their web site; why bother.

Clutching my precious boarding card in my somewhat sweaty palm, I endured without further ado being told that my bus was 20 minutes late, thanking business process analysis gods for deciding to go before due time, so I still had a lot of time.

- Where's the bus stop? – I enquired.
- Just outside! – was the answer. Not a sign of a bus-stop sign there, but a telltale, suitcase-armed queue made any doubts obsolete.

I joined the queue, wondering how to tell the end of the queue from its head, and how we'd be able to sort those willing to travel to Fumicino airport from those Ciampino-heading (no signs, no information boards, of course).

Finally, a bus to Fiumicino arrived. As I had already noticed a few days before, the fact that people exited the bus at exactly the same place as those wishing to enter queued, beautifully added to general chaos and irritation. I was happy for being observant, too, since the only indication on which way the bus was to go, was on its front, while its sides were decorated by a beautiful, but rather confusing sign "Rome <-> Fiumicino, Rome <-> Ciampino". Good idea, this! You really can, with some effort, design a customer process in the worst possible way! A gentle touch of very stupid and confusing user interface makes a mildly bad user process into real horror!

And horror did start immediately, as rather restive and desperate passengers attempted to enter the bus. Some had no tickets, believing they could buy them at the bus. Some had not exchanged their tickets for the required boarding cards, some were not sure to which airport the bus went, and some were unsure what to do with their luggage. The bus assistant immediately resorted to shouting, not so much to be heard, as to show his authority and passengers' stupidity. A good thing was, he was too busy shouting to be able to smoke, or perhaps he was a non-smoker, I couldn't possibly know.

Still shouting, he made, however, a very sensible move of telling the people who wanted to go to Ciampino, to form a separate queue. This could, possibly, give a thinking person a nice BPR-idea to actually mark two separate queues on the bus stop, and avoid some of the hassle in the future, but I do not think there was any thinking Terravision representative around. If there was, they might had discovered this genius solution many years before… Or simply, as is so often the case at IT companies, too, especially as software testers are concerned, the employees were expected to perform the duties assigned to them in an obedient manner, instead of arrogantly stepping on management prerogatives and proposing improvements. Good bye, Kaizen! Good bye, TQM! Good bye, Toyota system! Good bye, Juran, good bye, Deming! Terravision has still much to do before they catch up with the ideas that were known and widespread as early as forty years ago!

The bus to Fiumicino left, we Caimpino enthusiasts waited for our twenty-minute late bus to arrive. I decided I'd take a taxi when it was 09:40 (the bus would be 50 minutes late by then). As minutes went by, I could enjoy watching the growing restlessness of those waiting, and the total absence of any attempts to inform us about the situation from the nearby Terravision personnel. While I departed in the direction of the taxi stand, I could hear a Terravision lady shouting (they are good

at shouting at this company!) that the bus would arrive later still because of traffic. You may be interested to know, that on my way to the airport by taxi, I was told by the driver, that traffic from Ciampino to Rome was unusually light this morning… A blatant lie, too!

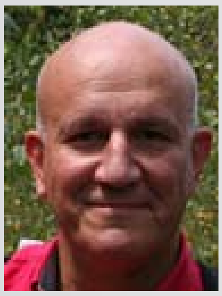
So this is the end of my Terravision story, but it'd be incomplete about adding some views on the feasibility of trying to achieve real BPR in any not market-driven situation. Socialism, as some of us can remember, was extremely adept at business process degeneration, rather than any improvement.

As I arrived at the nearby taxi stand, I was utterly surprised to find passengers waiting for taxis, not the other way round! Years and years of my age flew off my back and I felt thirty five years younger, in the middle of some communist era Eastern Europe city, where taxis, as any services, were scarce, and those in the power to bestow them on eager customers were arrogant, reckless and unfriendly. I gathered immediately that in Rome, taxi drivers' corporation alias trade union alias mafia must have won the privilege to limit the possibility to join this trade, thus ensuring for themselves endless monopoly benefits. In spite of my ex-communist training, it took me a while of patient and fruitless waiting first at the end of the line (the taxis then stopped at its head), then at its head (the taxis had by then switched their stopping habits), before I got back my uncanny ex-communist instincts and ran directly to grab a taxi before it had even come to the curb.

Here my BPR-Rome story ends. Ciao, Roma! I'll surely come back, yours is a beautiful city. Some BPR may make it a better place to live, and to visit, though! Terravision (what a f...ing arrogant name!), I hope you will pay my back four euro you stole form me, but anyway, I and probably all other passengers, too, would rather pay one or two euro more, and get serious and better service in exchange. So that you can have an extra bus in reserve, in case of one breaking down again, or some real, not imaginary, traffic jams in the future.

**AUTHOR**

**Bogdan Bereza**



Bogdan has worked with SQA for 25 years, for Swedish, German and Polish companies. He was the pioneer of ISEB Foundation, ISEB Practitioner, ISTQB Foundation and ISTQB Advanced in Sweden and in Poland.

Bogdan authored three books on SQA, translated two and published a number of articles in English, in Polish and in Swedish. He was involved in starting Swedish SAST (1995) and SSTB (2000), Polish SJSI (2003) and Polish SPIN (2006-2007).

QUALE

Magazine

**Publisher**
VWT Polska Michał Kruszewski
Przy Lasku 8 lok. 52, 01-424 Warszawa
Number NIP 5272137158
Number REGON 142455963

**Chief editor**
Karolina Zmitrowicz
karolina.zmitrowicz@quale.pl

**Deputy chief editor**
Krzysztof Chytla
krzysztof.chytla@quale.pl

**Editors**
Bartłomiej Prędki
bartlomiej.predki@quale.pl
Michał Figarski
michal.figarski@quale.pl

**WWW**
www.qualemagazine.com
www.quale.pl

**Facebook**
http://www.facebook.com/qualemagazine

**Advertisement**
info@quale.pl

**Cooperation**
If you are interested in cooperating with us, please send us a message:
info@quale.pl

The following graphics have been used:

Cover

http://pixabay.com/pl/aplikacja-telefon-kom%C3%B3rkowy-iphone-141046/

Content:
http://pixabay.com/pl/brzegowe-morze-morze-ba%C5%82tyckie-339251/
http://pixabay.com/pl/fraktal-spirala-gwiazda-%C5%9Bwiat%C5%82o-105892/
http://pixabay.com/pl/cz%C5%82owiek-biznesmen%C3%B3w-kobieta-162951/