QUALE

Quale 1/2015

User Story Effectiveness: How to Boost Your Development Quality

Dominik Maximini

Vision Planning Chapter 1 - Objectives

Tom Gilb

Testwarez 2015

Software-QS-Tag 2015

Content

SOFTWARE ENGINEERING

Chief editor
Karolina Zmitrowicz
karolina.zmitrowicz@quale.pl

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

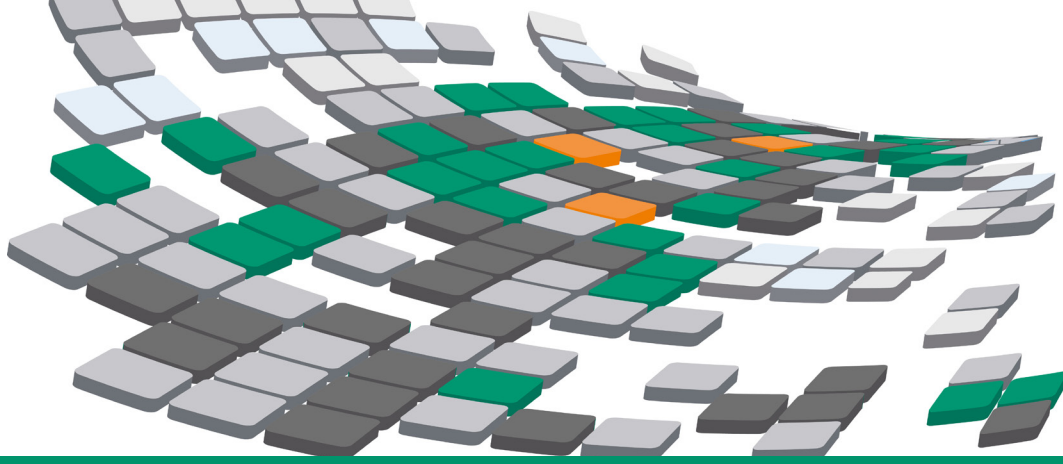
Editors
Bartłomiej Prędki
bartlomiej.predki@quale.pl
Michał Figarski
michal.figarski@quale.pl

Cooperation:
Tomasz Olszewski
tomasz.olszewski@quale.pl
Zuzanna Gościcka-Miotk
Zuzanna.goscicka-miotk@quale.pl

Website:
www.qualemagazine.com (ENG)
www.quale.pl (PL)

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

TestBench



TestBench – The smart solution for all test tasks

imbus TestBench is the working environment for test teams of all sizes, and provides everything that is required of a modern support tool:

Test planning, test design, test automation, test execution and reporting.

TestBench can be integrated seamlessly into your existing tool chain. Its key benefits include intuitive operation, high-performance test functions and a practical model for rights and roles, besides which it also supports structured testing according to the ISTQB® standard.

TestBench provides complete control and transparency in the content, status, progress and results of your software tests – at all times and across all development locations and all releases/versions of your software.

20 years of test management experience are reflected in every aspect of the TestBench.

- Provides fundamental support for all tasks in software testing
- Fully integratable in your test system landscape
- Rapid deployment thanks to various test description methods
- Especially efficient for test design and test specification
- Convenient for test planning and control
- Flexibility in automated test execution
- Extremely good value for money using the renting model

TestBench allows you to concentrate on what is truly important for your company:

- Ensuring that your products and IT systems are of the highest quality through optimum test control throughout their entire life cycle.
- Reducing development costs through efficient working processes and rapid flow of information between teams.
- Shortening your product's time-to-market by eliminating unnecessary bug fixing cycles.

> learn more...

imbus AG, Kleinseebacher Str. 9, 91096 Moehrendorf, Germany
Phone +49 9131 7518-0, testbench@imbus.de, www.imbus.de

How will „The Future of Testing“ look like?

imbus presents trend study by Dr. Bernd Flessner

The IT industry is developing rapidly – and so is the software testing sector.
What should one brace oneself for in the years ahead? imbus has asked the futurologist and science writer Dr. Bernd Flessner to cast light on the future of software testing.

The trend study “The Future of Testing” is the result.

Dr. Bernd Flessner invites the readers to take a look into the future. On circa 40 pages, scenarios for the time periods “from 2020”, “from 2035” and “from 2050” describe possible developments of the testing sector – each examing both a positive and a negative viewpoint. The study features forecasts on much-discussed technological developments, including industry 4.0, Outernet and smart home.
It explains, how they could directly affect the testing sector – and it derives, what this means for the software tester profession.

“The Future of Testing” features besides the scenarios also the results of an exclusive Delphi survey.
Each five well-known international experts of the IT and the testing industry were asked to assess the effects of specific trends on the software testing industry in the near future.
Dr. Bernd Flessner draws in “The Future of Testing” on the theses from his keynote “Wir Prothesengötter” at the Software-QS-Tag 2013 and the panel discussion at the Software-QS-Tag 2014.

The study was published in English and German. Both versions can be downloaded in PDF format free of cost at www.imbus.de/en/downloads.
Reader feedback on the scenarios illustrated in the study is always welcome – please send it to presse@imbus.de. imbus will gladly forward the feedback to the author and thus establish the contact.

> free PDF...

imbus AG, Kleinseebacher Str. 9, 91096 Moehrendorf, Germany
Phone +49 9131 7518-0, presse@imbus.de, www.imbus.de

Dominik Maximini

User Story Effectiveness: How to Boost Your Development Quality

The concept of User Stories might seem simple, but it definitely is not. To use them in a professional manner and improve the quality of your requirements and ultimately your product, one has to do a lot of work. This article explains what and how this can be done. However, be aware that there are many different opinions on User Stories out there. Depending on whom you ask, even the core elements of a User Story might vary.

Origins and Concept

User Stories are part of eXtreme Programming and advocated by Agilists like Ron Jeffries and Mike Cohn [1]. The original idea was that of the "Three Cs": *Card, Conversation* and *Confirmation*[2]; meaning a physical ticket of a certain maximum size (the card) should represent the requirement, the clarification should happen by means of dialogue (conversation) rather than written thought exchange, and there should be clear and verifiable acceptance criteria (the confirmation), so success can be evaluated. The question what to put on the card was answered in numerous ways. My personal favorite is that offered by Connextra [3]: ***As a <role> I want <desire>, so that <value>***. Traditionally, acceptance criteria go to the back of the card.

This concept is only very rarely put to good use. Far more often, misconceptions are introduced.

Eight Common Misconceptions

If you get things wrong, you will fail in one way or another. This is normal and not necessarily bad, if you have time and resources to learn from failure. Sometimes you do not have the time to learn, so it might be helpful to consider some pitfalls beforehand. Here are eight misconceptions you should avoid in order to increase quality (see: frame on the next page).

EIGHT MISCONCEPTIONS

Misconception 1: You must use User Stories if you do Scrum!

Actually, you don't. Scrum is a lightweight framework that tells you to define "Product Backlog Items" [4]. User Stories are one way to do it, but certainly not the only way. There are other ways of capturing requirements/desires that might better suit your context. If you use a tool without careful consideration and reason, you might be on the verge of hammering a screw into the wall – with a wrench. A painful and potentially expensive experience.

Misconception 2: Numerous pages of specification for each User Story are required.

Some organizations use their traditional requirements specification process and put the headline "User Stories" on top. While you might actually need comprehensive requirements in rare scenarios, User Stories are intended to replace documentation with conversation. If your requirement fills more than one page, you most likely won't be able to finish it in one short iteration and thereby compromising your Agility.

Misconception 3: You don't need to specify the <role> part. Just put "User" or "Product Owner" in.

Losing the first part of the syntax means that you no longer think of who the value is for. This might indicate you don't really know your stakeholders. Even if you only have a single customer and a single product, there are most certainly several roles involved. People in those roles might want to pursue different goals. It's good to know for whom you do what, otherwise 70-year-old aunt Harriet might end up with a command-line interface 25-year old admin Joe requested.

Example of this misconception: *"As User, I want a new login dialogue, so that I can access my profile within 10 seconds."*

Misconception 4: It is enough to state the <desire>. If we do that right, we don't need the <value> part.

So value is not important? Usually, misconceptions 2 and 4 play together: organizations don't specify what they need, they specify what to do. Of course, when you specify what to do, the monkeys doing it don't need to know why they are doing it. If you truly want to be Agile, this approach doesn't work as the expectation is that your team thinks for itself and provides better solutions than you could deliver alone. Your team needs to know the value required, otherwise they cannot focus their thinking and might build what you asked for, even though something else would be more cost-efficient or filling the actual need better.

Example of this misconception: *"As Aunt Harriet, I want a new login dialogue."*

Misconception 5: No quantification of what value is needed.

Since we are talking about value already: If you don't quantify your value, you will never know if you achieved it. Instead, you will have many lengthy discussions with your stakeholders.

Example of this misconception: *"As Aunt Harriet, I want a new login dialogue, so that I can access my profile faster."*

Misconception 6: Acceptance criteria are not Agile. We don't need them.

Usually, every User Story has some acceptance criteria. They represent what was already discussed and decided. They should also describe the expectations in the form of tests the person in the role has on how the product should behave when developed. If you do not capture what was discussed, mistakes happen and unnecessary discussion time is increased. This stays true, no matter if some self-appointed Agilist tells you to "stop expanding the documentation" since documentation wasn't agile in his opinion.

Agile primarily tells you to do what makes sense. So capture the information you need.

Misconception 7: No measurement of results. While features are tested, value is not.

Most teams are not able to tell their stakeholders how much value they created in the last Sprint/release/decade. They can only tell how many features they delivered. This also means that the team does not know when to stop working on a certain goal. As soon as you have quantified your goals, you must start measuring your progress towards them. This step alone should multiply your transparency, focus, and productivity.

Misconception 8: Planning ahead is waste. We can create User Stories on the spot. No prior work is needed on them.

As Eisenhower said: *"[…] Plans are useless, but planning is indispensable!"*

If you do not exercise your brain in time, you won't notice when you start missing your target. You need to have at least a rough and quantified idea of what you want to achieve within a release and a much clearer idea for your next Sprint, otherwise you are at risk much like a driver speeding through the fog at 220 km/h. Agile does not mean that you stop planning. It means you plan just enough to be prepared for what you are trying to achieve.

These – and other misconceptions – lead to low quality and waste in many forms. The most common ones are unnecessary discussion, developing the wrong features in the product and rework after product delivery. It is up to you to minimize these issues.

How User Stories are supposed to work

The purpose of User Stories is to reduce documentation time, allow flexibility in the solution and to avoid misunderstandings resulting from indirect communication. They are not meant to be a full requirements specification document, as it is used today in most traditional projects. However, you can only be successful with User Stories if you focus on value – and quantify it. Since value is always dependent on the recipient, this is the place to start.

How to fill the <role> part

You probably have dozens of stakeholders, and you have to know them all. In most cases, those who pursue the same goals can be grouped. Others are not interested in the requirements as such or can be queued for the time being due to other reasons. This leaves you with a list of stakeholders and stakeholder groups you must take care of. One very efficient way to find out and capture what those groups expect is to create "personas". This means that you create a flipchart and note down all aspects that characterize this particular group and are relevant to the project. For example, you might find yourself with a group of end-customers, averaging 60 years of age, female, wearing strong glasses, being computer-illiterate and primarily being used to ordering via catalogue forms. Having captured this data, you can label it with a name and picture, sometimes with a role description. In our case, this could be "Aunt Harriet". From this point on, whenever you mention Aunt Harriet, everybody involved will know that this group of customers needs slightly bigger fonts, clear forms on a single page and no information messages whatsoever popping up on the screen. Don't you dare to force this group to remember their passwords!

In addition, it makes sense to capture the goals of every persona, that is, stakeholder group. While these do not have to be noted down on the same flipchart, you need to be aware of each one's goal conditions.

How to fill the <value> part

Now that you know what your stakeholders want, you can quantify it, top-down. Start with the project's goals, continue with release and iteration goals - all quantified. Those goals should result from the careful consideration of different strategies to achieve the respective higher goals. So project, release, and iteration goals are streamline with higher business goals, and with each other. This means Sprint goal Z represents a strategy to achieve Release Goal R1 and you only have decided to go for Sprint goal Z, because strategies X and Y seemed less efficient. Only then start thinking about your User Stories, and only then create them. This approach sounds easy, but if you aren't used to it, you are up for a tough time. Here are some examples of relatively good value statements in User Stories:

Aunt Harriet

- 60 years old
- female
- wearing strong glasses
- computer-illiterate
- primary being used to ordering via catalog forms
- panics when information messages pop up on her screen
- cannot remember her passwords

As Aunt Harriet, I want a new checkout dialogue, so that I can complete my order within 10 seconds.

As Aunt Harriet, I want bigger fonts, so that I can read the product information from a distance of 80 cm.

As Teenager Chris, I want a clear site structure, so that I can find all navigation options with my smartphone within 5 seconds.

As Teenager Chris, I want to be able to pay for my purchase via smartphone, so that I do not have to remember any payment information on checkout.

Of course, there is still a fair amount of uncertainty in these statements. We do not know from what point Aunt Harriet wants to complete her order or what "complete" exactly means for her. We also do not specifically know how bad her eyes are and what type of computer, screen and settings she uses. As for Chris, we need to know what type of smartphone he uses and what "navigation" means for him, as well as which type of payment process is useful for him. There is probably much more vagueness in the statements above, but the goal is clear enough to discuss it in a focused way while measuring the outcome. All relevant uncertainties either need to be specified as "acceptance criteria" upfront, or must be discussed and clarified during development. If you have to invest a lot of time in the specification process, you are probably better off discussing the details during development. Usually, a multidirectional conversation produces mutual understanding far quicker than unidirectional documents ever could.

How to fill the <desire> part

Knowing the value you are striving to achieve, you now can choose the best strategy to get there. This implies that you strictly separate "goals" from "means". The goal is specified in the value part of your User Story, the means is what many people describe as "feature" or "function". Unfortunately, most organizations start with formulating a bunch of features instead of specifying the value they are striving for and evaluating different strategies to get there. This leads to inferior efficiency and production of waste. What should be done instead is a thorough comparison of strategies with respect to their ability to reach the aspired values, under consideration of their respective costs. Only when this has been done, should the User Stories be written.

Let's consider our example: We want to improve our sales volume and figured out that one reason for lower sales is that our target customer group represented by Aunt Harriet is dropping out of the order completion process after 10 seconds in 80% of all cases. This results in lost potential sales, so we want to allow Aunt Harriet to be able to complete her order within 10 seconds and presume we could get the drop-out number down to 40% by achieving this goal. We are now considering different strategies:

1. We could offer call center support to complete the order process for Aunt Harriet. This would be costly, but would reduce Aunt Harriet's time in the order completion process to zero.

2. We could change the whole sales platform in a way that when somebody leaves the page, a dialogue comes up, asking if the order should be completed now. Unfortunately, Aunt Harriet doesn't like any pop-ups.
3. We could simplify the checkout process with a one-page dialogue that looks familiar to Aunt Harriet.

Out of these three strategies, only option one and three are valid since Aunt Harriet could not cope with a pop-up dialogue. Both strategies could solve the issue and reach the goal, but the costs are significantly lower for option three. So we would opt for this one and formulate a User Story:

> *As Aunt Harriet, I want a new checkout dialogue, so that I can complete my order within 10 seconds.*

If you want to learn more about defining value, separating goals from means, and reducing uncertainty, I recommend you read Tom Gilb's excellent book "Competitive Engineering".

Acceptance Criteria

No Agile method tells you to stop thinking. In fact, the primary Agile tool is common sense – at least in my opinion. So whenever you feel the need to document something, note it down. Whenever you notice that your documentation doesn't benefit your project, stop writing. It's really that simple and means that your level of documentation can change with rising or declining needs throughout a project. The right place to capture decisions and information directly related to the User Story is the User Story card (or its equivalent in digital tool). This part is often called the *"Acceptance Criteria Section"*. Don't try to capture every little tidbit of information, stick to the really important ones. Let's take a look at our example again:

> *As Aunt Harriet, I want a new checkout dialogue, so that I can complete my order within 10 seconds.*

Acceptance criteria:

- The dialogue must fit on one screen page without scrolling
- It must roughly resemble a catalog form
- We only capture essential information

In many cases (including this example), acceptance criteria are too vague to be of great help. We know the intentions, but we aren't sure what exactly is meant. This is where direct communication comes into play: A team confronted with this type of acceptance criteria will immediately ask questions like:

- "What exactly does 'one screen page' mean?"
- "What is a catalog form in this context?"
- "What information is essential for us?"

Whatever the outcome of this discussion is can be used to clarify the requirement. The answers are usually stored in the acceptance criteria:

- The dialogue must fit on one 1024x768 pixels screen page without scrolling
- It must roughly resemble a catalog form, see mail-order catalog of "Quelle Inc." from 2010 for details

- We only capture essential information, which is: Name, postal address, and billing information, while "pay via bank transfer after goods received" is the standard option

As soon as a good-enough level of clarity is achieved, the Development Team can start to develop the <*desire*> and achieve the <*value*>. Small uncertainties are not an issue since the concept of User Story is based on close collaboration between Development Team and customer (e.g. Product Owner, Stakeholder, etc.). Better options and wrong turns are spotted early, discussed, and remedied.

Verification

If you invest time, money and sweat into anything, you should verify if it was worth the effort. To do this, you must do two things in the context of a User Story: First, verify if the desire is fulfilled and your product actually does what it was intended to do. Second, verify to what degree the aspired value was achieved. First verification can be done through acceptance tests, ideally in an automated way. If you were really advanced you would even implement the automated tests before you would have developed the actual product (ATDD) – this way you optimize your effort and know immediately when you are done. The value verification is usually a bit more complicated, because it can not nec-

AUTHOR

Dominik Maximini

Dominik Maximini is an Agile Coach and Professional Scrum Trainer, working for NovaTec Consulting GmbH in Germany. He has written several books and articles and is a frequent speaker on national and international conferences. His Blog can be found at http://scrumorakel.de/blog/. He is looking forward to your feedback.

essarily be done in the product itself. In our example, we have to verify two values: Completing the order within 10 seconds and reducing the drop-out number to 40%. While the order completion time can be measured from within the application or even the feature, it is more difficult with the drop-out number. Here we need to survey real customers purchasing real products. However, it must be done – otherwise we have no clue whether we achieved our goals and how we should adapt our strategy in the future.

Conclusion

The User Story concept is simple, but requires a lot of thought to be put into it in order for it to work well. The aim is to achieve good-enough quality in your requirements to enable you to build a great product. You have to know your stakeholders and their goals before you start writing anything down. Then you need to specify and quantify the added value you are striving for. Only when you know this value, can you weigh different strategies against each other, choose the best one, and implement it via a User Story. Additional information and decisions can be captured in the form of acceptance criteria. The whole concept has a natural tendency to vagueness, which is intended to stimulate constant direct communication. When using User Stories, save yourself time and money by doing it right!

REFERENCES

1. E.g. Mike Cohn (2004), *„User Stories Applied: For Agile Software Development*", Addison Wesley
2. Ron Jeffries (2001), *"Essential XP: Card, Conversation, Confirmation"*, http://ronjeffries.com/xprog/articles/expcardconversationconfirmation/
3. http://agilecoach.typepad.com/photos/connextra_user_story_2001/connextrastorycard.html
4. Cf. The Scrumguide. http://scrumguides.org/

TESTWAREZ

October 7-9 2015

Windsor Palace Hotel & Conference Center, Jachranka, Poland

en.testwarez.pl

Join the biggest Polish conference for software testers.

This year we celebrate our 10th anniversary edition!

Sign up today!

Tom Gilb

Vision Engineering

Chapter 1 - Objectives

How to support your core business vision by detailed practical plans and actions

Clarifying Core Ideology

We can begin with the core ideology statements themselves. It might be healthy to increase the clarity and intelligibility of the core value statements, and the core purpose statements, themselves. This can either be done as a clarification, without changing the original statements themselves – they may be somewhat 'holy' and traditional – or you might find it advantageous to directly modify them for clarity.

Clarifying a Core Value Statement

Take for example a core value statement:

3M [1, p.68, p.152-3] **"Tolerance for Honest Mistakes"**

Anyone could reasonably ask:

- How much 'tolerance'?
- What does 'honest' mean?
- What is a 'mistake'?

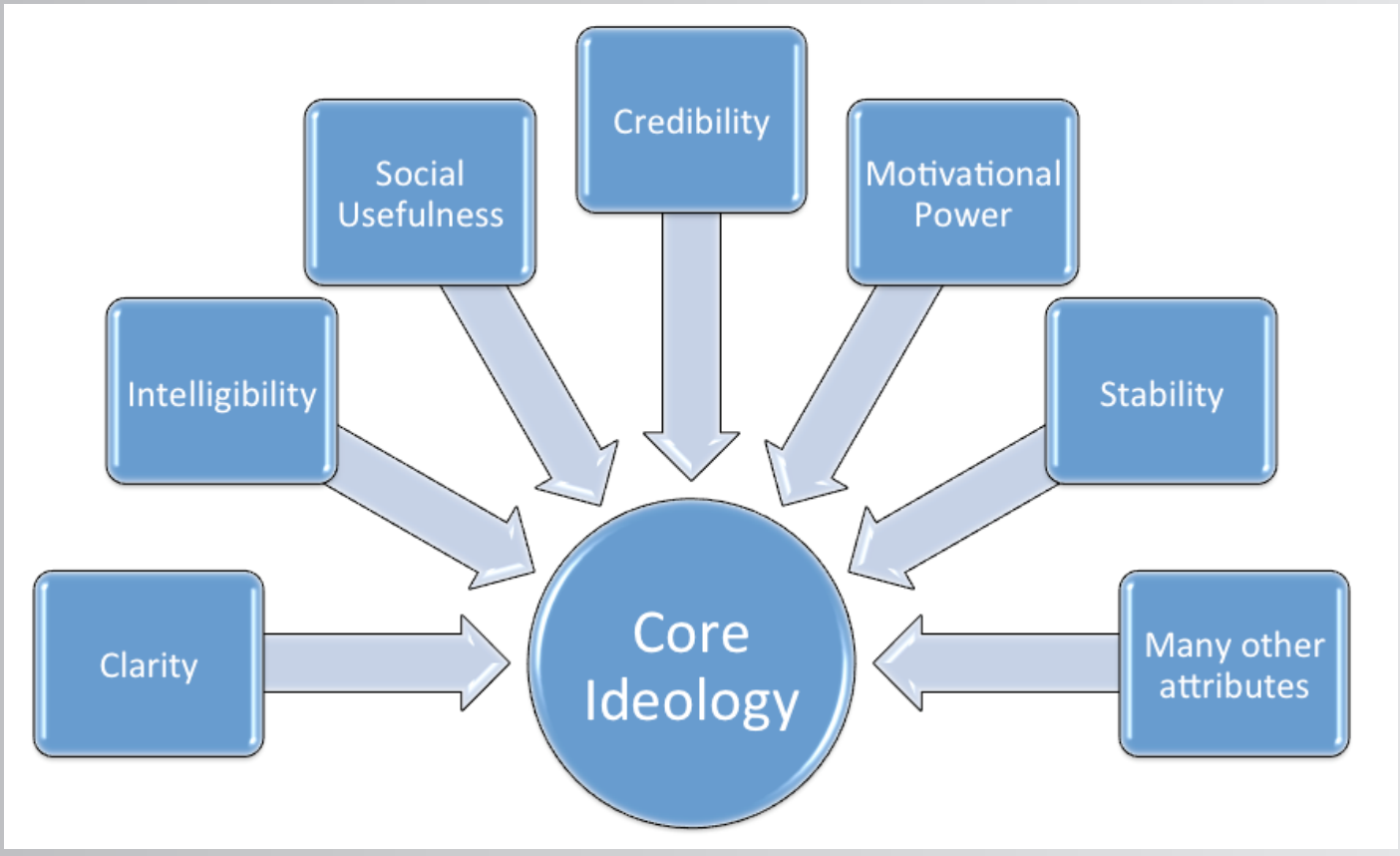


Figure 1. Some attributes of a Core Ideology

No doubt, the corporate practice itself, and senior employees, can answer these questions in practice. But let us say that we wanted to clarify even better, because of rapid growth in distant cultures – so people got it better, faster?

We might also want to clarify as a better basis for deriving more-detailed plans and practices. We can rewrite the statement, or provide helpful interpretation commentary.

Why?

A useful approach to clarification is to ask 'why?'. In this case the reasons (for 'tolerate honest mistakes') seem to be to encourage experimentation, so that improved ideas are more likely to emerge, than if people were afraid of being criticized for failed experiments.

So we could rewrite the core value, in order to get nearer the real intent:

> "Judge efforts on their **useful outcome**, not on necessary experiments to get there."
>
> "Judge ***results***, not process".

There are a large number of other possible methods for clarification of core values, and indeed any planning statement, at any level. More follows in the rest of this chapter, and other chapters in the book, and its references.

The main point is that no matter how 'hallowed' the statement is ("All men are created equal.") you should consider as your first step, some clarification of the core statement itself, maybe a real 'elevation': it is 'core', right?.

The penalty, if you fail to clarify, might be that all other critical planning will be based on misinterpretations of the core! The cost to get it right is small; like an hour to a day of effort.

Clarifying a Core Purpose

It is arguably even more critical to have a rock solid, crystal clear Core Purpose Statement as the basis for further planning.

Take for example: Merck [1930s, 1, p.236]
"To preserve and improve human life"

This is intended as the fundamental performance measure of all corporate activity for a pharmaceuticals company. It is of course constrained – and thus partly defined - by their core values (keywords: responsibility, excellence, science, honest, profit).

If someone found that their pharmaceutical technology could be used for animals

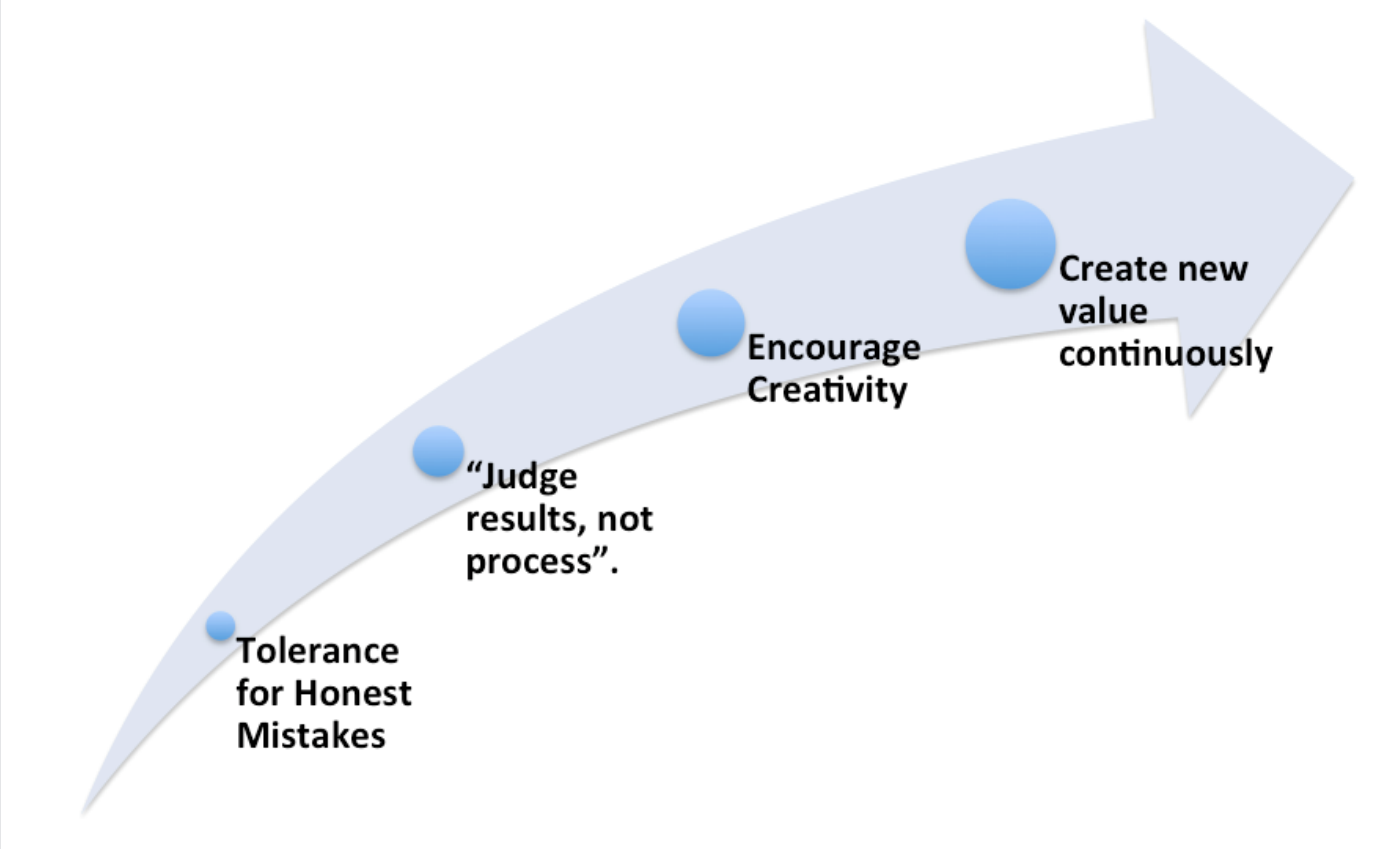


Figure 1. Asking 'Why?', multiple times, is not only a practical way to 'clarify' a core value. It might easily lead to your own recognition that you need to reformulate your core values at a higher level. What you had originally, might have been but one means (tolerate mistakes) to the real ends (create value).

or plants – does the 'human life' idea apply, or disqualify the product area?

If they could extend human life for people living in a coma, does that count, as within their core purpose?

If they found psychological, mechanical, electronic or religious means, or other 'services', for improving the human life; are they valid, or is there some constraint about sticking to the drug business, even if other available means are more cost-effective?

I can't see where it says, strategy constraint 'drugs only'.

Let us look at some possible Merck 'clarifications' for **"To preserve and improve human life"**:

'To improve life quality by ***any means***.'

'To ***provide products*** to improve life quality.'

'To develop ***knowledge***, and apply it, to get improved human mental and physical life quality.'

Each one of these is significantly different from the other. So consider a rethink of the articulation of your most fundamental purpose, before making it the touchstone of all other planning work.

Defining a Scale of measure

One 'device' we will need, sooner or later, to really clarify performance objectives, is to define them, so that we can quantify them in practice.

EXAMPLE

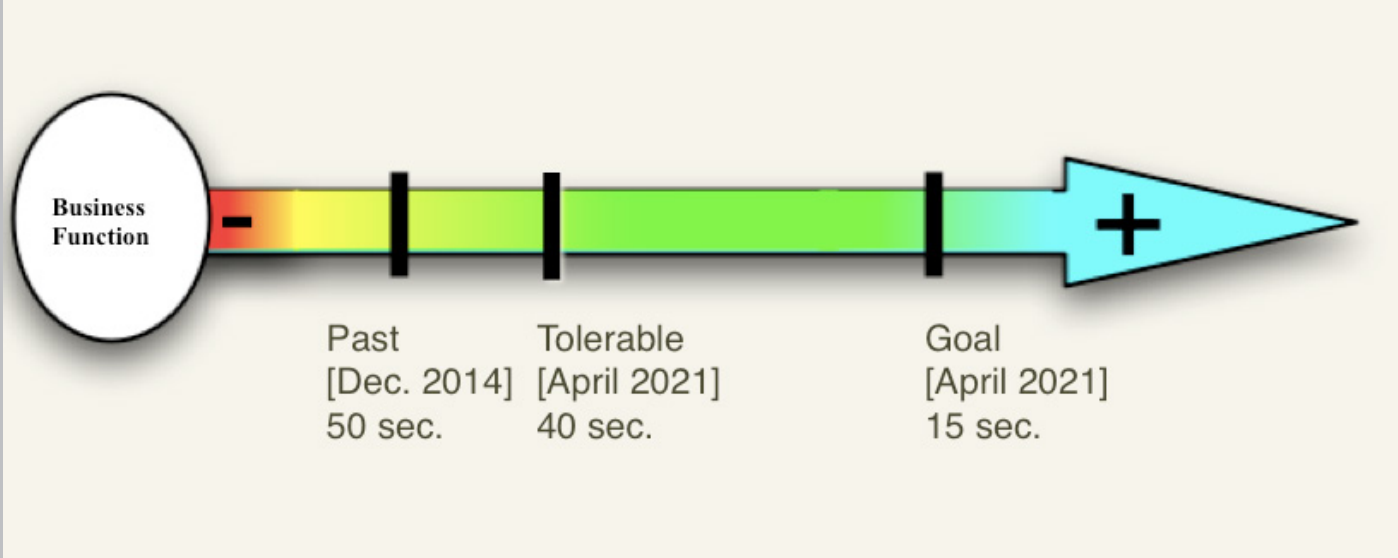


Figure 3. Any performance measure, for an organization, can be thought of as an arrow, with the performance itself varying from very bad (left hand side '-') to very good (right hand side '+'). The arrow itself is defined as a useful performance variable, such as "average time needed to make a sale". Along this defined Scale of measure, we can then describe useful degrees of performance data. Past performance, and future plans.

The fact that we can then set numeric targets, and numeric constraints, and track them, is powerful; but in fact is not the main point.

The main purpose of 'quantification of performance objectives', is to force us to think deeply, and debate exactly, what we mean; so that others, later, cannot fail to understand us.

Performance objectives, ranging from core objectives to 'any' detailed performance objective – where 'getting better-and-better in time' is implied – can always be defined using 'scales of measure'.

And once we have agreed 'scales of measure', we can apply a large useful set of devices, to utilize the fact that we have entered 'numeric territory'.

Less poetry, more logic [nothing wrong with poetry and the arts, in their place].

Let me introduce a 'planning language' method ('Planguage', I call it, rhymes with 'language').

- We write "**Scale**:… " in front of our defined **scale of measure**.
- Note that we are NOT defining a testing, tracking or measuring process (later called 'Meter: ---") yet. Volts, not voltmeter.
- We are just enabling ourselves to think about our most cherished core purpose numerically.
- Let us try with the example: **"To provide products to improve life quality"**.
- What is the 'scale' to quantify this, and to define what we mean numerically?

S1: Scale: New Products Released Annually.

You can see the weakness with this draft, S1?

S2: Scale: Annual Sales for all products that improve life quality.

And the weakness with this, S2? For example, Merck is famous for giving away some drugs!

I would prefer this draft:

> **S3: Scale:** Estimated **B**etter **D**ays for defined [**Life Form**] as a direct result of defined [**Products**].
>
> **Better Days:** days where the entity themselves, or another better judge, would judge their life to be better than without Our Product.
>
> **Life Form:** {Human, Animal, Plant}
>
> **Products:** {Patents, Drugs, Machines, Licenses, Services, Distribution, Education, Motivation, Others}.

S1,S2 and S3 are arbitrary reference tags to the statements. Capitalized terms ('Better Days') are formally defined terms.

I would argue that 'S3' is a pretty good draft effort, as a powerful definition of our Core Purpose. The core purpose has not changed. But our ability to articulate it, and to discuss any related plans, is arguably improved.

I would argue that it can help us, in deriving relevant aligned plans, and help us

EXAMPLE

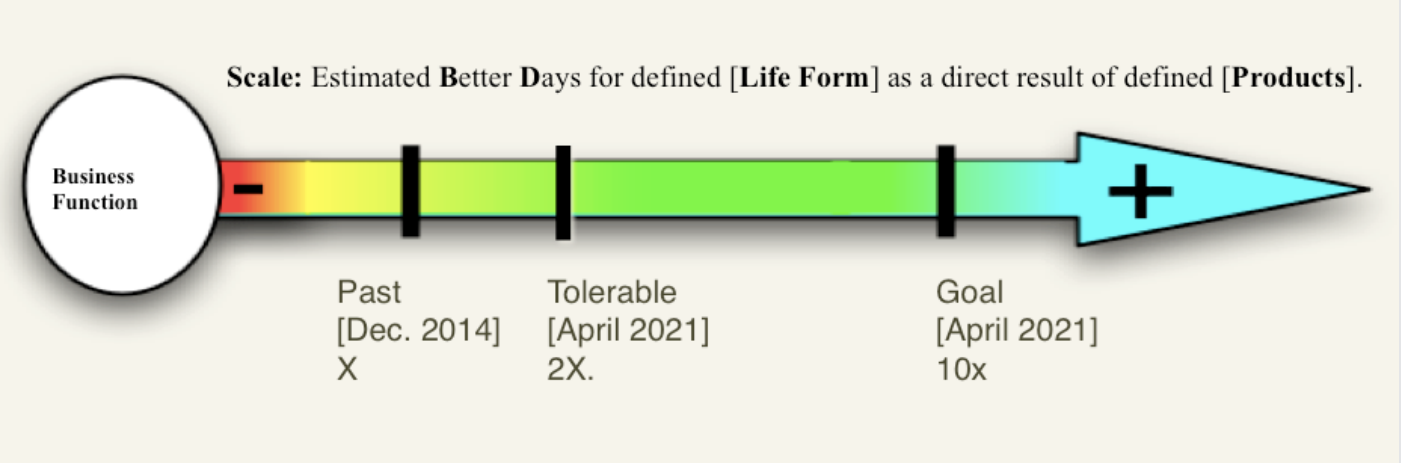


Figure 4. My suggested, draft, 'scale of measure', for my 'improved' variant of Merck's original core purpose ('To preserve and improve human life') "**To *provide products to* improve life quality**".

to judge their effectiveness, for promoting our core purpose.

I would argue that 'S3' is a better top management tool than the 'poetic' phrases (S1, S2) that preceded it, even though poetry might still be useful for simple emotive presentation, in some circumstances.

Management can usefully distinguish between 'presentation formats' (like:

> **Ambition Level:** To ***provide products to*** improve life quality.

and '**planning** formats' (like S3).

You probably need both formats, for different audiences and purposes.

Deriving Objectives from the Core Purpose and Core Values

A defined 'Scale' gives us a 'numeric-scale definition' of core value and core purposes.

This enables us to move our planning from a 'poetic' to a 'numeric' basis.

We can now plan, by determining a useful set-of-points on that scale of measure.

There are three major planning categories:

- **Benchmarks:** points for comparison with plans.
- **Constraints:** borders, worst acceptable levels.
- **Targets:** levels of performance we are aiming for.

I have defined a number of these concepts in Planguage. Here is a useful set.

Benchmarks: levels of performance worth knowing about, in comparison with future planned levels. For us, for competitors, for the past and possible future.

> **Past:** any estimated, or measured, level for us, or others, that is interesting to compare future plans to.

> **Trend:** an estimation of the levels, good

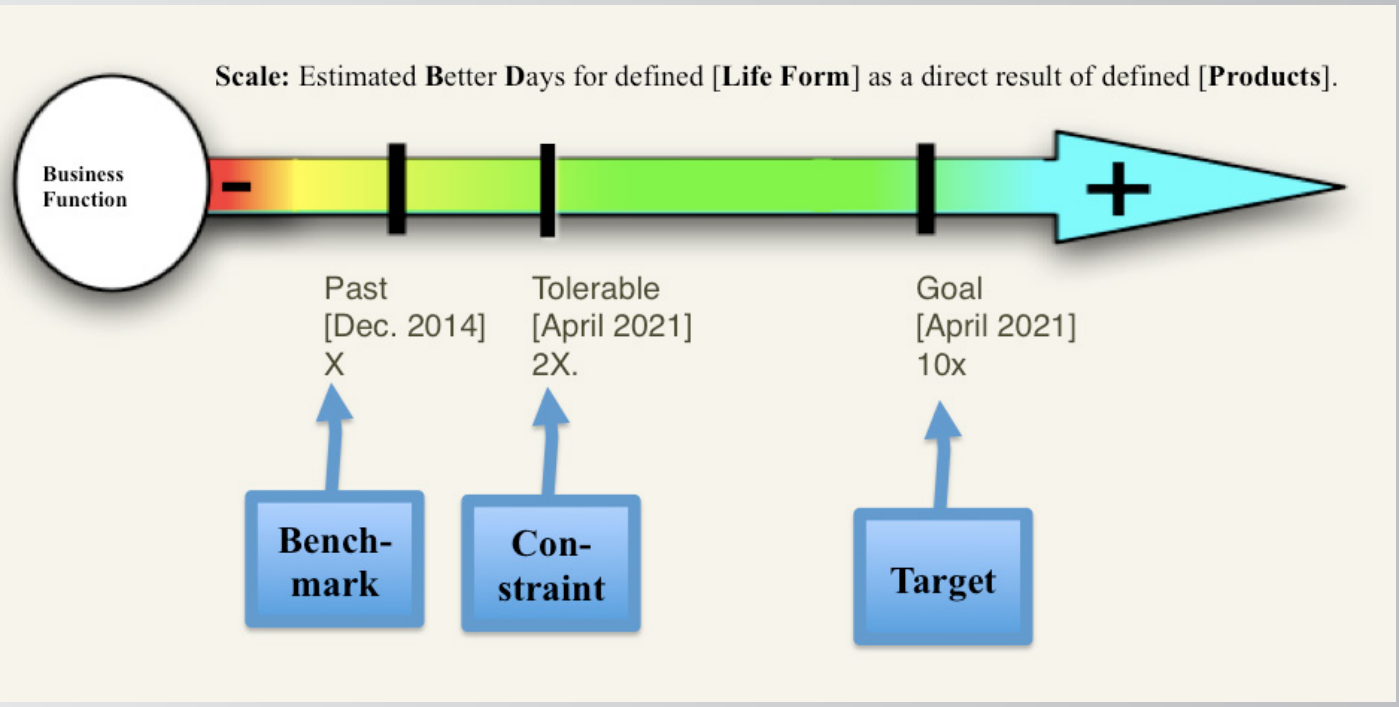


Figure 5. Three major categories of 'levels' of performance.

or bad, that will probably be reached by us, or others, at defined times, and under defined circumstances.

Record: a state-of-the-art extreme, attained under defined conditions.

Constraints: less-than-successful area we are trying to avoid.

Catastrophe: the edges of a numeric range of performance results that are disastrous in consequence, and possibly not recoverable.

Tolerable: the edges of a numeric range that is tolerable, just above Catastrophe, but still failing to some degree to satisfy, even at the OK level.

OK: a range just above the tolerable range.

- Not intolerable.
- Not failing.
- Pretty 'good',
- but not yet at an ambitious and competitive 'success' level, the Goal.

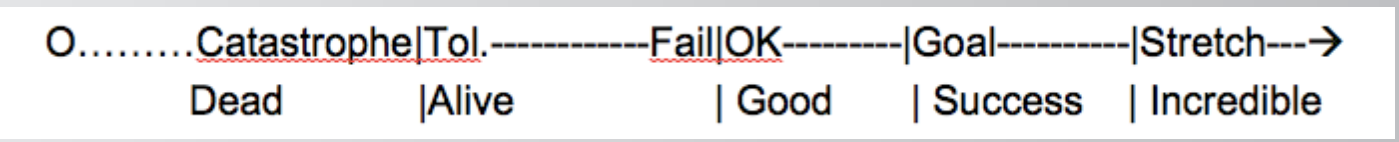


Figure 6. Points and ranges on a scale of performance

Targets: levels we are aiming to reach.

Goal: a level which is both satisfactory, and considered feasible; you can promise it.

- 'Better than the Goal level' is a range, we can call the success range.
- But, there might not be any defined or planned value for getting better, in that range.

Stretch: a level that has stakeholder value, and which you will attempt to move towards, if resources remain, after all other critical objectives' Goal levels are reached. This means we are not fully committed to achieve this level: it depends.

Ideal: (rarely used except to distinguish it from more practical targets) a level of perfection unlikely to be achievable in practice, and not necessary (since competitors cannot get to it either). But we can aim to 'tend towards' it. From another point of view, it is also a 'benchmark'.

- Examples 100% availability, zero time to learn to do a non-trivial task.

Determining the numbers

The next step is to determine some 'planning values' (some numbers on the scale that are valuable for our planning purposes), using any useful means to determine the numbers.

For example:

Merck Core Purpose:
S3: Scale: Estimated Better Days for defined [**Life Form**, default: Humans] as a direct result of defined [Product, Default: All].
Past: 100.
Goal: 1,000.

This makes the point that we plan to get 'ten times better'. But it would be more intelligible, if we added some 'implied but not stated here yet' defined conditions, in a 'qualifier' statement, in [*square brackets*], like this:

Merck Core Purpose:
S3: Scale: Estimated Better Days for defined [**Life Form**, default: Humans] as a direct result of defined [**Products**, Default: Pharmaceuticals].
Past *[2014, Europe, Products = Tranquilizers, Life Form = {Humans, Animals}]* 100.
Goal *[2024, USA, Products = All Merck Products and Services]* 1,000.

The [qualifier] statement enables us to be more specific. And since we can have many such statements (many Goals, Many Pasts) about different interesting levels of performance, we can plan for a both ***complex*** enterprise (many connected parts), and ***complicated*** enterprise (difficult to predict, estimate and understand, especially with respect to its environment).

This avoids vagueness, over-simplification, misunderstandings, and over-generalization. We can be as clear, exact, and specific as is useful, at a given stage of planning.

For example:

EXAMPLE

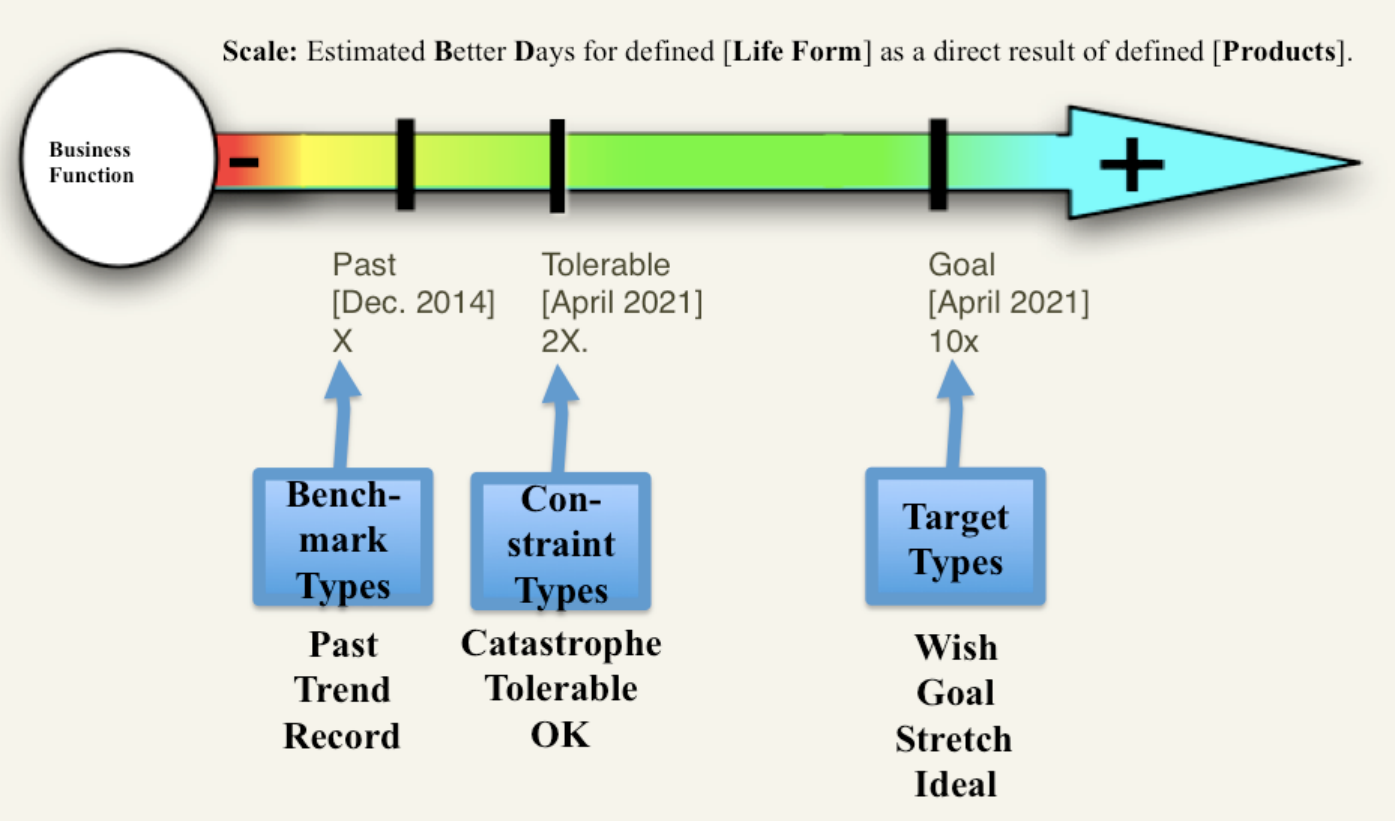


Figure 7. Specific instances of each type of planning point on a scale.

Merck Core Purpose:
S3: Scale: Estimated Better Days for defined [Life Form, default: Humans] as a direct result of defined [Product, Default: All].
Past [2014, Europe, Products = Tranquilizers] 100.
Goal [2024, USA, Products = All Merck Products and Services] 1,000.
Goal [2020, Worldwide, Products = All Merck Products and Services, If Merge Approved] 500.
Tolerable [2020, Europe, Products = Pharmaceuticals] 200±100? <- CEO Vision Statement.

Any useful number of points on the business performance scale (in this case a 'Scale' for the 'top level performance' core purpose) can be defined, using any interesting set of the types of points (Past, Goal, Tolerable, etc.). And any set, or combination of, [qualifier conditions], a sort of 'adjectives', can be planned, in addition to any one of these scale points.

It is obvious that the qualifier conditions permit 'drilling down' into detail of plans laid, later, and at sub-levels (for example year by year, and country by country).

Notice that the Scale definition is being 'reused' (write once, use many times), by all these scale points.

It therefore becomes more obvious, why we take pains to be precise in defining a Scale (using 10 words, rather than one), and why we parameterize it ('defined [Life Form]').

Clarifying Objectives

It is central that objectives are perfectly understood, by all intended readers. All employees, investors, media etc.

Perfect practical clarity is a nearly attainable objective, using fairly simple means.

It is unacceptable, (a bad practice we have measured worldwide) for objectives to be so badly written, that employees and managers judge 30% to 70% of all specification words they read, to be either ambiguous or unclear (to at least some of the potential readership).

The astute reader will already have noticed some of the devices we use to reduce ambiguity (Scale + Goal for example, beats 'exceedingly'). Many other devices in the Planning Language are not yet explained or mentioned. But they are available when you are ready.

Here are some clarification practices that have already appeared in the examples above:

1. Consistent official definition of key planning parameters (like Past, Goal, Scale). A formal Glossary exists [Planguage]
2. Our drive to become numeric (beats nice words like 'substantially improved')
3. The use of qualifiers to define 'when', 'where' and other conditions. "[2020, UK, If Finance Approved]"
4. The consistent formal use of terms written with **C**apital **L**etters, indicating that the term is formally defined. Like: **Life Form**: {Human, Animal, Plant} in the initial S3 example above. And like "**P**ast, **S**cale, and **Merck Core Purpose**:".
5. There are dozens more devices, you can choose to improve clarity, too many to enumerate here.

Your practical 'organizational planning-improvement campaign', we have found, should be 'to reduce 'major defects' (avoid planning specification terms that might possibly cause misinterpretation, of serious consequence, by some reader) in planning. The degree of improvement should be from a 'normal', but unacceptable, level of 30% or more, to a level of less than 1 per 300 words. A tough but do-able objective.

Extending Understanding of the Objectives – Background

In addition to the specification devices mentioned above ("Clarifying Objectives"), we have developed a large set of simple devices for adding background information to a fundamental objective.

It consists of a predefined set of 'parameters' (Scale and Past are 'parameters' too), and other Planguage [2] devices; as well as the ability to define any new additional parameters you find useful.

We already inserted some background information in the example:
±200 means the tolerable range is 200 days (500 to 900 days)

Tolerable [2020, Europe, Products = Pharmas] 700 ±200 ? <- CEO Vision Statement.

EXAMPLE

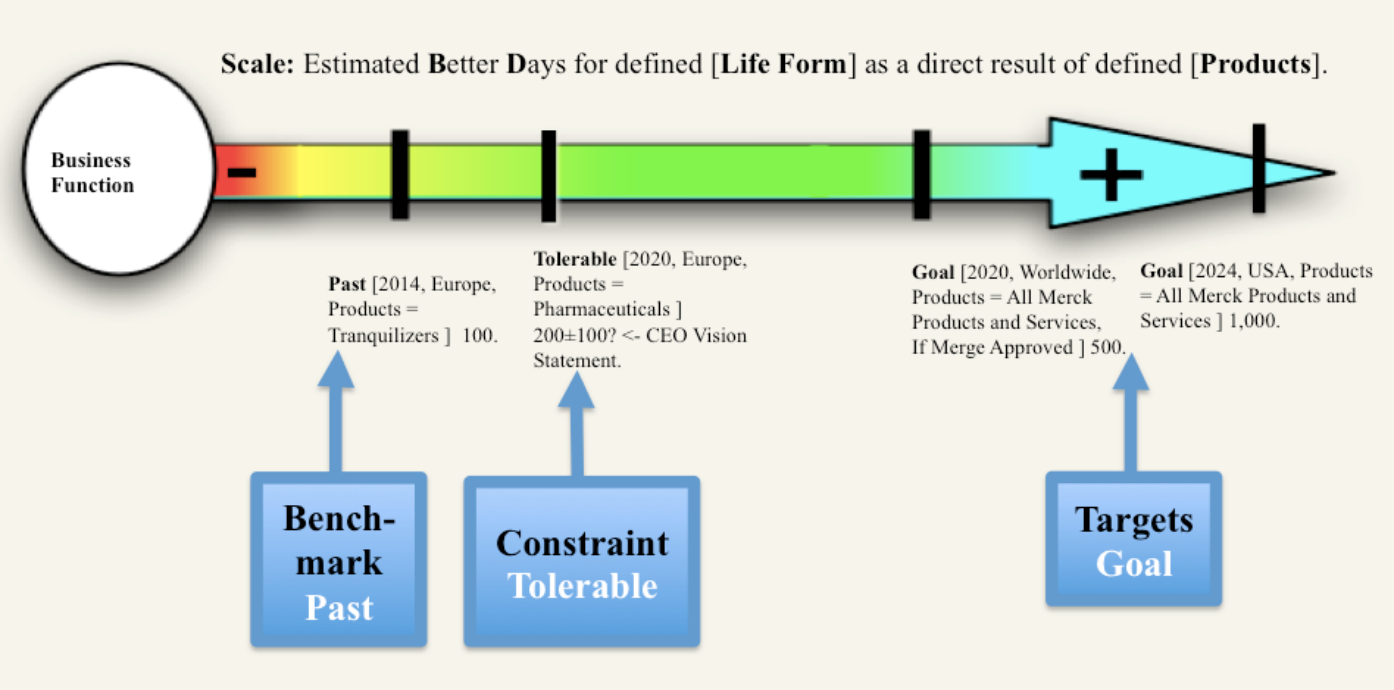


Figure 8. Simple examples of defining interesting planning points along a Scale.

? means even this is a questionable number or interpretation. Don't take it too seriously. Uncertainty.

<- is a 'Source' arrow, used to specify our source of the specification. In this case the 'CEO Vision Statement'.

We could have also written:
Here is a small sample of some of the other available background statements (with illustrative text after the parameter):

Tolerable [2020, Europe, Products = Pharmas] 700.
Range: ±200 days
Risk: incorrect interpretation of actual CEO slide 25.
Source: CEO Vision Statement, Jan 1 20xx Brussels.

With all the statements you might want to use, you can easily fill a page, or a slide, with 20 to 60 statements for a single defined objective. It is up to you, to use or create, what you find valuable to add, as background to the core specification. The full specification, for a single objective, forms a small collection ('database') of 'everything' we need to know, in relation to that objective. Of course, subsets, right down to one-liners, can be extracted for

Supports: Core Purpose
Supported By: Top Long-Range Objectives
Constrained By: Core Values
Implementation Responsibility: CEO
Plan Owner: Strategic Planning office

specific presentation purposes, while other statements can be drilled-down to, on demand.

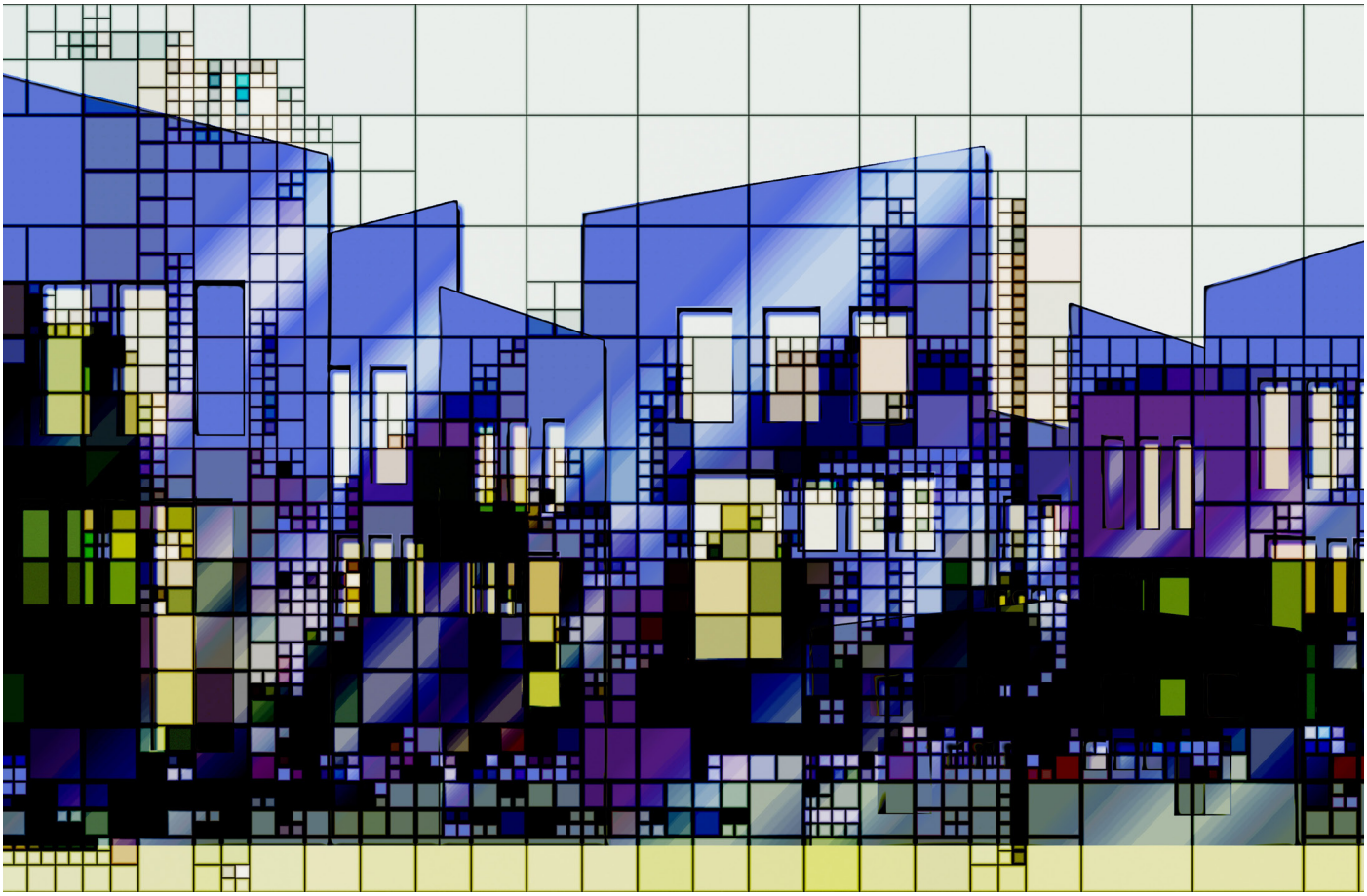
Once when we were having a top management fight in London, about using this format (the Marketing Guys wanted to keep it simple and unintelligible) a seasoned director stopped the show by saying:

> *I have estimated that we spend on average £200,000 for each one of these objectives, and too frequently screw things up. If defining an objective in 40 lines instead of one line solves that problem, then that is a small price to pay, and a necessary investment in getting our business right" (Thanks BW).*

If you use the simple principle of investing more effort, in management planning quality, only if it pays off, you should not end up with unnecessary bureaucracy.

I know we have too much meaningless low-quality verbiage in planning, everywhere, today. My suggestion is, in fact, to write less in total, and to make it 'reusable'; and to raise the quality of what we do write - by two orders of magnitude. Get rid of those many major defects per page.

This 'specification quality improvement' is measurable using the methods immediately below [Chapter 2 of Vision Engineering].



AUTHOR

Tom Gilb

Tom Gilb was born in Pasadena in 1940, emigrated to London 1956, and to Norway 1958, where he joined IBM for 5 years, and where he resides, and works, when not traveling extensively.

He has mainly worked within the software engineering community, but since 1983 with Corporate Top Management problems, and since 1988 with large-scale systems engineering (Aircraft, Telecoms and Electronics).

He is an independent teacher, consultant and writer. He has published nine books, including the early coining of the term “Software Metrics” (1976) which is the recognized foundation ideas for IBM CMM/SEI CMM/CMMI Level 4.

He wrote “Principles of Software Engineering Management” (1988, in 2006 in 20th printing), and “Software Inspection” (1993, about 14th printing). Both titles are really systems engineering books in software disguise. His latest book is ‘Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Management Using Planguage’, published by Elsevier, Summer 2005.

He is a frequent keynote speaker, invited speaker, panelist, and tutorial speaker at international conferences.

He consults and teaches in partnership with his son Kai Gilb, worldwide. He happily contributes teaching and consulting pro bono to developing countries (India, China, Russia for example), to Defense Organizations (UK, USA, Norway, NATO) and to charities (Norwegian Christian Aid and others).

He enjoys giving time to anyone, especially students, writers, consultants and teachers, who are interested in his ideas - or who have some good ideas of their own. He is a member of INCOSE (www.incose.org).

His methods are widely and officially adopted by many organizations such as IBM, Nokia, Ericsson, HP, Intel, Citigroup - and many other large and small organizations.

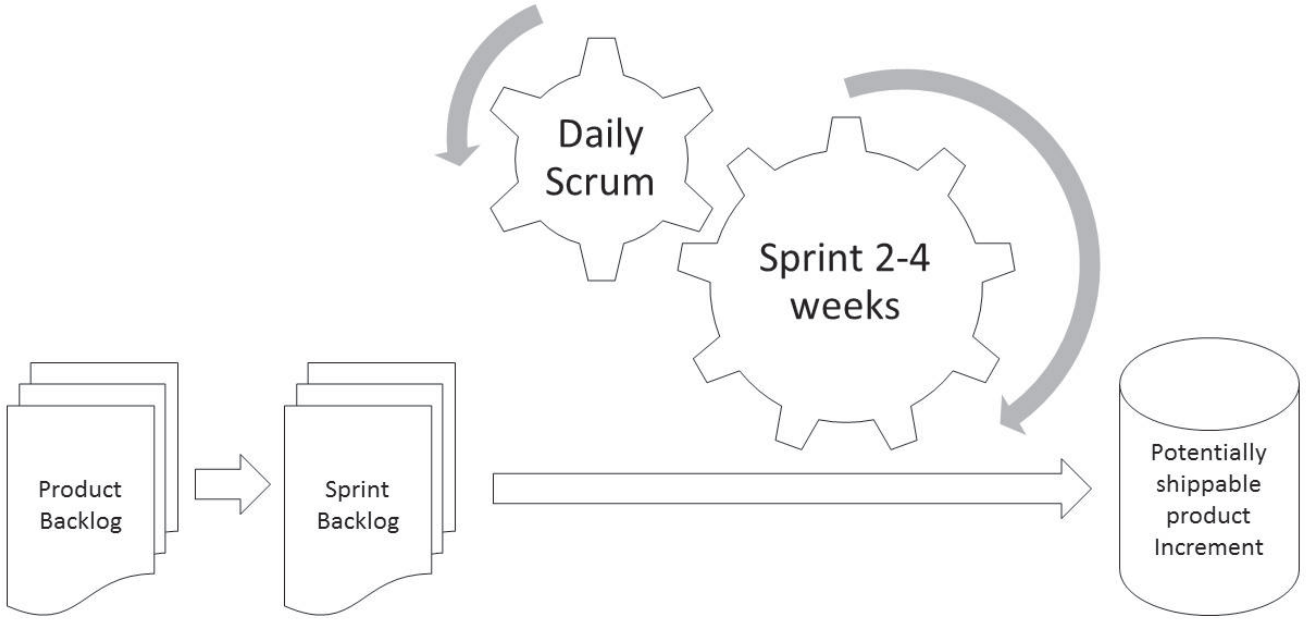
Website: www.gilb.com

Marina Gil-Santamaria

Agile and Scrum Methodologies from a Testing/QA Perspective

Abstract

Agile software development is already beyond the innovation stage and rapidly moving through an early adoption stage. Have you noticed agile and Scrum being mentioned "everywhere" you look? This write-up will describe key agile/Scrum concepts, the different phases of an agile project managed using Scrum, and the top three things that you should expect as a QA engineer/tester professional. If your organization is looking at agile/Scrum, or you want to keep up-to-date on industry trends, read on.



Introduction

Agile software development is a methodology for undertaking software development projects in which incremental functionality is released in smaller cycles, and work is performed in a highly collaborative manner by self-organizing teams that embrace and adapt changes to ensure that customer's needs are truly met. Agile Software Development is not new, in fact it was introduced in the 1990s as a way to reduce costs, minimize risks and ensure that the final product is truly what customers requested. The idea behind the Agile approach is that instead of building a release that is huge in functionality (and often late to market), an organization would adapt to dynamic changing conditions by breaking a release into smaller shorter cycles of 1 to 6 weeks. Each cycle is called an iteration, or sprint, and it's almost like a miniature software project of its own, because it includes all of the tasks necessary to release the incremental new functionality. In theory, at the end of each sprint, the product should be ready for a GA release. Agile methodology emphasizes real-time communication, preferably face-to-face, versus written documents and rigid processes. In addition, one of the most broadly applicable techniques introduced by the agile processes is to express product requirements in the form of user stories. Each user story has various fields including an "actor", a "goal" or task that they need to perform, an explanation

Most agile teams include all the people necessary to release software. At a minimum, this includes programmers and the group or team they are developing the application for, often referred to as their "customers" (customers are the people who define the product; they may be product managers, business analysts, or actual customers). Typically an agile team will also include a ScrumMaster, testers, interaction designers, technical writers, and managers.

What is scrum ? Scrum is really a project management methodology to facilitate agile software development, and enable the creation of self-organizing agile teams. A ScrumMaster is like a traditional project manager in the sense that he/she oversees the centralization of team communication, requirements, schedules and progress. But it is also very different because his/her main responsibility is to facilitate team communications and provide guidance and coaching, while removing impediments to the ability of the team to deliver its goals. Unlike a traditional project manager, the ScrumMaster doesn't direct the team, because an agile team is based on the philosophy that a team member is committed to the other team members, not to a management authority.

Phases of an Agile Development Project using Scrum

Agile can be customized to fit each corporation in terms of size, iteration time, experience, etc, but typically an agile project will have these phases and milestones.

1. **Kickoff meeting**. Although this may seem routine for any project, with an agile development project this is a key element for getting the project launched. The goal of this meeting is to get everybody on the team together to review the product backlog (which is the

master list of all requirements desired in the product that the product owner has drafted in the form of user stories), as well as the user personas (or the profile of each type of product user). In my opinion, this is a nicer and clearer way to introduce product requirements, because you really have more visibility into who is using the product, what are they trying to achieve and why, right from the beginning. A kickoff meeting usually lasts at least half a day with everybody together going over a "story writing workshop" -- in which stories are selected and then decomposed into programmable tasks and written in a white board together along with the time estimates for completion. If you have never seen a product backlog, you can check few samples here in QAZone. Sometime real customers are invited to the kickoff meeting as well to review and clarify the product backlog with the agile team.

2. The next step in the **sprint/iteration planning**, in which the team collectively decides the sprint goal and sprint backlog (list of prioritized work to be done for that particular sprint). While the team is collectively creating the sprint backlog, stories need to be broken into either sub-stories or smaller tasks. During this collective team exercise you can really see the differences in project management (at least if you have a more rigid and formal **waterfall** like type of background), because there is no management authority that assigns tasks to team members. On an agile team all the members jointly associate level of difficulty to specific tasks, they can remove or add additional stories and/or tasks, and tasks are distributed among the team on a per volunteers basis. Unlike a traditional project manager function, the ScrumMaster role in this meeting is to maintain the backlog list in the meeting based on team feedback and consensus, make sure that nobody is volunteering for too many tasks to the point of overload, and facilitate the process of building personal commitment to the team.
3. Now that the Sprint planning is ready in the form of sprint backlog – which is dynamic and not set in stone, in fact it is very likely that it will adapt and change based on new stories, new tasks and/or impediments found throughout the iteration – **scrum meetings** will be set at the same time and in the same place on a daily basis. If you have never attended a scrum meeting, these meetings are very dynamic in nature and fast, never more than 30 minutes, and ideally 10-15 minutes. The objective is to go around so each team member can answer 3 questions: what have I accomplished since the last meeting (developed, tested, written, etc), what will I be working on next, and what are the problems, if any, preventing me from accomplishing my goals. These meetings are very important to make sure that the team moves towards achieving their sprint goal, or adapts/evolves and changes priorities and tasks as needed if new stories, impediments or new scenarios are encountered.
4. At the end of the sprint or iteration, usually a **final acceptance meeting** takes place, which is typically done by presenting what the team has accomplished, and by delivering a demo to the customer or to a larger audience.
5. At the end of an iteration there is also a **sprint retro meeting**, similar to a postmortem meeting at the end of other traditional projects, so the team gets

together to evaluate what worked well, and what needs to be improved during their next iteration.

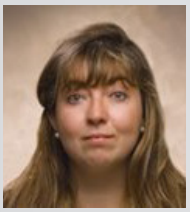
Top 3 things a QA professional should expect when an organization adopts Agile/Scrum development techniques

Agile and Scrum are really changing the way testing is perceived throughout the project. Testing is not a phase at the end; it really is integrated throughout the entire iteration cycle, and it goes hand in hand with programming tasks. In my experience, when comparing a testing role performed within an agile project, or when using more rigid and formal approaches, I have found that with agile methodologies there is:

1. **Better communication and more collaboration among QA & development folks.** Gone are the days of "give me requirements" and "I will give you bugs and reports back"...QA folks are involved in the project from the start–along with their development counterparts–and they have access to the same information about product requirements and customer needs at the same time. This participation from the onset, combined with the fact that development and QA are part now of the same agile team, that they get together on a daily basis, and that they have full visibility into the tasks that each other is performing towards the overall success of the sprint, means better and more frequent communication among themselves. In addition, because the entire team meets everyday (development, QA, product management, etc) there are more opportunities for collaboration and more view points towards performing a particular task. Also the traditional "rivalry" that you may find among QA and development is eliminated because there is a single agile team now working to achieve a common goal.
2. **A new "peer to peer" relationship between development and QA personnel.** You should be prepared to "speak up" much more. Agile methodologies are all about building self-organized teams, and the voice of a QA engineer/tester carries the same weight than a developer. Think about it. In the daily scrum meeting each team member gets asked about their accomplishments (testing, developing, writing product documentation, etc), future plans, and obstacles, treating all of the members as equal partners. On an agile team the question of "how are we going to test it", is as important as "how are going to build it". In addition, because testers tend to be exceptionally good at clarifying requirements and identifying alternative scenarios, (especially when they have full visibility into product requirements and customer needs), they provide valuable input on design and architectural decisions throughout the project, right from the beginning. And these contributions translate into more respect and appreciation from their development counterparts.
3. **Looking for ways to optimize testing efforts will be a "must".** You really need to think about automation, and planning and performing your testing efforts very efficiently. With shorter development cycles of typically no more than 6 weeks, and with builds being released all the time, testing efforts really need to be optimized as much as possible, because there is not sep-

AUTHOR

Marina Gil-Santamaria



Marina Gil-Santamaria is the Founder of Blue Arrow Marketing Inc., where she offers technical marketing, project and launch management consulting services for the High Tech Industry. During the last fifteen years Marina has held positions in product management, project management, development, QA, technical marketing and services organizations at CA, Wily, Empirix, Oracle, Gomez, Compuware and Ipswitch. Marina holds an MS in electrical engineering from the Universidad Politecnica de Madrid, Spain.

arate test phase as such. One of the ways to achieve this is by leveraging both Exploratory Testing and Automated Testing throughout the project. Exploratory testing will come very handy when looking for bugs, opportunities to improve, and missing features. So you should plan on "exploring" the product at the beginning of each new sprint, or any time that there is a change done to a product feature within the sprint cycle. Similarly, you will need to plan and build your scripts to perform automated functional and regression testing within the sprint, because there is not enough time for performing thorough manual testing. One of the things to remember is that there are no really lengthy requirement document or specifications–other that the stories encapsulated on the backlog files–so the only way to make sure that each feature is fully developed, tested, and accepted by the product owner before counting it as "DONE!", is by using the sprint backlog as your own test plan (or writing a test case or script for every feature). Some teams are treating test case scenarios as entries that need to be added to the product/sprint backlog files for planning and tracking purposes. Another factor to consider is that development is much more heavily engaged in testing, so you should leverage this, and work very closely with them to plan and build more automated scripts that cover realistic scopes.

Summary

If you enjoy being involved in product decision making, helping to shape how a product looks and works, and working in a collaborative environment that encourages team work and peer to peer relationships with your development counterparts, you will enjoy working on an agile project. On the down side, agile software development can be a little bit intimating at the beginning. Agile is all about embracing and rapidly adapting to changes–which might be hard to accept at the beginning–plus there are new processes, and new communication styles in place, so you might feel a little reluctant about it. However, once you get into the dynamics of agile software development, it can be a very fun and empowering experience!

Article was originally published on:
http://www.stickyminds.com/article/agile-and-scrum-methodologies-testingqa-perspective

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:

Content
http://pixabay.com/

Free for commercial use / No attribution required

Cover

Northern Lights, Sweden, Lapland, Aurora Borealis
http://pixabay.com/
Free for commercial use / No attribution required