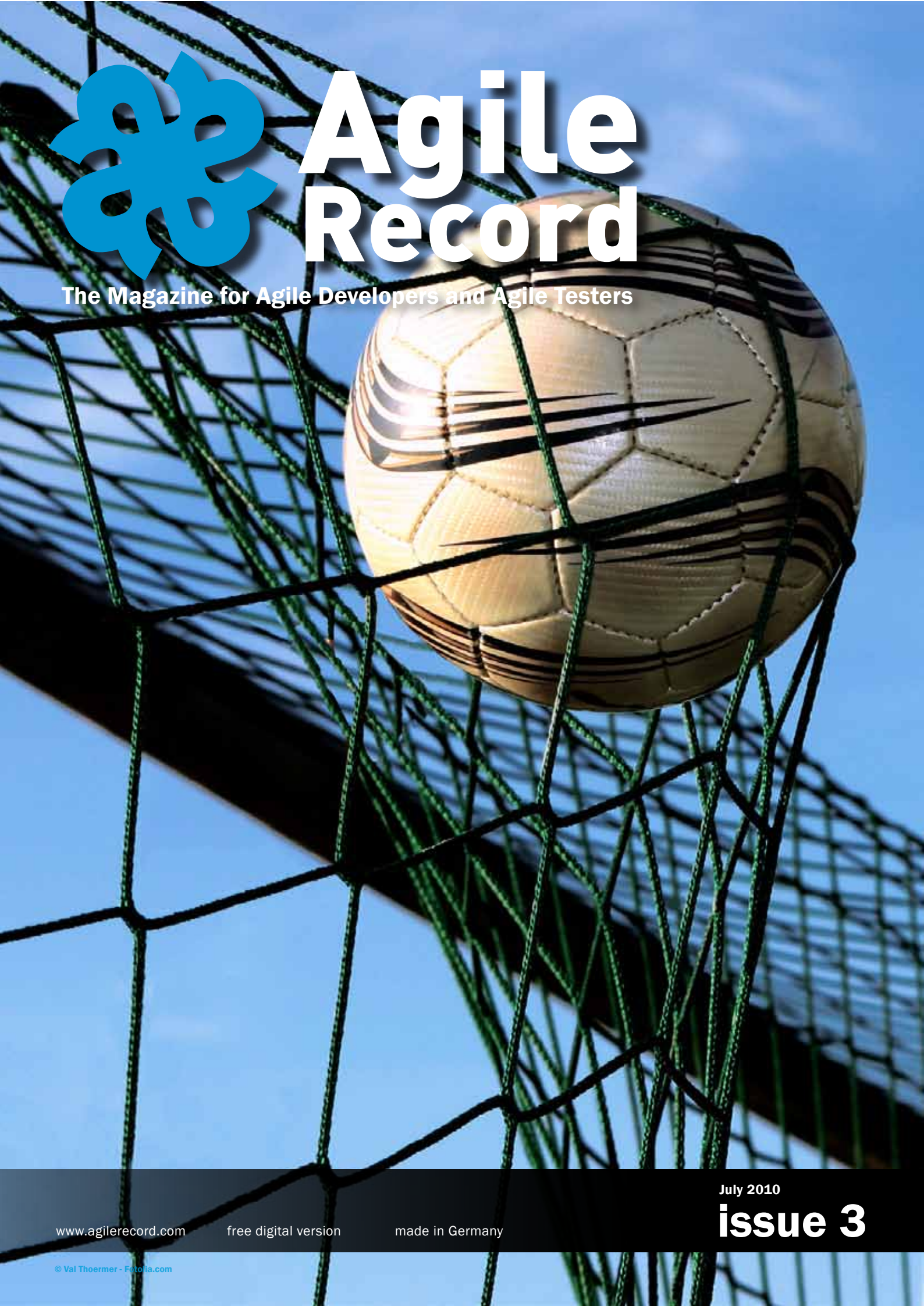
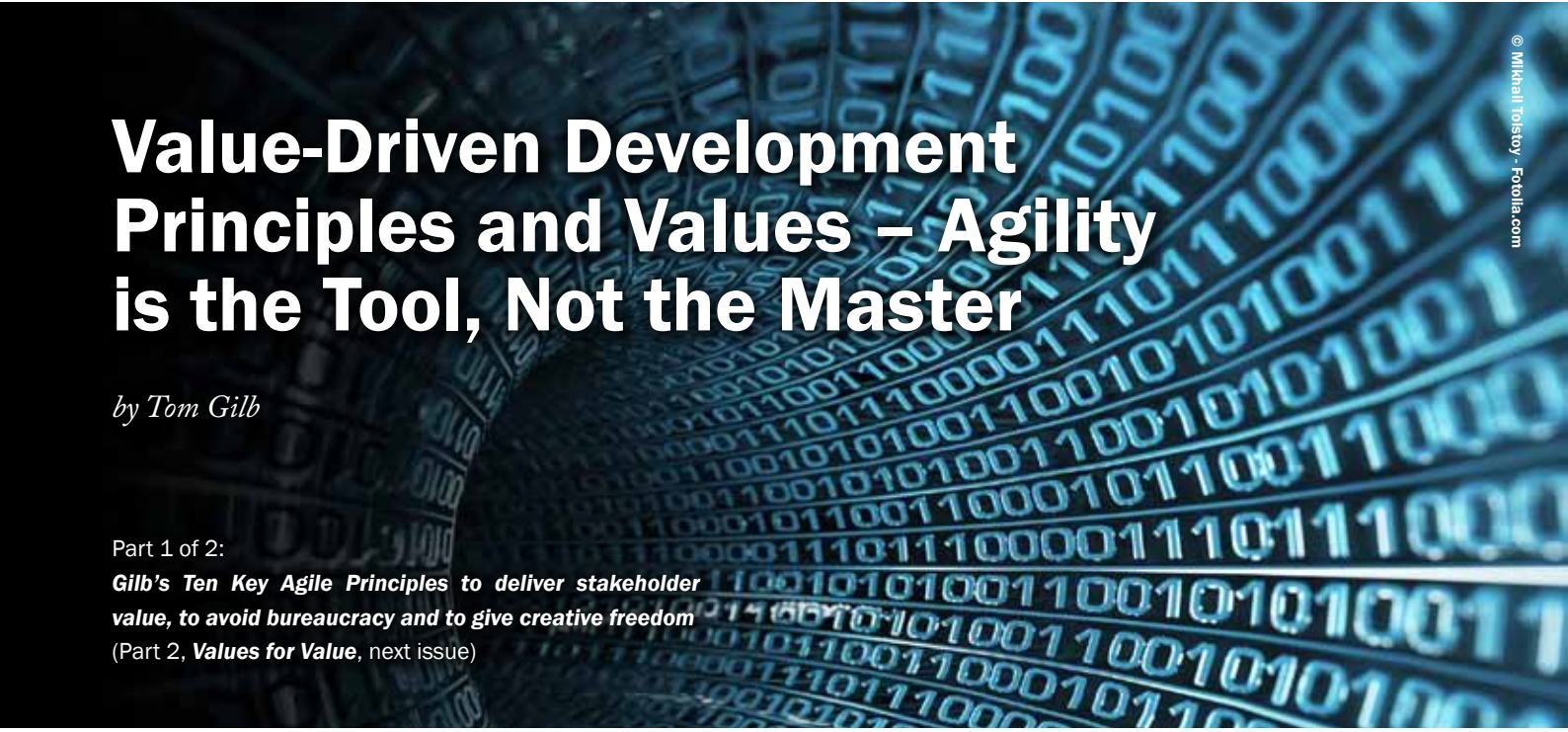


Agile Record

The Magazine for Agile Developers and Agile Testers

July 2010
issue 3

www.agilerecord.com free digital version made in Germany



Value-Driven Development Principles and Values – Agility is the Tool, Not the Master

by Tom Gilb

Part 1 of 2:
Gilb's Ten Key Agile Principles to deliver stakeholder value, to avoid bureaucracy and to give creative freedom
(Part 2, ***Values for Value***, next issue)

Introduction

The Agile Manifesto [2] has its *heart* in the right place, I am only worried about its *'mind'*. And its first principle *"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software"*, is central to the ideas in this article. It is not strange that I agree, since many of the Agilistas, for example Kent Beck, explicitly point to my 1988 book as a source of some of their ideas [4, 5, 6, 7, 8, 9, 12].

My problem with agile is not in its *ideals*, but in the everyday *teaching* and *practices* we see. What we see is the same problem that has been afflicting all software and IT projects long before agile appeared. Our technocratic culture has forgotten our stakeholders and their values.

The practices are far too 'programmer centric', and far too little 'stakeholder value' centric. The result is that 'working software' [2] *is* delivered to a *'customer'* [2].

However, necessary *values* are not necessarily, and all too rarely, *delivered* to *all critical stakeholders*. Code has no value in itself. We can deliver bug-free code, that has little or none of the anticipated value. We can deliver software functions, as defined in requirements, to the 'customer' – but totally fail to deliver critical *value* to many critical *stakeholders*.

The Opera ticket system at the magnificent new Oslo Opera House is a simple example. In the old days, a ticket was ripped off a stack and immediately delivered to the ticket buyer. The new system, at opening, took 10–20 minutes for the ticket sellers to negotiate what to do and how to do things like senior and child discounts. With frequent access to supervisors, who were not sure of what or how. No bugs, all the functions, but something was catastrophically wrong. 'Working software' – working badly. Thank God, the Agilistas did not design the Opera House itself! That was done by real first-class architects! [3]

I fear this article may not correct the narrow-mindedness of the coder community, and my principles do apply to a higher level of thinking than coding. I am going to try to formulate a much *clearer* set of principles, a more explicit set; and in the subsequent article, I will formulate clearer 'values' than the Agilistas managed to do. I have one decided advantage: I am not subject to lowest common denominator politics, as they were; I can express my own opinion – unopposed! I hereby give them specific permission to update their wooly and dangerous ideals with these more focussed ideals. And because they won't (chiselled in stone, unagile documentation Manifesto) I give the reader the right to spread it, and update, and improve it, as they like.

'Value Principles'

Principle 1. Control projects by quantified critical-few results. 1 page in total! (Not stories, functions, features, use cases, objects, ...)

Most of our so-called functional requirements are not actually requirements. They are *designs* to meet unarticulated, higher-level, and critical requirements [14]. For example the requirement to have a 'password' is hiding the real 'Security' quality requirement [13]. Most of the really critical project objectives are almost always buried in old management slides, and formulated in a wooly and un-testable way. They are *never* used in architecture, contracting or testing. This is a major cause of project failure [14]. Management and project sponsors are led to believe the project will deliver certain improvements. In practice the agile culture has no mechanism for following up and delivering expected values. Scrum would argue that that is the job of the product owner. However, even top Scrum gurus openly acknowledge that the situation in practice is nowhere near what it should be. We simply do not teach and practice the necessary mechanisms. Software people were always bad at this, but agile did not deliver clear ideals on its own.

Principle 2. Make sure those results are business results, not technical. Align your project with your financial sponsor's interests!

People do not do development projects to get function, features and stories. Yet these seem primary in the current agile methods. We need functions, stories and perhaps 'features' to make sure the application will do the fundamental business activities that are expected (e.g. *'issue an opera ticket', 'give a child discount'*). But these fundamentals are never the *primary* drivers for the investment in a development project. As a rule, the stakeholders already have those functions in place, in current systems. If you look at the project documentation, someone 'sold' management on better systems – some improvements. Faster, cheaper, more reliable etc.

These are always improvements that are specifically stated somewhere, and they are always quantifiable,. Unfortunately, we in agile development avoid being specific *at this level*. We use adjectives like 'better', 'improved', 'enhanced' and leave it at that. We have learned long ago that our customer is too uneducated and too stupid (common sense *should* compensate for lack of education) to challenge us on these points. And they happily pay us a lot of money for worse systems than they already have.

We need to make it part of our development culture to carefully analyze business requirements ('save money'), to carefully analyze stakeholder needs (reduce employee training costs'), to carefully analyze application quality requirements ('vastly better usability'). We need to express these requirements quantitatively. We need to systematically derive stakeholder requirements from the business requirements. We need to derive the application quality requirements from the stakeholder requirements. Then we need to design and architect the systems to deliver the quantified requirement levels on time. We are nowhere near trying to do this in current conventional agile methods. So we consistently fail the business, the stakeholders, and fail to deliver the quality levels required [15].

Let me be clear here. You can do this as the system evolves, and it can be expressed on a single page of quantified top level requirements [examples 14]. So don't try the 'up front bureaucracy' argument on me!

Principle 3. Give developers freedom, to find out *how* to deliver those results.

The worst scenario I can imagine is when we allow real customers, users, and our own salespeople to dictate 'functions and features' to the developers, carefully disguised as 'customer requirements'. Maybe conveyed by our product owners.

If you go slightly below the surface of these false 'requirements' ('means', not 'ends'), you will immediately find that they are not *really* requirements. They are really bad amateur design for the 'real' requirements – implied, but not well defined [17]. I gave one example earlier (a real one, Ohio), where a 'password' was required, but 'security' (the *real* requirement) was not at all defined.

We are so bad at this, that you can safely assume that almost all so-called requirements are not real requirements, they are bad designs. All you have to do to see this is ask: *WHY?* Why 'password'? (*Security*, stupid!) – Oh! Where is the security requirement? Not there, or worse, stated in management slides as 'state-of-the-art security' – and then left to total amateurs to design it in!

Imagine if Test Driven Development (TDD) actually tested the quality levels, like the security levels, to start with? Far from it; and TDD is another disappointment in the agile kitbag.

I analyze real requirements about once a week, internationally, and find very few exceptions – i.e. situations where the real requirements are defined, quantified, and then designed (engineered, architected) towards. Agile culture has no notion of real engineering at all. Softcrafting [4], sure. But not engineering – totally alien.

You cannot design correctly towards a vague requirement ('better security'). How do I know if a password is a good design? If the security requirements are clear and quantified (and I simplify!) like "Less than 1% chance that expert hackers can penetrate the system within 1 hour of effort", then we can have an intelligent discussion about the 4-digit pin code that some think is an OK password.

I have one client (Confirmit, [16]) who pointedly refuses to accept functions and features requirements from any customer or salesperson. They focus on a few critical product qualities (like usability, intuitiveness) and let their developers engineer technical solutions to measurably meet the quantified quality requirements.

This gets the right job (design) done by the right people (developers) towards the right requirements (higher level overall views of the qualities of the application). They even do their 'refactoring' by iterating towards a set of long-term quality requirements regarding maintainability, and testability. Probably just a coincidence that their leaders have real engineering degrees?

Principle 4. Estimate the impacts of your designs, on your quantified goals.

I take *quantified* improvement requirements for granted. So do engineers. Agilistas do not seem to have heard of the 'quantified quality' concept. This means they cannot deal with specific, or 'high', quality levels.

The concept of 'design' also seems alien. The only mention of design or architecture in the Agile Manifesto is *"The best architectures, requirements, and designs emerge from self-organizing teams."* [2]. There is some merit in this idea. But, the Agile view

on architecture and design is missing most all essential ideas of real engineering and architecture [18].

We have to design and architect with regard to many stakeholders, many quality and performance objectives, many constraints, many conflicting priorities. We have to do so in an ongoing evolutionary sea of changes with regard to all requirements, all stakeholders, all priorities, and all potential architectures. Simply pointing to 'self-organizing teams' is a method falling far short of necessary basic concepts of how to architect and engineer complex, large-scale critical systems. Indeed, even for much smaller systems such as the 13-developer Confirmit system [16].

Any proposed design or architecture must be compared numerically, with estimates, then measurements, of how well it meets the multitude of performance and quality requirements; and to what degree it eats up resources, or threatens to violate constraints. I recommend the Impact Estimation table as a basic method for doing this numeric comparison of many designs to many requirements [19, 10, 4]. It has been proven to be consistent with agile ideals and practices, and has been given far better reported results than other methods [16]. If other methods have better results, they are unable to report them convincingly, since they do not deal numerically with the values and qualities of stakeholders.

Sometimes the quantified quality and value requirements are overambitious.

If a designer is unable to estimate the many impacts of a suggested design on our requirements, then the designer is incompetent, and should not be employed. Most software designers are by this definition incompetent. They don't just fail to estimate, they do not even understand their obligation to try!

Principle 5. Select designs with the best value impacts for their costs, do them first.

Assuming we find the assertion above, that we should estimate and measure the potential, and real, impacts of designs and architecture on our requirements, to be common sense. Then I would like to argue that our basic method of deciding 'which designs to adopt' should be based on which ones have the best *value for money*. Scrum, like other methods, focuses narrowly on estimating *effort*. This is not the same as *also* estimating the multiple values contributed to the top ten project objectives (which 'Impact Estimation' does routinely) [19]. It seems strange to me that agile methods understand the *secondary* concept of estimating costs, but never deal with the *primary* concept of estimating value to stakeholders, and to their requirements. There is little point in managing cost, if you cannot first manage value. The deeper problem here is probably not Agile methods, but is a total failure of our business schools to teach managers much more about finance, and nothing about quality and values [20]. If management were awake and balanced, they would demand far more accountability with regard to value delivered by software developers and IT projects. But the development community has long since realized that management was asleep on the job, and lazily taken advantage of it.

Principle 6. Decompose the workflow into weekly (or 2% of budget) time boxes.

At one level, the Agilistas and I agree that dividing up big projects into smaller chunks, of a week or so, is much better than a Waterfall/Big Bang approach [21].

However, I would argue that we need to do more than chunk by 'product owner prioritized requirements'. We need to chunk the *value* flow itself – not just by story/function/use cases. This value chunking is similar to the previous principle of prioritizing the designs of best value/cost. We need to select, next week (next value delivery step to stakeholders) the greatest value we can produce in an arbitrarily small step (our team, working a week). In principle this is what the Scrum Product owner should be doing. But I don't think they are even remotely equipped to do this well. They just do not have the quantified value requirements (above), and the quantified design estimates (above) to make it happen in a logical manner.

One dispute I do not seem to have with Agilistas is about whether you can in fact divide any project into small (2% of budget) delivery steps. I find that you always can, but there are a lot of people out there who are firmly, and falsely, convinced it is impossible [21]. But maybe the dispute will surface when they are confronted with the need to chunk by value, not by function.

Principle 7. Change designs, based on quantified value and cost experience of implementation.

If you get stepwise numeric feedback on the actual delivered value of a design, compared to estimated and perceived value, as is normal at Confirmit [16], then you will occasionally be disappointed with the value achieved. This will give you the opportunity to reconsider your design, or your design implementation, in order to get the value you need, irrespective of your previous lack of understanding. You might even learn that 'coding alone is not enough' to deliver value to stakeholders.

I fear that this realistic insight possibility is largely lost; since the agile methods neither quantify the value required, nor do they quantify the value expected from a step or a design at a given delivery step.

The result is that we get stuck with bad designs until it is too late. To me, that does not seem very 'agile'.

Principle 8. Change requirements based on quantified value and cost experience, new inputs.

Sometimes the quantified quality and value requirements are overambitious. It is too easy to dream of perfection, and impossible to actually get it. It is too easy to dream of great improve-

ment, without being aware of its true cost, or state-of-the-art limitations. Sometimes we have to learn the reality of what we can or should require by practical experience. This is of course normal engineering and science. To learn technical and economic realities step by step.

The agile community, however, has, as we have pointed out, little concept of quantifying any requirements. Consequently, they cannot learn what is realistic. They will just get what they get by chance or custom.

If they actually quantified their key requirements, and if they measured the incremental numeric results, then if requirements were either overambitious or unacceptably costly, we would have a chance to react quickly (agility!). Learning and agile change are threatened by the lack of quantification and measurement in the normal development process. But today's agile community remains unconcerned.

Principle 9. Involve the *stakeholders*, every week, in setting quantified value goals.

Agile methods refer to *users* and *customers*. The terms used are 'sponsors, developers, and users, customers'. In systems engineering (incose.org) there is no doubt that the generic concept is 'stakeholder'. Some parts of software engineering have been adopting a stakeholder paradigm [22]. But agile methods do not mention the concept. In real projects of moderate size, there are 20 to 40 interesting stakeholder roles worth considering. Stakeholders are sources of critical requirements. Microsoft did not worry enough about a stakeholder called the EU – a costly mistake. In every failed project – and we have far too many – you will find a stakeholder problem at the root. Stakeholders have priorities, and their various requirements have different priorities. We have to keep systematic track of these. Sorry, if it requires mental effort. We cannot be lazy and then fail. I doubt if a Scrum product owner is trained or equipped to deal with the richness of stakeholders and their needs In fact, the PO seems to be a dangerous bottleneck in this concern.

However, it can never be a simple matter of analyzing all stakeholders and their needs and the priorities of those needs up front. The fact of actual value delivery on a continuous basis will change needs and priorities. The external environment of stakeholders (politics, competitors, science, economics) will constantly change their priorities, and indeed even change the fact of who the stakeholders are. So we need to keep some kind of line open to the real world, on a continuous basis. We need to try to sense new prioritized requirements as they emerge, in front of earlier winners. It is not enough to think of requirements as simple functions and use cases. The most critical and pervasive requirements are overall system quality requirements, and it is the numeric levels of the 'ilities' that are critical to adjust, so that they are in balance with all other considerations.

A tricky business indeed, but – are we going to really be 'agile'? Then we need to be realistic – and current agile methods are not even recognizing the stakeholder concept. Head in the sand, if you ask me!

Principle 10. Involve the *stakeholders*, every week, in *actually using* value increments.

Finally – the stakeholders are the ones who should get value delivered incrementally, at every increment of development.

I believe that this should be the aim of each increment and not 'delivering working code to customers'. This means you need to recognize exactly which stakeholder type is projected to receive exactly which value improvement, and plan to have them, or a useful subset of them, on hand to get the increment, and evaluate the value delivered. Current agile methods are not set up to do this, and in fact do not seem to care at all about value or stakeholders.

In fact, developers would have to consider the *whole* system, not just the code, in order to deliver real value – and coders feel very uncomfortable with anything outside their narrow domain.

Isn't it is amazing that they have been given so much power by 'managers' to screw up society? ■

Here is an overview of my Agile Values, the subject of a more detailed article in the next issue.

My 10 Agile Values? © Tom Gilb 2004-10

Simplicity	
1.	Focus on real stakeholder values
Communication	
2.	Communicate stakeholder values quantitatively
3.	Estimate expected results and costs in weekly steps and get quantified measurement feedback on your estimates the same week
4.	Install real quantified improvements for real stakeholders weekly
5.	Measure the critical aspects in the improved system weekly
6.	Analyze deviations from value and cost estimates
Courage	
7.	Change plans to reflect weekly quantified learning
8.	Immediately implement the most valued stakeholder needs by next week *Don't wait, don't study (analysis paralysis), don't make excuses. Just do it!*
9.	Tell stakeholders exactly what quantified improvement you will deliver next week
10.	Use any design, strategy, method, process that works well quantitatively in order to get your results *Be a systems engineer, not a just a programmer (a 'Softcrafter'). Do not be limited by your craft background, in serving your paymasters.*

References

1. Gilb's Agile Principles and Values
The draft basis for a full paper. Originally formulated for conference speeches in November 2004 (London, XP Days Keynote, What's Wrong with Agile?). http://xpday4.xpday.org/slides.php Original slides still here. Slides 38-39 the Principles and Values statements. This is why I am copyrighting from 2004. I literally wrote the values and principles at the conference just before my speech, and they first appeared in my slides. I have updated the principles to stress "values, stakeholders and costs" in 2007 and 2010 for this article.

2. Agile Manifesto:
URL http://agilemanifesto.org/principles.html

3. http://en.wikipedia.org/wiki/Oslo_Opera_House

4. Gilb, Principles of Software Engineering management, 1988.
http://books.google.co.uk/books?q=gilb+principles+of+software+engineering+management&spell=1&oi=spell

5. Mike Cohn, "I've always considered Tom to have been the original agilist. In 1989, he wrote about short iterations (each should be no more than 2% of the total project schedule). This was long before the rest of us had it figured out."
http://blog.mountaingoatsoftware.com/?p=77

6. Comment of Kent Beck on Tom Gilb, Principles of Software Engineering Management: " A strong case for evolutionary delivery – small releases, constant refactoring, intense dialog with the customer". (Beck, page 173).
In a mail to Tom, Kent wrote: "I'm glad you and I have some alignment of ideas. I stole enough of yours that I'd be disappointed if we didn't :-), Kent" (2003)

7. Jim Highsmith (an Agile Manifesto signatory) commented: "Two individuals in particular pioneered the evolution of iterative development approached in the 1980's – Barry Boehm with his Spiral Model and Tom Gilb with his Evo model. I drew on Boehm's and Gilb's ideas for early inspiration in developing Adaptive Software Development. ... Gilb has long advocated this more explicit (quantitative) valuation in order to capture the early value and increase ROI" (page 4, July 2004).

8. Ward Cunningham wrote in April 2005: Tom -- Thanks for sharing your work. I hope you find value in ours. I'm also glad that the agile community is paying attention to your work. We know (now) that you were out there ahead of most of us. Best regards. – Ward, http://c2.com

9. Robert C. Martin (Agile Manifesto initial signatory, aka Uncle Bob): "Tom and I talked of many things, and I found myself learning a great deal from him. The item that sticks most prominently in my mind is the definition of progress.", "Tom has invented a planning formalism that he calls Planguage that captures this idea of customer need. I think I'm going to spend some serious time investigating this." from http://www.butunclebob.com/ArticleS.UncleBob.TomGilbVisit

10. Gilb, Competitive Engineering, 2005, http://books.google.co.uk/books?q=gilb+competitive+engineering&btnG=Search+Books

11. Scott Ambler on Amazon reviews, of Competitive Engineering: Tom Gilb, the father of the Evo methodology, shares his practical, real-world experience for enabling effective collaboration between developers, managers, and stakeholders in this book. Although the book describes in detail Planguage, a specification language for systems engineering, the methodological advice alone is worth the price of the book. Evo is one of the truly underappreciated agile methodologies and as a result Gilb's thought-provoking work isn't as well known as it should be, although I suspect that this will change with this book. The book describes effective practices for requirements and design specification that are highly compatible with the principles and practices of Agile Modeling, yet it goes on to address planning activities, quality, and impact estimation. I suspect that this book will prove to be one of the "must read" software development books of 2006.

12. http://leansoftwareengineering.com/2007/12/20/tom-gilbs-evolutionary-delivery-a-great-improvement-over-its-successors/
"But if you really want to take a step up, you should read Tom Gilb. The ideas expressed in Principles of Software Engineering Management aren't quite fully baked into the ADD-sized nuggets that today's developers might be used to, but make no mistake, Gilb's thinking on requirements definition, reliability, design generation, code inspection, and project metrics are beyond most current practice." Corey Ladas

13. Re Security Requirements:
http://www.gilb.com/tiki-download_file.php?fileId=40
A paper on how to quantify security requirements.

14. Top Level Objectives:
http://www.gilb.com/tiki-download_file.php?fileId=180
A handful of real case studies regarding top level project requirements, or lack of them.

15. One example of systematic quantitative analysis and connection of business, stakeholder and quality levels of requirements in the Norwegian Post case study by Kai Gilb. http://www.gilb.com/tiki-download_file.php?fileId=277

16. Confirmit Case: Paper
http://www.gilb.com/tiki-download_file.php?fileId=32
Confirmit case slides:
http://www.gilb.com/tiki-download_file.php?fileId=278

17. Real Requirements: How to find out what the requirements really are, paper.
http://www.gilb.com/tiki-download_file.php?fileId=28

18. Architecture, a View.
http://www.gilb.com/tiki-download_file.php?fileId=47

19. Design Evaluation: Estimating Multiple Critical Performance and Cost Impacts of Designs
http://www.gilb.com/tiki-download_file.php?fileId=58

20. Hopper: The Puritan Gift: Reclaiming the American Dream Amidst Global Financial Chaos', http://www.puritangift.com/
This is not least a direct and deep attack on Business Schools to teach much more than a narrow financial agenda (aka greed), forgetting the broader set of values that lead to long-term financial soundness.
http://twitter.com/puritangift

21. Decomposition:
http://www.gilb.com/tiki-download_file.php?fileId=41

22. Susanne Robertson's several papers regarding stakeholders:
http://www.systemsguild.com/GuildSite/Guild/Articles.html

> About the author

Tom Gilb
(born 1940, California) has lived in UK since 1956, and Norway since 1958. He is the author of 9 published books, including Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage, *2005.*

He has taught and consulted world-wide for decades, including having direct corporate methods-change influence at major corporations such as Intel, HP, IBM, Nokia. He has had documented his founding influence in Agile Culture, especially with the key common idea of iterative development. He coined the term 'Software Metrics' with his 1976 book of that title. He is co-author with Dorothy Graham of the static testing method 'Software Inspection' (1993). He is known for his stimulating and advanced presentations, and for consistently avoiding the oversimplified pop culture that regularly entices immature programmers to waste time and fail on their projects.

More detail at www.gilb.com.