Agile Record

The Magazine for Agile Developers and Agile Testers

October 2010

issue 4

www.agilerecord.com free digital version made in Germany

Values for Value

by Tom Gilb & Lindsey Brodie

Part 2 of 2:
Some Alternative Ideas On Agile Values For Delivering Stakeholder Value
(Part 1, *Value-Driven Development Principles and Values – Agility is the Tool, Not the Master*, last issue)

The Agile Manifesto (Agile Manifesto, 2001) has its heart in the right place, but I worry that its advice doesn't go far enough to really ensure delivery of stakeholder value. For instance, its first principle, "*Our highest priority is to satisfy the customer through early and continuous delivery of valuable software*" focuses on "the customer" rather than the many stakeholders whose views all need consideration. It also places the focus on the delivery of "valuable software" rather than the delivery of "value" itself (If still in doubt about such a focus, the Agile Manifesto itself states "working software"). These are the same problems that have been afflicting all software and IT projects long before agile appeared: too 'programmer-centric'. Code has no value in itself; it is perfectly possible to deliver bug-free code of little value. We can deliver software functions, as defined in the requirements, to the 'customer' – but still totally fail to deliver critical value to the many critical stakeholders.

I should probably at this point mention that I do agree with many of the ideals of the agile community. After all, my 1988 book, 'Principles of Software Engineering Management' (Gilb, 1988) has been recognized as a source for some of their ideas. I also count several of the 'Agilistas' as friends. It is just that what I see happening in everyday agile practices leads me to believe a more explicit formulation is needed. So in this article, I set out my set of values – modified from the Agile values - and provide ten associated guidelines for delivering value. Feel free to update them and improve them as you see the need.

Perhaps a distinction between 'guidelines', 'values' and 'value' is in place. 'Guidelines' or 'principles' provide advice: 'follow this guideline and things will probably turn out better'. 'Values' are deep-seated beliefs of what is right and wrong, and provide guidance as to how to consider, or where to look for value. 'Value' is the potential perceived benefit that will result from some action (for example, the delivery of a requirement) or thing. For example, you might follow the *guideline* of always buying from a known respected source. Your *values* concerning financial affairs and the environment will probably influence what you buy. Your perceived or actual benefits of what you will gain from your purchases (say, more time, lower costs, and increased satisfaction) reflect their *value* to you.

Here then is a summary of my values for building IT systems – agile or not! These values will necessarily mirror to some degree the advice given in the principles set out in an earlier article (Gilb & Brodie, 2010), but I will try to make a useful distinction between them. I consider there are four core values – **simplicity, communication, feedback, and courage**.

Simplicity

1. Focus on delivering real stakeholder value

I believe in simplicity. Some of our software methods, like CMMI (Capability Maturity Model Integrated) have become too complicated. See for example, (CMMI, 2008). Agile is at least a healthy reaction to such extremes. But sometimes the pendulum swings too far in the opposite direction. Einstein was reputed to have said (but nobody can actually prove it! (Calaprice, 2005)), "Things" (like software methods) "should be as simple as possible, but no simpler". My main argument with agile practice today is that we have violated that sentiment. We have oversimplified. The main fault is in the front end to the agile process: the require-

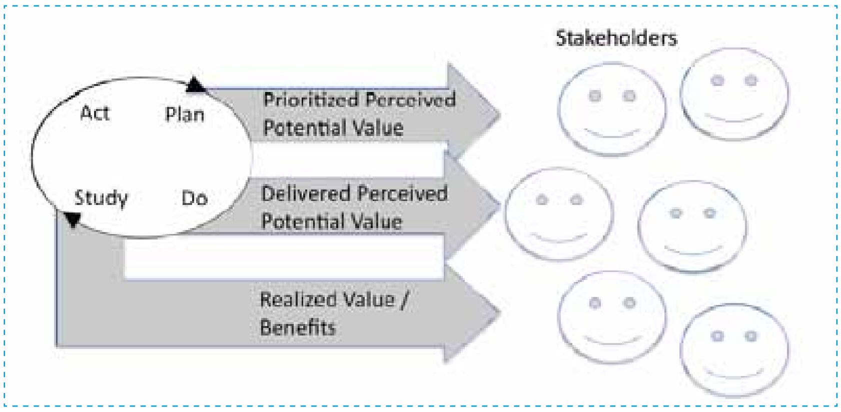

Stakeholders
Act
Plan
Study
Do
Prioritized Perceived Potential Value
Delivered Perceived Potential Value
Realized Value / Benefits

Figure 1. Value to stakeholders: most agile practices today usually fail to identify or clarify all the stakeholders, and their stakeholder value!

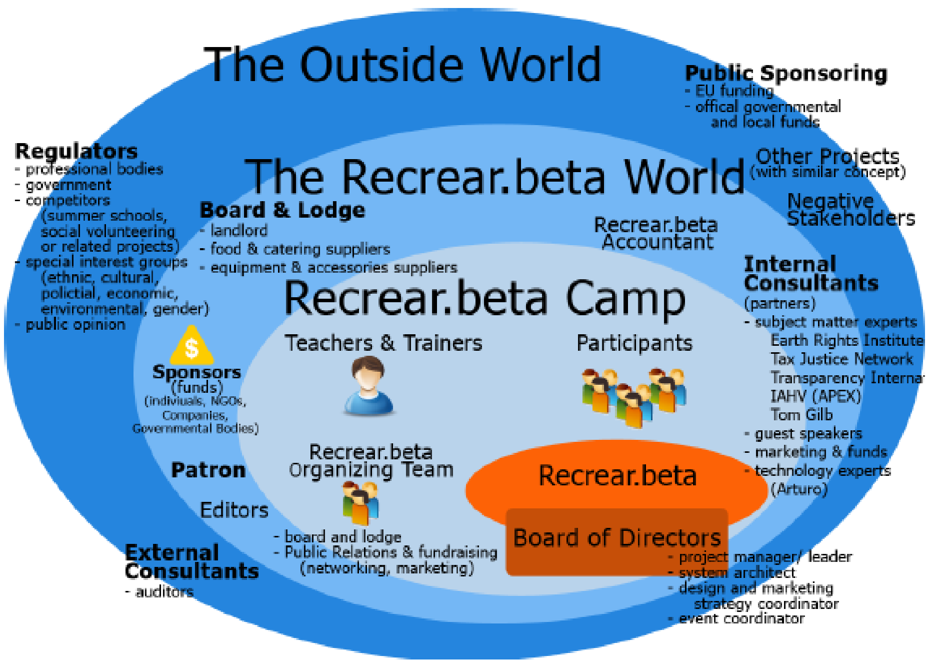


Figure 2. Some examples of stakeholders: the source is re-crear.org, a voluntary-sector client of the author

ments. The current agile practices put far too much emphasis on user, use cases and functions. They say 'value' and they say 'customer', but they do not teach or practice this in a reasonable way for most real projects. They are 'too simple'.
I'll return to discuss this point later, but one of the main failings of the agile process is not recognizing that setting the direction – especially stating the qualities people want and the benefits (the received value) they expect when they invest in an IT system – is key. Iterative and incremental development without such direction is much less effective.

If you want to address this failing, then the simplest thing you can do is to identify and deal with the top few dozen critical stakeholders of your system. To deal with 'the user' and/or 'the customer' only is 'too simple'. The 'top few critical stakeholders' can be brainstormed in less than 30 minutes, and refined during the project, as experience dictates. It is not a heavy 'overhead'. It is one of the necessities for project success.

The next step is to identify the primary and critical quality requirements of each stakeholder. As a rough measure, brainstorming this to get an initial reasonable draft is an hour's work for a small group. For example:
End Users: Easy To Learn, Easy To Use, Difficult to Make Errors, Fast System Response, Reliable.
Financial Admin: Up-to-Date, Accurate, Connectivity to Finance Systems.
IT Maintenance: Easy to Understand, Easy to Repair, Defect-Free.

Note: this is just a start! We need to define the requirements well enough to know if designs will work and if projects are measurably delivering value incrementally! The above 'nice sounding words' are 'too simple' for success. For brevity, I'm not going to explain about identifying scales of measure and setting target quality levels in this paper, see (Gilb, 2005: especially Chapter 5) for further detail.
You can refine the list of quality requirements as experience dictates. You can also often reuse lists of stakeholders, and their known quality requirements in other projects within your domain. Doing this is NOT a heavy project overhead. The argument is that both exercises (identifying the stakeholders and their quality requirements) save time and aid successful project completion. It is part of 'the simplest path to success'. There are, by implication, even simpler paths to failure: just don't worry about all the stakeholders initially – but they will 'get you' later.

Communication
Now we come to my second value, communication. I am sure we all believe in 'good communication', and I suspect most people are probably under the illusion that 'communication is not perfect, but it is pretty good, maybe good enough'. However, my experience worldwide in the IT/software industry is that communication is typically poor.

2. Measure the quality of communication quantitatively
I have a simple way of measuring communication that never fails to surprise managers and technical people alike. I use a simple (5 to 30 minutes) specification quality control (SQC) exercise, on 'good requirements' of their choice. See (Gilb & Graham, 1993; Gilb, 2005: Chapter 10) for further detail on this method.
SQC is a really simple way to measure communication. I just ask the participants to look at a selected text of 100 to 300 words. I prefer the 'top level most critical project requirements' (because that will be most dramatic when they are shown to be bad!). I get their agreement to 3 rules:

1. The text (words and phrases) should be **unambiguous to the intended readership**
2. The text should be **clear enough to test** successful delivery

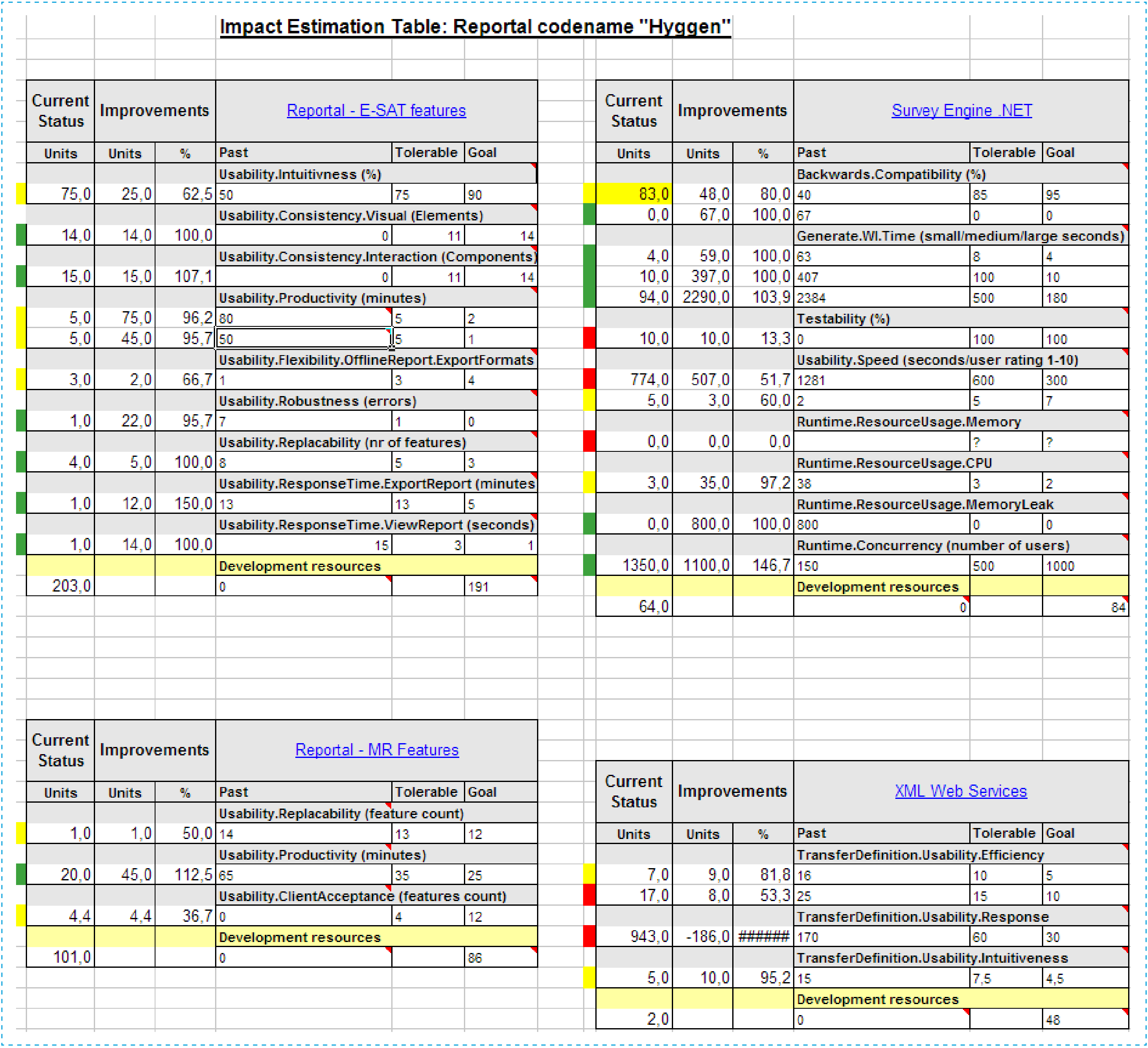
of it.

3. The ‘objectives’ should **not specify proposed designs or architecture** for getting to our objectives.

The participants have to agree that these rules are logically necessary. I then ask them to spend 5 to 30 minutes identifying any words, terms or phrases, which fail these rules. And ask them to count the number of such failures (the ‘specification defects’).
I then collect the number of defects found by each participant. That is in itself enough. In most cases, everyone has found ‘too many’ defects: typically 5 to 40 defects per 100-300 words. So this written communication – though critical - is obviously ‘bad’. Moreover, it gets even more serious when you realize that the best defect finder in a group probably does not find more that 1/6 of the defects actually provably there, and a small team finds only 1/3 of them! (Gilb & Graham, 1993).
The sad thing is that this poor communication is pervasive within IT projects, and clear communication (we can define this as “less than one defect per 300 words potentially remaining, even if unidentified”) is exceptional. Clear communication is in fact only the result of persistent management attention to reducing the defects. One of my clients managed to reduce their level of major defects per page from 82 to 10 in 6 months. The documentation of most IT projects is at about 100-200 defects per page, and many in IT do not even know it.

3. Estimate expected results and costs in weekly steps and get quantified measurement feedback on your estimates the same week

My experience of humans is that they are not good at making estimates for IT systems: for example, at estimating project costs (Gilb, 2010a). In fact, rather than estimating, it is far simpler and more accurate to observe what happens to the cost and quality attributes of actual, real systems as changes are introduced.
One great benefit with evolutionary projects (which include both iterative cycles of delivery and feedback on costs and capability, and the incrementing of system capability) is that we can let the project inform us about what’s actually happening, and we can then relate that to our estimated quality levels and estimated incremental costs: we can learn from unexpected deviation from



Impact Estimation Table: Reportal codename "Hyggen"

	Current Status	Improvements		Reportal - E-SAT features		
	Units	Units	%	Past	Tolerable	Goal
				Usability.Intuitivness (%)		
🟨	75,0	25,0	62,5	50	75	90
				Usability.Consistency.Visual (Elements)		
🟩	14,0	14,0	100,0	0	11	14
				Usability.Consistency.Interaction (Components)		
🟩	15,0	15,0	107,1	0	11	14
				Usability.Productivity (minutes)		
🟨	5,0	75,0	96,2	80	5	2
🟨	5,0	45,0	95,7	50	5	1
				Usability.Flexibility.OfflineReport.ExportFormats		
🟨	3,0	2,0	66,7	1	3	4
				Usability.Robustness (errors)		
🟩	1,0	22,0	95,7	7	1	0
				Usability.Replacability (nr of features)		
🟩	4,0	5,0	100,0	8	5	3
				Usability.ResponseTime.ExportReport (minutes		
🟩	1,0	12,0	150,0	13	13	5
				Usability.ResponseTime.ViewReport (seconds)		
🟩	1,0	14,0	100,0	15	3	1
				Development resources		
	203,0			0		191

	Current Status	Improvements		Survey Engine .NET		
	Units	Units	%	Past	Tolerable	Goal
				Backwards.Compatibility (%)		
🟨	83,0	48,0	80,0	40	85	95
🟩	0,0	67,0	100,0	67	0	0
				Generate.WI.Time (small/medium/large seconds)		
🟩	4,0	59,0	100,0	63	8	4
🟩	10,0	397,0	100,0	407	100	10
🟩	94,0	2290,0	103,9	2384	500	180
				Testability (%)		
🟥	10,0	10,0	13,3	0	100	100
				Usability.Speed (seconds/user rating 1-10)		
🟥	774,0	507,0	51,7	1281	600	300
🟨	5,0	3,0	60,0	2	5	7
				Runtime.ResourceUsage.Memory		
🟥	0,0	0,0	0,0		?	?
				Runtime.ResourceUsage.CPU		
🟨	3,0	35,0	97,2	38	3	2
				Runtime.ResourceUsage.MemoryLeak		
🟩	0,0	800,0	100,0	800	0	0
				Runtime.Concurrency (number of users)		
🟩	1350,0	1100,0	146,7	150	500	1000
				Development resources		
	64,0			0		84

	Current Status	Improvements		Reportal - MR Features		
	Units	Units	%	Past	Tolerable	Goal
				Usability.Replacability (feature count)		
🟨	1,0	1,0	50,0	14	13	12
				Usability.Productivity (minutes)		
🟩	20,0	45,0	112,5	65	35	25
				Usability.ClientAcceptance (features count)		
🟨	4,4	4,4	36,7	0	4	12
				Development resources		
	101,0			0		86

	Current Status	Improvements		XML Web Services		
	Units	Units	%	Past	Tolerable	Goal
				TransferDefinition.Usability.Efficiency		
🟨	7,0	9,0	81,8	16	10	5
🟥	17,0	8,0	53,3	25	15	10
				TransferDefinition.Usability.Response		
🟥	943,0	-186,0	######	170	60	30
				TransferDefinition.Usability.Intuitiveness		
🟨	5,0	10,0	95,2	15	7,5	4,5
				Development resources		
	2,0			0		48

Figure 3. Extract from a case study at Confirmit.

plans how good we are at estimating (Gilb, 2005: Chapter 10). However, in order to support evolutionary project measurement, we have to do better than the typical way of measuring – that is, better than using the rate of user story 'burn-down'. We have to measure the real top-level stakeholder value that is being produced (or not). Yet most IT projects fail to specify upfront what stakeholder value is expected. In such a situation, it is difficult to learn.

To give an example of better communication, see Figure 3, which is an extract from a case study at Confirmit (Johansen & Gilb, 2005). Using the Evo Agile method, 4 small development teams with 13 developers in total worked on a total of 25 top-level critical software product requirements for a 12-week period with weekly delivery cycles. Figure 3 is a snapshot of cycle 9 of 12. If you look at the "%" under "Improvements", you can see that they are on track to meeting the required levels for delivery – which in fact they are very good at doing. This is a better way of tracking project progress than monitoring user story burn-down rates - they are directly tracking delivery of the quality requirements of their stakeholders.

Feedback

4. Install real quantified improvements for real stakeholders, weekly

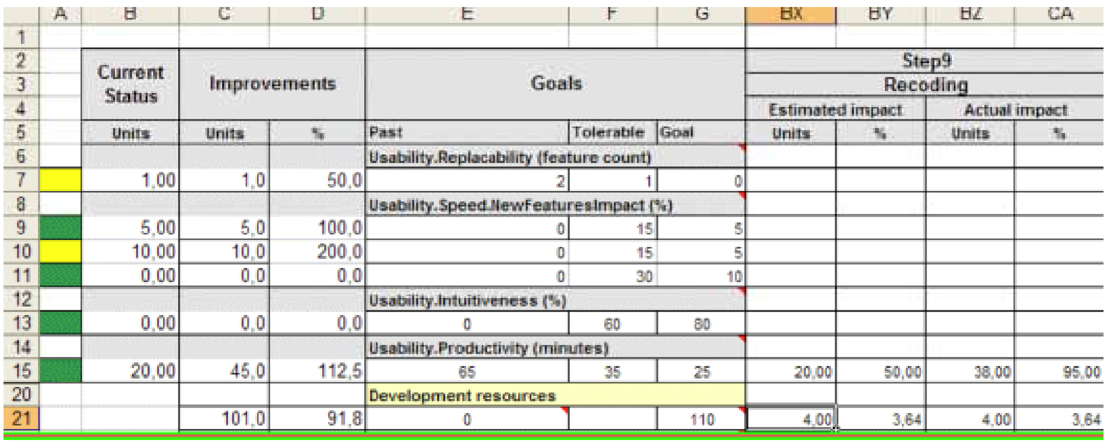
I value getting real results. Tangible benefits that stakeholders want! I value seeing these benefits delivered early and frequently. I have seen one project where user stories and use cases were delivered by an experienced Scrum team, systems development successfully delivered their code, but there was just one 'small' problem - the stakeholder business found that their sales dropped dramatically as soon as the fine new system was delivered (Kai Gilb, 2009). Why? It was taking about 300 seconds for a customer to find the right service supplier. Nobody had tried to manage that aspect. After all, computers are so fast! The problem lay in the total failure to specify the usability requirements quantitatively. For example, there should have been a quality requirement, 'maximum time to find the right supplier will be 30 seconds, and average 10 seconds'. The system needed better requirements specified by the business, not the Scrum team. As it was, the project 'succeeded' and delivered to the wrong requirements: the code was bug-free, but the front end was not sufficiently usable. It was actually a management problem, not a programming problem. It required several levels of management value analysis above the developer level to solve!
Stakeholders do not EVER value function (user stories and use cases) alone. They need suitable quality attributes delivered, too. Traditional agile practice needs to take this on board.
It is also very healthy to prove that you can deliver real value incrementally, not just assume that user stories are sufficient – they are NOT. Such real value delivery means that we must apply total systems thinking: people, hardware, business processes - much more than code.

5. Measure the critical aspects in the improved system, weekly.

Some, in fact most developers seem to never ever measure the critical aspects of their system! And we wonder why our IT system failure rates are notoriously high!
Some developers may carry over to agile a Waterfall method concept of measuring critical quality attributes (such as system performance) only at the end of a series of delivery cycles - before a major handover, or contractual handover. I think we need to measure (test) some of the critical quality attributes every weekly cycle. That is we measure any of the critical quality attributes that we think could have been impacted, and not just the ones we are targeting for improvement in the requirements.
Measurement need not be expensive for short-term cycles. We can use appropriate simplification methods, such as sampling, to give early indications of progress, the order of magnitude of the progress, and any possible negative side effects. This is known as good engineering practice.
The Confirmit project (Johansen & Gilb, 2005), for example, simply decided they would spend no more than 30 minutes per week to get a rough measure of the critical quality attributes. So they measured a few, each week. That worked for them.

6. Analyze deviations from value and cost estimates

The essence of 'feedback' is to learn from the deviation from your expectations. This requires using numbers (quantification) to specify requirements, and it requires measuring numerically, with enough accuracy to sense interesting deviations. To give an example, see Figure 4, which is from the Confirmit case study previously mentioned.



	A	B	C	D	E	F	G	BX	BY	BZ	CA
1											
2		Current Status	Improvements		Goals			Step9			
3								Recoding			
4								Estimated impact		Actual impact	
5		Units	Units	%	Past	Tolerable	Goal	Units	%	Units	%
6					Usability.Replacability (feature count)						
7	🟨	1,00	1,0	50,0	2	1	0				
8					Usability.Speed.NewFeaturesImpact (%)						
9	🟩	5,00	5,0	100,0	0	15	5				
10	🟨	10,00	10,0	200,0	0	15	5				
11	🟩	0,00	0,0	0,0	0	30	10				
12					Usability.Intuitiveness (%)						
13	🟩	0,00	0,0	0,0	0	60	80				
14					Usability.Productivity (minutes)						
15	🟩	20,00	45,0	112,5	65	35	25	20,00	50,00	38,00	95,00
20					Development resources						
21			101,0	91,8	0		110	4,00	3,64	4,00	3,64

Figure 4. Another extract from the Confirmit case study

In this case when the impact of the 'Recoding' design deployed in Step 9 was almost twice as powerful as expected (actual 95% of the requirement level was met as opposed to the 50% that was estimated), the project team was able to stop working on the Productivity attribute and focus their attention for the 3 remaining iterations before international release on the other requirements, like Intuitiveness, which had not yet met their target levels. The weekly measurements were carried out by Microsoft Usability Labs. This feedback improved Confirmit's ability to hit or exceed almost all value targets, almost all the time. I call this 'dynamic prioritization'.

You cannot learn about delivery of the essential stakeholder quality attributes any other way – it has to be numeric. However, numeric feedback is hardly mentioned, and hardly practiced in agile systems development. Instead, we have 'apparent numeracy' by talking about velocity and burn-down rates – these are indirect measures.

All the quality attributes ('-ilities', like reliability, usability, security) or work capacity attributes (throughput, response time, storage capacity) are quantifiable and measurable in practice (Gilb, 2005: Chapter 5), though few developers are trained to understand that about the 'quality' requirements (For example, ask how they measure 'usability').

Courage

Courage is needed to do what is right for the stakeholders, for your organization, and for your project team – even if there are strong pressures (like the deadline) operating to avoid you doing the right thing. Unfortunately, I see few signs of such courage in the current agile environment. Everybody is happy to go along with a weak interpretation of some agile method. Many people don't seem to care enough. If things go too badly – get another job. If millions are wasted – who cares, 'it's not my money'. But if the project money were your money, would you let things continue as they are? Even when your family home is being foreclosed on, and you cannot feed or clothe your children very well, because your project is $1 million over budget?

7. Change plans to reflect quantified learning, weekly

One capability, which is implicit in the basic agile notion, is the ability to change quickly from earlier plans. One easy way to do this is to have no plans at all, but that is a bit extreme for my taste.

The feedback we get numerically and iteratively should be used to attack 'holy cows'. For example, say the directors, or other equally powerful forces in the organization, had agreed that they primarily wanted some particular quantified quality delivered (say, 'Robustness'), and it was clear to you from the feedback that a major architectural idea supported by these directors was not at all delivering on the promise. Courage would be to attack and change the architectural idea.

Of course, one problem is that these same directors are the main culprits in NOT having clear numeric critical objectives for the quality values of the system. The problem is that they are not even trained at Business School to quantify qualities (Hopper & Hopper, 2007), and the situation may be as corrupt or political as described in 'Plundering the Public Sector' (Craig & Brooks, 2006).

In my experience, however, the major problem is closer to the project team, and is not corruption or politics, or even lack of caring. It is sheer ignorance of the simple fact that management must primarily drive projects from a quantified view of the top critical objectives (Gilb, 2008b). Intelligent, but ignorant: they might be 'champions' in the area of financial budgets, but they are 'children' when it comes to specifying quality.

One lesson I have learned, which may surprise most people, is that it seems if you really try to find some value delivery by the second week and every week thereafter, you can do it. No matter what the project size or type. The 'big trick' is that we are NOT constructing a large complex system from scratch. We invariably leverage off of existing systems, even those that are about to be

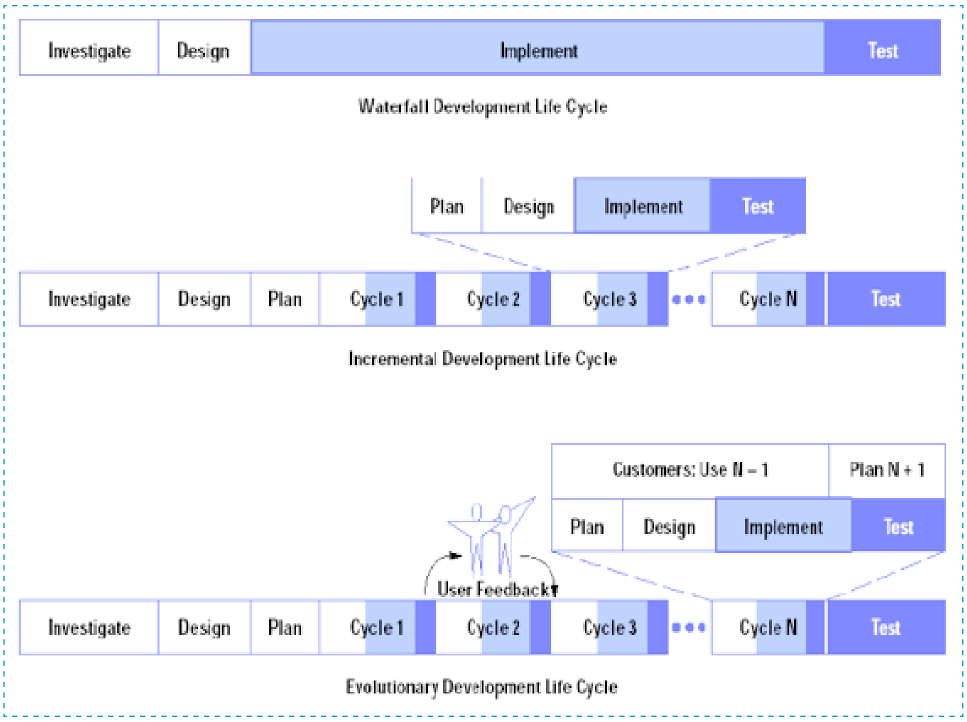


Figure 5. Concepts of weekly delivery cycles with stakeholder feedback. From HP, a client applying the Evo method on a large scale (Cotton 1996; May & Zimmer 1996; Upadhyayula, 2001)

retired, which need improvement. We make use of systematic decomposition principles (Gilb, 2010b; 2008a; 2005: Chapter 10). The big trick is to ignore the 'construction mode' that most developers have, and focus instead on the 'stakeholder value delivery' mode.

Policies for Evo Decomposition

- PP1: Budget: No Evo cycle shall exceed 2% of total budget before delivering measurable results to a real environment.

- PP2: Deadline: No Evo cycle will exceed 2% of total project time (that's one week, for a one-year project) before it demonstrates practical measurable improvement, of the kind you targeted.

- PP3: Priority: Evo cycles which deliver the most 'planned value' to stakeholders, for the 'resources they claim', shall be delivered first, to the stakeholders. Do the juicy bits first!

Figure 6. Evo decomposition policies

See Figure 6 (Gilb, 2010b) for my advice to top managers, when they ask me how they can support deploying the Evo method, and getting rapid results: put in place these decomposition policies as guidance. Demand this practice from your development teams. If they complain, re-train or re-place. No excuses! They will just delay necessary results if not led by management. History is clear.

8. Immediately implement the most-valued stakeholder needs by next week

Don't wait, don't study (analysis paralysis), and don't make excuses. Just do it! This attitude really is courageous. In development environments, where managers are traditionally happy to wait years with no results at all, it takes courage to suggest we should try to start delivering the value stream immediately and continuously. It is rather revolutionary. Yet surely no one would argue it is not desirable?

Part of being courageous is having the courage to say you are sure we will succeed in finding small (weekly) high-value delivery increments. The issue is that most people have no training and no theory for doing this. Most people have never seen it happen in practice. Agile developers have now a widely established practice of delivery of functionality (user stories) in small increments. That is a start, culturally, towards breaking work down into smaller timescales. But as I pointed out earlier (several times!), functions are not the same thing as value delivery to stakeholders.

Assuming you can deliver reasonable value for the effort spent (the costs) - week after week – a surprising thing happens:

- People cease to care about 'the deadline'
- People cease to ask for estimates of the monetary budget
- You are strongly encouraged to keep on going, until value is less than costs
- You end up delivering far more real value than other projects do, well before 'the deadline' (that would have been set, and would have been overrun)
- Management shifts focus from budget and costs to return on investment (ROI)

I sometimes simplify this method by calling it the '1.1.1.1.1.1' method, or maybe we could call it the 'Unity' method:

Plan, in 1 week
To deliver at least 1%
Of at least 1 requirement
To at least 1 real stakeholder
Using at least 1 design idea,
On at least 1 function of the system.

The practical power of this simple idea is amazing. If you really try, and management persists in providing encouragement and support, it almost always works. It sure beats waiting for weeks, months, and years, and 'nothing happens' of any real value for stakeholders.

As a consultant, I always have the courage to propose we do this, and the courage to say I know our team will find a way. Management is at least curious enough to let us try (it costs about a week or two). And it always works. Management does not always actually go for real delivery the second week. There can be political, cultural and contractual constraints, but they get the point that this is predictably doable.

Delivering value to 'customers' is in fact what the agile people have declared they want to do, but in my view they never really took sufficient steps to ensure that. Their expression of value is too implicit, and (of course!) the focus should be on all the stakeholders.

9. Tell stakeholders exactly what quantified improvement you will deliver next week (or at least next release!)

Confirmit used impact estimation (IE) [4, 10, 19] to estimate what value would be delivered the next week (see Figure 3). I think they did not directly tell the affected stakeholders what quality levels they predicted. However, most of the stakeholders got to see the actual delivered results each quarter. And the results were incredibly good. In fact, once Confirmit realized they could continually get such great improvements, they did brag about it numerically on their website!

Since it is quite unpredictable to fully understand what precise quality improvements are going to result and when, it is perhaps foolhardy (rather than courageous) to announce to your stakeholders precisely what they are going to get weekly/fortnightly/monthly in the next cycle. However, based on your understanding of the improvements you are getting each cycle, it is safe to announce what improvements in value you were going to deliver in the next major release!

10. Use any design, strategy, method or process that works well quantitatively in order to get your results

Be a systems engineer, not a just a programmer (a 'softcrafter' (Gilb, 1988)). Have the courage to do whatever it takes to deliver first-class results!

In current agile software practices, the emphasis is on programming, and coding. Design and architecture often mean only the program logic and the application architecture. Agile developers

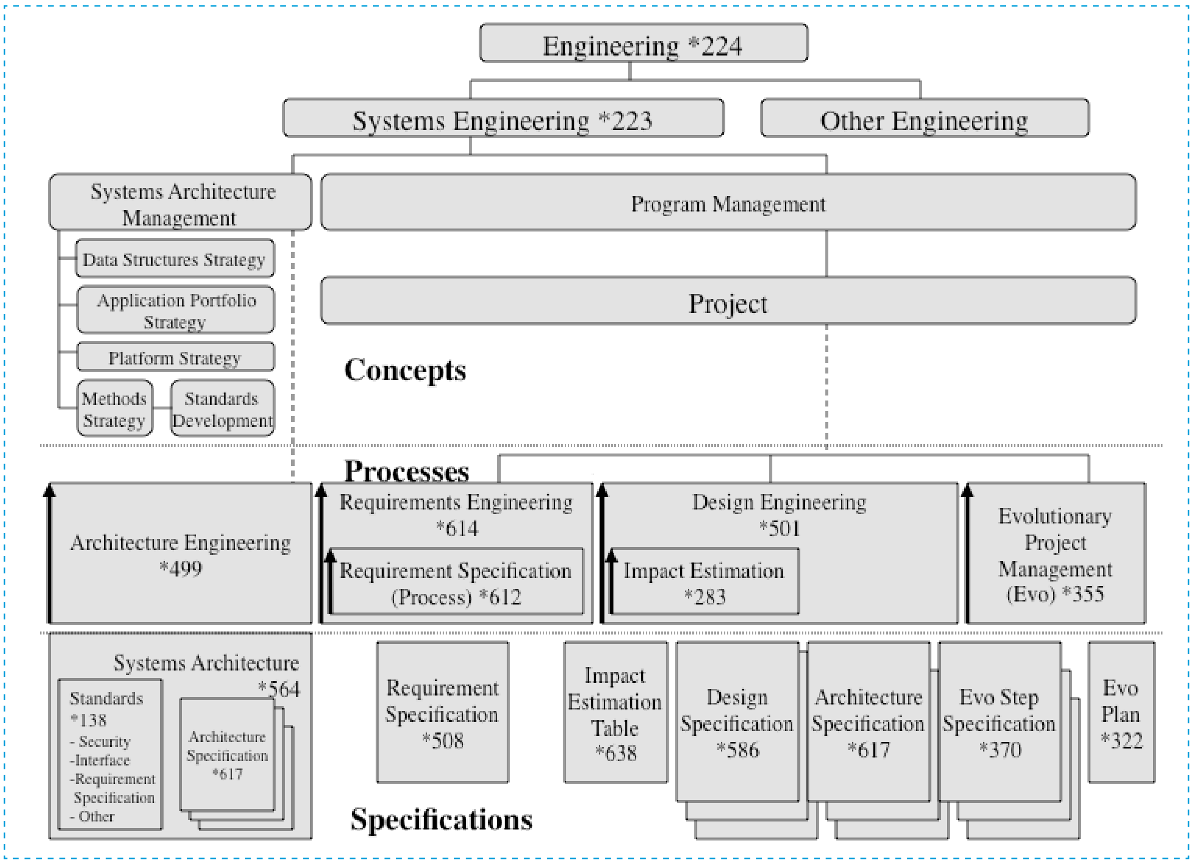

Engineering *224
Systems Engineering *223
Other Engineering
Systems Architecture Management
Program Management
Data Structures Strategy
Application Portfolio Strategy
Platform Strategy
Methods Strategy
Standards Development
Project
Concepts
Processes
Architecture Engineering *499
Requirements Engineering *614
Requirement Specification (Process) *612
Design Engineering *501
Impact Estimation *283
Evolutionary Project Management (Evo) *355
Systems Architecture *564
Standards *138
- Security
-Interface
-Requirement Specification
- Other
Architecture Specification *617
Requirement Specification *508
Impact Estimation Table *638
Design Specification *586
Architecture Specification *617
Evo Step Specification *370
Evo Plan *322
Specifications

Figure 7. A 'Competitive Engineering' view of systems engineering (Gilb, 2005). This shows a set of processes and artifacts needed within systems engineering.

often do not include in their design aspects such as maintenance, system porting, training, motivation, contractual deals, working practices, responsibility, operations and all other elements of a real system. They seem narrowly focused on their code. In fact, as I have discussed earlier, they focus on the code

functionality, and not even the code qualities! Listen to them write, speak, and tweet – it is all about code, user stories and use cases. In order to get competitive results, someone else – a real systems engineer - will have to take over the overall responsibility.

Summary

Agile development embraces much that is good practice: moving to rapid iteration is a 'good thing'. However, it fails to worry sufficiently about setting and monitoring the direction for projects, and instead concentrates on programmer-focused interests, such as use cases and functions. It fails to adequately address multiple stakeholders and achievement of real, measured stakeholder value. Instead it has 'solo' product owners and implicit stakeholder value. Here in this article, I have presented some ideas about what really matters and how agile systems development needs to change to improve project delivery of stakeholder value.

Systems engineering is still a young discipline. The software community has now seen many failed fads come and go over the last 50 years. Maybe, it is time to review what has actually worked. After all, we have many experienced intelligent people: we ought to be able to do better. I think we need to aim to get the IT project failure rate (challenged 44% and total failure 24%) down from about 68% (Standish, 2009) to less than 2%. Do you think that might be managed by my 80th birthday? ■

Acknowledgments

Thanks are due to Lindsey Brodie for editing this article.

References

Alice Calaprice (Editor) (2005) "The New Quotable Einstein", Princeton University Press.

Agile Manifesto (2001). See http://agilemanifesto.org/principles.html [Last Accessed: September 2010].

Todd Cotton (1996) "Evolutionary Fusion: A Customer-Oriented Incremental Life Cycle for Fusion." See http://www.hpl.hp.com/hpjournal/96aug/aug96a3.pdf

Daniel Craig and Richard Brooks (2006) Plundering the Public Sector, Constable.

Kai Gilb (2009) A Norwegian Post case study. See http://www.gilb.com/tikidownload_file.php?fileId=277

Tom Gilb (2010a) Estimation or Control. Draft paper, see http://www.gilb.com/tiki-download_file.php?fileId=433

Tom Gilb (2010b) Decomposition. A set of slides, see http://www.gilb.com/tiki-download_file.php?fileId=350

Tom Gilb (2008a) "Decomposition of Projects: How to Design Small Incremental Steps", Proceedings of INCOSE 2008. See http://www.gilb.com/tiki-download_file.php?fileId=41

Tom Gilb (2008b) “Top Level Critical Project Objectives”. Set of slides, see http://www.gilb.com/tiki-download_file.php?fileId=180

Tom Gilb (2005) Competitive Engineering, Elsevier Butterworth-Heinemann. For Chapter 10, Evolutionary Project Management, see http://www.gilb.com/tiki-download_file.php?fileId=77/ For Chapter 5, Scales of Measure, see http://www.gilb.com/tiki-download_file.php?fileId=26/

Tom Gilb (1988) Principles of Software Engineering Management, Addison-Wesley.

Tom Gilb and Lindsey Brodie (2010) “What’s Fundamentally Wrong? Improving our Approach Towards Capturing Value in Requirements Specification”. See http://www.requirementsnetwork.com/node/2544#attachments [Last Accessed: September 2010].

Tom Gilb and Dorothy Graham (1993) Software Inspection, Addison-Wesley.

CMMI (2008) “CMMI or Agile: Why Not Embrace Both!”, Software Engineering Institute (SEI). See http://www.sei.cmu.edu/pub/documents/08.reports/08tn003.pdf [Last Accessed: September 2010].

Kenneth Hopper and William Hopper (2007) “The Puritan Gift”, I. B. Taurus and Co. Ltd..

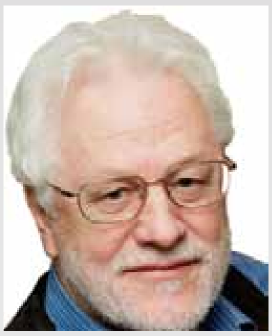
Trond Johansen and Tom Gilb, From Waterfall to Evolutionary Development (Evo): How we created faster, more user-friendly, more productive software products for a multi-national market, Proceedings of INCOSE, 2005. See http://www.gilb.com/tiki-download_file.php?fileId=32

Elaine L. May and Barbara A. Zimmer (1996) “The Evolutionary Development Model for Software”, Hewlett-Packard Journal, August 1996, Vol. 47, No. 4, pages 39-45. See http://www.gilb.com/tiki-download_file.php?fileId=67/

The Standish Group (2009) “Chaos Summary 2009”. See http://www.standishgroup.com/newsroom/chaos_2009.php [Last Accessed: August 2010].

Sharma Upadhyayula (2001) MIT Thesis: “Rapid and Flexible Product Development: An Analysis of Software products at Hewlett Packard and Agilent”. See supadhy@mit.edu. http://www.gilb.com/tiki-download_file.php?fileId=65

> 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.*

Lindsey Brodie

is currently carrying out research on prioritization of stakeholder value, and teaching part-time at Middlesex University. She has an MSc in Information Systems Design from Kingston Polytechnic. Her first degree was Joint Honours Physics and Chemistry from King’s College, London University. Lindsey worked in industry for many years, mainly for ICL. Initially, Lindsey worked on project teams on customer sites (including the Inland Revenue, Barclays Bank, and J. Sainsbury’s) providing technical support and developing customised software for operations. From there, she progressed to product support of mainframe operating systems and data management software: databases, data dictionary and 4th generation applications. Having completed her Masters, she transferred to systems development - writing feasibility studies and user requirements specifications, before working in corporate IT strategy and business process re-engineering. Lindsey has collaborated with Tom Gilb and edited his book, “Competitive Engineering”. She has also co-authored a student textbook, “Successful IT Projects” with Darren Dalcher (National Centre for Project Management). She is a member of the BCS and a Chartered IT Practitioner (CITP).