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

Part 2

"Values for Value"

© by Tom Gilb

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 part 2 article.
It is not strange that I agree with Agile ideas, since many of my good friends, the 'Agilistas', for example Kent Beck, explicitly point to my 1988 book, 'Principles of Software Engineering Management' as a source of some of their ideas [4, 5, 6, 7, 8, 9, 12].
My problem with agile is not in its *ideals (in the Manifesto; although I do argue for a more explicit formulation here)*, but in the everyday *teaching* and *practices* we see. The bad practices we see, are the same problems that have been afflicting all software and IT projects long before agile appeared. Our technocratic culture (aka Nerds) has forgotten our *stakeholders* and their *values*. Maybe we *never* even actually 'remembered' stakeholders and their values?
The agile practices are far too 'programmer centric', and far too little 'stakeholder value' centric. The result is unfortunately that 'working software' [2] *is* delivered to a *'customer'* [2].
However, necessary system *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 which has little or none of the anticipated value. We can deliver software functions, as defined in requirements, to the 'customer' – but still totally fail to deliver critical *value* to many critical *stakeholders*.

I fear, and am sure, this article will not correct the inbred narrow-mindedness of the coder community. My principles and values do apply to a *higher level* of thinking than 'coding', well outside of coders interests. I tried in Part 1 to formulate a much *clearer* set of principles, a more *explicit* set; and in *this* article, Part 2, I will try to formulate clearer 'values' than the Agilistas managed to do. I have one decided, 'unfair' advantage: I am not subject to lowest common denominator agreement politics, as they were; I can express my own opinion – unopposed! I hereby give them, and you all, specific permission to update their woolly and dangerous ideals, with these more focused ideals. And because *they* won't ever change their static creed (chiseled in stone, un-agile Manifesto), I give the reader the right to spread these updated principles and values, and update them , and improve them, as they please and as needed.

Part 1 of 2: was in previous issue
Gilb's Ten Key Agile Principles to deliver stakeholder value, to avoid bureaucracy and to give creative freedom [23, summary of Principles]

This is Part 2, *Values for Value*

My 10 Agile Values? © Tom Gilb 2004-10

Perhaps a distinction between 'principles' and 'values' is in place.

Principles are operational advice: 'follow this principle and things will probably turn out better'. **Values** are deep-seated beliefs of what is right and wrong, what is 'valuable' and 'valued' – or not.

Here then is a summary of my values for building systems – agile or not
The Values will necessarily mirror to some degree the advice given in the Principles. But I will try to make a useful distinction between them.

Here is a summary of my Values. There are four core values – **simplicity, communication, feedback, and courage.** , and the statements below them are intended to explain in more depth exactly what they mean to me – we all have different interpretations of these laudable value sentiments..

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

Feedback
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.

A DETAILED EXPLANATION OF THE AGILE VALUES
Simplicity
1. Focus on real stakeholder values.

I believe in, and value, ***simplicity****.*

I know that some of our software methods, like CMMI (Capability Maturity Model Integrated, ® SEI, http://www.sei.cmu.edu/pub/documents/08.reports/08tn003.pdf) have gotten ridiculously complicated beyond any real usefulness. This in spite of a simple beginning (Radice et al, IBM Systems Journal, vol. 24, no. 2, 1985, my client using some of my methods for level 4, 'quantification of quality', with credit)

Agile is not least a healthy reaction to such insane 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: Einstein Quotations) "Things" (like software methods) "should be as simple as possible, but no simpler". My main argument with agile practice today is that we violated that sentiment. We have oversimplified. We have gotten rid of some excellent and necessary practices – especially regarding stakeholder values – especially about the qualities people want and expect when they invest in an IT system. The main fault is in the front end to the process, the requirements. The current agile practices put far too much emphasis on user, use cases, function. They SAY 'value', 'they say 'customer'. But they do not teach or practice this in a reasonable way for most real projects. They are 'too simple'.

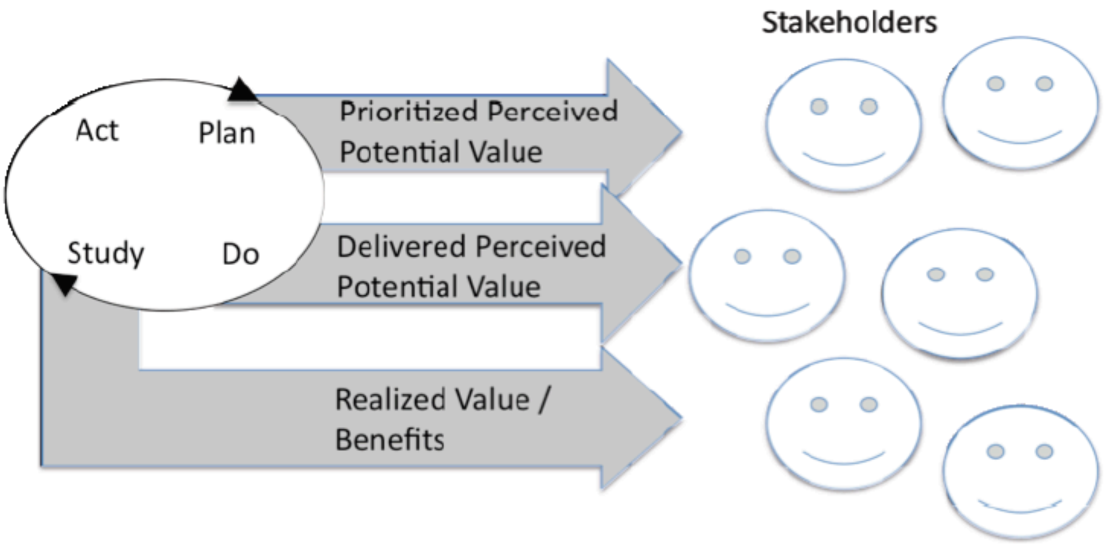

Stakeholders
Act
Plan
Study
Do
Prioritized Perceived Potential Value
Delivered Perceived Potential Value
Realized Value / Benefits

Illustration: This is the core of what it is all about. "Value to stakeholders". Yet agile practices today usually fail to identify or clarify most stakeholders, and most of their values! Current agile practices are thus 'too simple' to be practical and successful.

If you deeply believe this, then the simplest thing you can do is to identify and deal with the top few dozen critical stakeholders. To deal with 'the user' and/or 'the customer' only is 'too simple. This 'top few critical stakeholders' can be brainstormed in 30 minutes, and refined during the project, as experience dictates. It is not a heavy 'overhead', to be conscious of your key stakeholders. It is a necessity for project success.

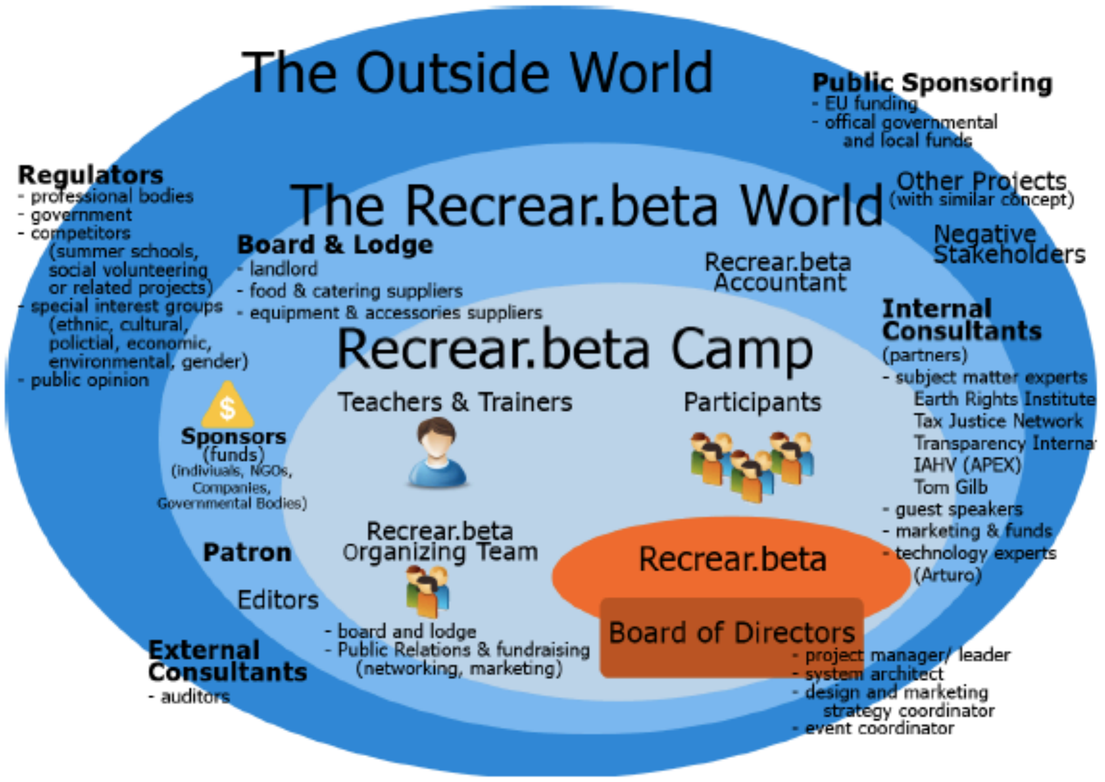


Illustration: Source re-crear.org, a voluntary client of the author.

If you deeply believe that we need to list our stakeholders, then you will also take the trouble to identify the primary and critical **values** of each stakeholder. As a rough brainstorming this is an hour's work, for a small group, to get an initial reasonable values draft.

For Example:

- ***Users****: values are*
  - *Easy To Use*
  - *Easy To learn*
  - *Easy To Correct*
  - *Difficult to Mess Up*
  - *Productive*

Note: this is just a start! We need to define the values well enough to know if designs will work and if projects are measurably delivering value incrementally! Otherwise the above 'nice sounding value words' would be 'too simple, for success. See below and Part 1 of this paper [23, 24]

You can refine the list of values as experience dictates. You can also largely reuse lists of stakeholders, and their known values in other projects in your domain.

Doing this is NOT a heavy project overhead. The argument is that *both* exercises (stakeholders, their values) save time, prior to 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 stakeholder values initially – but they will 'get you' later. I value 'real simplification', not 'illusory simplification'.

<u>My Communication Value</u>

So now we come to my second value, communication. I am sure we all believe in good communication, and are against bad communication. But in the IT/Software world I wander in, world-wide, they do what I would call 'bad communication' normally. And they do not even discuss this and make conscious decisions. I have a simple way of measuring 'bad communication' that never fails to 'surprise' managers and techies alike; who are under the illusion that 'communication is not perfect, but it is pretty good, maybe good enough. I bring them through a simple (5 to 30 minutes) exercise, on 'good requirements of their choice'. I use a method described elsewhere in more detail. I call the method "Agile Specification Quality Control" (SQC) [27, and 10, Chapter on SQC]. This is a really simple way to measure communication. Just ask 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). Ask them to agree to 2 rules:
1. The text (words and phrases) should be <u>unambiguous</u> to the intended readership
2. The text should be <u>clear enough to test</u> successful delivery of it.
Usually we add a third rule
3. The 'objectives' should <u>not specify proposed designs</u> or architecture for getting to our objectives.

They will have to agree that these rules are logically necessary. Then ask them to spend 5 to 30 minutes identifying any words, terms phrases which fail these rules. And ask them to count the number of such failures ('specification 'defects' – fail the rules).

Collect the number of defects found from each participant. That is in itself enough. Most everyone has found 'too many'. The critical communication is obviously 'bad'. People find something like 5 to 40 defects per 100-300 words.

But, it can get even more fun if you realize that the best defect finder in the group probably did not find more that 1/6 of what is actually provably there, and a small team found only 1/3 of it! [27]

The sad thing is that this poor communication about IT projects is pervasive, 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 customer of ours, good guys, reduced major defects per page from 82 to 10 in 6 months. Most people are at about 100-200 defects/page and do not even know it. But, they admit they are guilty after this test!

If you measure, and systematically improve software specification (including code and tests), then you value communication like I do. But, you don't really do this yet, do you? So we share the words *'I value good communication'*, but we have not *really* communicated what we mean by it – yet. I tried. Your turn now! Do you really believe that 100 major defects per 300 words (a virtual page), that your organization is probably guilty of, is good enough communication? Have I communicated my 'communication' value clearly enough to you yet? What part of **'more than 1 major defect per page is unacceptable'** do you not understand?

Communication, Value **2. Communicate stakeholder values quantitatively.**
So, I value the tool of 'numbers' to communicate values clearly. Most IT Projects seem to believe in nice sounding words.

Here is a real example for a $100,000,000, 10 year project (from 14, which has many such cases). The CEO objected, when the CIO suggested clarification!

1. Central to The Corporations business strategy is to be the world's premier integrated_ <domain> service provider.
2. Will provide a much more efficient user experience

3. Dramatically scale back the time frequently needed after the last data is acquired to time align, depth correct, splice, merge, recompute and/or do whatever else is needed to generate the desired products
4. Make the system much easier to understand and use than has been the case for previous system.
5. A primary goal is to provide a much more productive system development environment than was previously the case.
6. Will provide a richer set of functionality for supporting next-generation logging tools and applications.
7. Robustness is an essential system requirement (see rewrite in example below)
8. Major improvements in data quality over current practices

Illustration: Real (slightly modified for confidentiality) objectives for a large software project. How many unclear and ambiguous terms can you count here? How many testable, or measurably clear ideas do you find?

We were able to immediately quantify and clarify everything in the objectives. Nobody had every tried in the previous 8 project years.

Here is a partial example:

<u>Robustness.Restore Speed:</u>
Type: Software Quality Requirement.
Part of: Robustness
Ambition: Should an error occur (or the user otherwise desire to do so), Horizon shall be able to restore the system to a previously saved state in less than 10 minutes. <- 6.1.2 HFA.

Scale: Duration from Initiation of Restore to Complete and verified state of a defined [Previous: Default = Immediately Previous] saved state.

<u>Initiation</u>: defined as {Operator Initiation, System Initiation, ?}. Default = Any.

Goal [Initial and all subsequent released and Evo steps] 1 minute?

Fail [Initial and all subsequent released and Evo steps] 10 minutes. <- 6.1.2 HFA

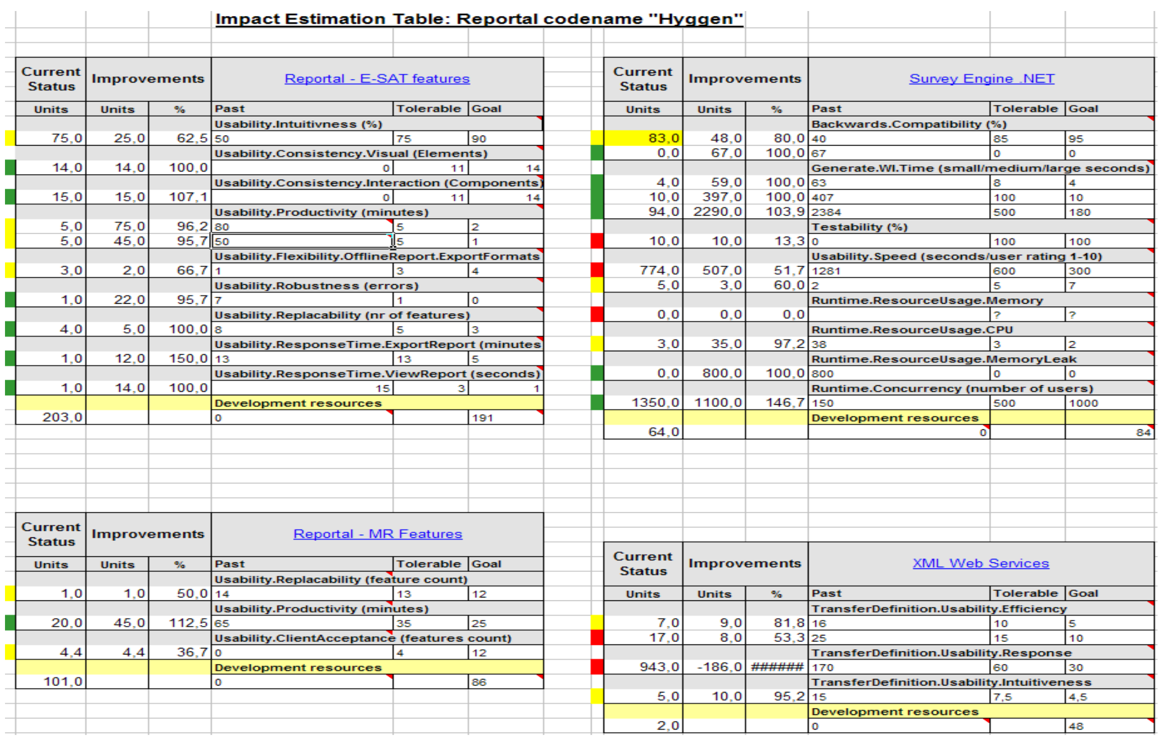
Catastrophe: 100 minutes.

Illustration: using Planguage [10] to define one sub-attribute of Robustness. This took about 10 minutes.

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

I am a strong believer in 'Human <-> Software Communication' (as opposed to us guessing in a meeting of just humans). My experience of humans is that they don't know much truth of the technical systems they discuss. For example when making estimates for project costs. [28]. It is far simpler and more accurate to observe real systems, and their cost and quality attributes – that to 'make estimates' without yet observing reality.

One great benefit with **evolutionary** [29] projects (which *include* iteration of delivery and feedback on costs and capability, and incrementing of system capability) is that we can let the project speak back to us about what's happening, and relate that to our human-agreed performance and quality levels, and incremental costs. We can learn from unexpected deviation from plans. But, in order to let the system talk back to us, with its opinions about itself, we have to do better than measuring 'stories burned down'. We have to listen to (measure) the real top level stakeholder values that are being produced, or not. And most of you have not yet specified what those values are yet. So there is nothing to learn or listen to at all.



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

Case Example [16] Confirmit. Using the Evo Agile method. 4 small development teams, 13 developers in total, work on a total of 25 top level critical software product requirements, for a 12 week period of weekly value delivery cycles. This is a snapshot of cycle 9 of 12. If you look at the 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. They are communicating with the system in a far more advanced way that looking at story burn down rates. They are tracking the primary concerns of their stakeholders. Values.*

I value dialogue between developers and the emerging system at this level. The system tells the truth. People have illusions that need rapid correction. The most important dialogue is about values and costs. The values need quantification in order to have an effective dialogue. Use cases and stories are not the right measure of stakeholder value.

Feedback Value **4. Install real quantified improvements for real stakeholders. weekly**

- I value getting real results. Tangible benefits that make stakeholders want to dance in the streets with joy! I value seeing these benefits delivered early, frequently, and in large measure.
- I have seen stories and use cases delivered [15] by experienced Scrum teams, with just one small problem. The stakeholder businesses found that their sales dropped dramatically as soon at the fine new Scrum system was delivered. Why? A little detail. It was taking about 300 seconds for a customer to find the right service supplier. Nobody had tried to manage that little detail. After all computers are so fast! The problem lay in total failure to specify usability requirements quantitatively. Like 'maximum time to find the right supplier will be 30 seconds, and average 10 seconds'. The system needed better design and value requirements outside the Scrum team. And then succeeded. Scrum was ok, but the front end was not. It was a management problem, not a programming problem. It required several levels of management value analysis above the developer level to solve.
- Scrum burn rates were fine, but they were burning the wrong values.
-
- Stakeholders do not EVER value 'function' (stories, use cases) <u>alone</u>. They need suitable quality and performance attributes delivered too. 'Traditional agile practice' hardly acts as though this is worth considering. I think they are dead wrong.
- It is also very healthy to prove that you can deliver real value incrementally, not just assume that stories are the entire real value – they are NOT. This real value delivery means that we must apply total systems thinking: people, computers, databases, and much more than code.

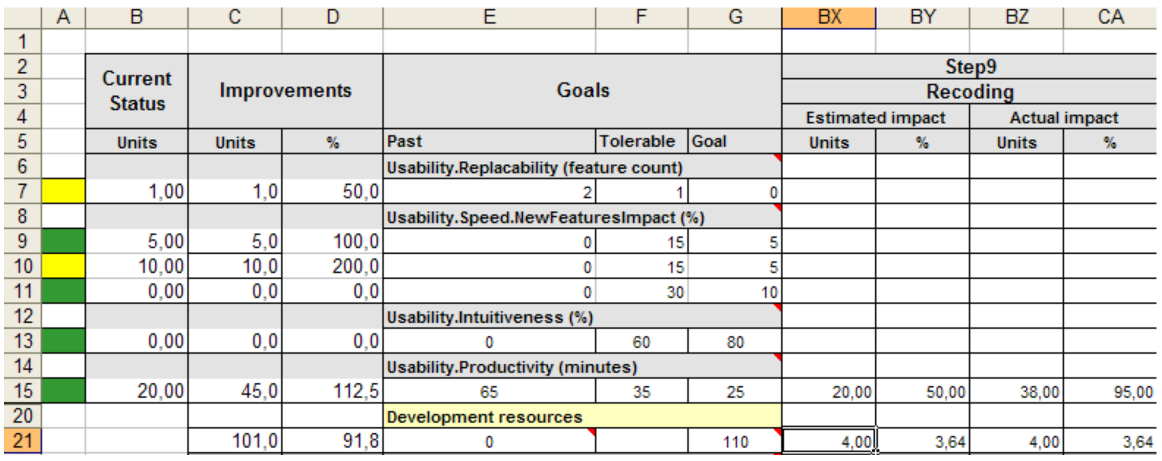
-

Feedback Value **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! Our IT-system failure rates are notoriously high too!
- Some developers may carry-over to agile a Waterfall method concept of measuring critical attributes (at least performance) at the **'end'** of a series of delivery cycles, before a major handover, or contractual handover.
- Assuming the critical aspects are few (5 to 25) I think that we need to measure (test) any one of them that has changed, not just the ones we are targeting for improvement on a *weekly cycle*, and also measure any one which may have been *negatively* impacted by our development changes.
- Measurement need not be expensive for short term cycles. We don't need the final truth, and we are not doing a PhD. We can use appropriate simplification methods such as sampling, to give early indications of progress, order of magnitude of the progress, and of possible negative side effects. This is known as good engineering practice (they do *call* themselves software *engineers*?).
- One of my clients (Confirmit) [16] simply decided they would spend no more than 30 minutes per week to get a rough measure of the critical value-to-stakeholder attributes (a few, not all 25 each week). That worked for them.
-

Feedback Value **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 to specify requirements, and it requires measuring numerically, with enough accuracy to sense interesting deviations. Confirmit [16] does this in the case above.
- *For example:*



	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

- *Illustration: In this case when the impact of the 'Marketing Info Recoding' was twice as powerful as expected (actual 95% of requirement level met, versus 50% estimated) the team was able to stop working on the Usability.Productivity attribute and focus their attention in the 3 iterations remaining before international version release, on the other weaker requirements, like Intuitiveness. The weekly measurement was done by Microsoft Usability Labs in this case. This 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 the essential stakeholder values any other way – it has to be numeric. But numeric feedback about the key stakeholder values is hardly mentioned, and hardly practiced. We have 'apparent numeracy' by talking about velocity and burndown rates – but I mean numeracy about the critical stakeholder values. These are always either **qualities** (-ilities like reliability, usability, security) or **work capacity** (throughput, response time, storage capacity). All of these are quantifiable and measurable in practice [30] though few developers are trained to understand that, about the 'quality' requirements (ask how they measure 'usability') [30].

Courage

I value courage. I practice courage in my work (I am also doing right now by writing this paper). Courage to do what is right for the stakeholders. Courage to do what is right for your organization, and your project team – even if there are strong pressures (like the deadline) to avoid doing the right thing.
I see no signs of this courage in the current agile environment. Everybody is happy to go along with a weak interpretation of some agile method. I do not see the strong leaders, the warriors for ethical development, the champions of the oppressed (stakeholders)! People don't seem to care. If things go too badly – get another job. If millions are wasted – who cares, 'it's not my money' [20, 31].

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?

Courage Value **7. Change plans to reflect weekly quantified learning.**
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 to attack and change the architecture, supported by powerful directors, who have been led to believe in it by too-clever marketing. This would presuppose that these same directors, or other equally powerful forces in the organization, had agreed that they primarily wanted some particular quantified value delivered (say for 'Robustness', see example above), and it was clear to you from the feedback that a major architectural idea was not at all delivering on the promise.
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 [20], and the situation may be as corrupt or political as described in [30] 'Plundering the UK Economy'.
But, in my experience, the problem, closer to the project, 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 [14]. Intelligent, but ignorant. Champions with 'financial budgets', and children with quality.

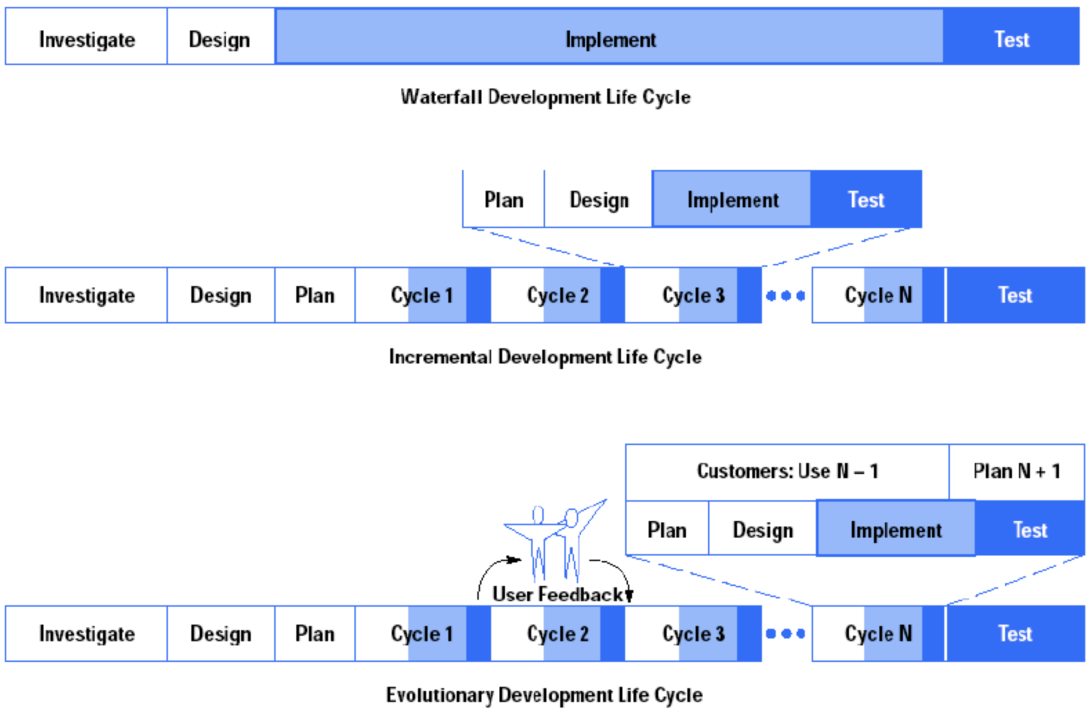


Figure 1. Life Cycle Concepts (Cotton 1996)

Ill. Concepts of weekly delivery cycles with stakeholder feedback. From HP, a client applying our Evo method on a large scale [24, 29, 32, 33, 34 ill. source]

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. I have done it on large US Defense IT systems [35, Persinscom]. I have done it on 25 successive aircraft projects. We just did it at a large multinational bank a few months ago. I've been doing it since I was 20! 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 planned to be retired, which need improvement. And we make use of systematic decomposition principles [29, 21, and in decomposition principles in detail with cases 35]. The big trick is to ignore the 'construction mode' that developers have, and focus on 'Stakeholder Value Delivery' mode.

•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 (=one week, for a 1 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 stakeholder. Do the juicy bits first!

Template: "Decomposition Policy" [35]: This is my simple advice to top managers, when they ask me how they can support deploying the Evo method, and getting rapid results. Demand this 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.

Courage Value **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!

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) value delivery increments.

Most people have no training and no 'theory' [35] for doing this. Most people have never seen it happen in practice. Agile developers have now widely practiced delivery of functionality (stories) in small increments. That is a start, culturally. But *functions* are not the same thing as *value delivery to stakeholders*.

Assuming you can deliver reasonable value for effort-spend, 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

I sometimes simplify this method by calling it the '1.1.1.1.1.1' method, or 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

It is amazing the practical power of this simple idea of Unity

If you really try, and management persists in encouragement and support, it almost always works. It sure beats waiting for weeks, months, and years, and 'nothing happens' – of 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 do-able.

Delivering value to 'customers' is in fact what the Agile people have declared they wanted to do – but in my view they never really understood value and stakeholders. They (Manifesto Signers) don't even have the courage to recognize that, in public, and do something active about it. I have a feeling they do not really care. It should be said

several of them have acknowledged the potential usefulness of Planguage (value, stakeholder, quantified quality) ideas to me and in public. But I have not seen them come out and write or teach their methods differently.
One interesting exception – but he did not sign the Manifesto, and is really a very different culture from the other guys (hint Top Gun, Medical Researcher) – Jeff Sutherland. Jeff is the guy behind the most successful agile method, Scrum. He has come out in public and acknowledged the need for a much better value-oriented front end to Scrum. I hope he can influence the change to current Scrum practices in the direction of a far more stakeholder-value delivery environment. As Jeff makes clear, Scrum essence is the feedback and learning loop. All kinds of practical things have to be added locally to make it a practical tool. And he and I believe stakeholder value focus is a necessary improvement. His colleague (Scrum Alliance), Gabrielle Benefield has worked with me to develop some teaching materials for Product Owners, in order to teach the basics of stakeholders and value quantification [35]. Gabrielle has had the *real courage* to put in significant time, money, and client-directed effort to make changes in Scrum front end practices. She is really 'agile' !

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

- Confirmit [16] actually uses impact estimation [4, 10, 19] to guess what value will be delivered next week (see illustration above). I think they did not directly tell the affected stakeholders what they predicted. Most of them got to see the results each quarter. And the results were incredible.

Description of requirement/work task	Past	Status
Usability.Productivity: Time for the system to generate a survey	7200 sec	15 sec
Usability.Productivity: Time to set up a typical specified Market Research-report (MR)	65 min	20 min
Usability.Productivity: Time to grant a set of End-users access to a Report set and distribute report login info.	80 min	5 min
Usability.Intuitiveness: The time in minutes it takes a medium experienced programmer to define a complete and correct data transfer definition with Confirmit Web Services without any user documentation or any other aid	15 min	5 min
Performance.Runtime.Concurrency: Maximum number of simultaneous respondents executing a survey with a click rate of 20 sec and an response time<500 ms, given a defined [Survey-Complexity] and a defined [Server Configuration, Typical]	250 users	6000

- *Case, Confirmit. The 5 best of 25 initial results in the first quarter release of using Evo.*
- But once Confirmit realized they could continually get such great improvements, they did brag about it *numerically* on their website marketing! So they were not afraid to indicate they were good and using Evo, as they mentioned, they expected to continue dramatic improvement.
- Since even this process of getting amazing improvements is still quite unpredictable, you don't really know if you will get 20% or 200% improvement; maybe it would be foolhardy, rather than courageous, to tell people what they were going to get next *week*. But, based on the weekly proven changes, you could announce, safely, what improvements in value you were going to deliver in the next major release!

Courage Value **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' [4]).
Do not be limited by your personal craft background, in serving your paymasters – getting real stakeholder results.
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 mean *program logic* architecture. Agile developers do not include in their design practices or responsibility, *data structures, databases, network architecture, hardware architecture, motivation, training, contractual deals, maintenance, system porting, operations* and all other elements of the real 'system'. They seem narrowly focused on their code. In fact, as I have said, they focus on the code 'functions', and not even the code qualities!. Listen to them write, speak, tweet – it is all about code, stories, use cases.

If the hot-shot coders of the world decline responsibility, then someone else must have the courage to take the systems-wide responsibility for requirements, design, architecture, and full practical implementation. The coder will be 'reduced' to the 'bricklayer' they seem to be.

In order to get competitive results, someone else – a 'real' architect will have to take over the overall responsibility. Sad, because many of the coders I know are quite intelligent – they *could* learn to do a *real* system architect's job, a *real* system engineers job. But they don't seem to want to.

Fine. First class coding is valuable, like first class crafts of any kind. But I wish they would not call themselves engineers and architects; not to mention Scrum 'Masters' (after two days attendance). Who do they think they are kidding? Not exactly 'humble'!

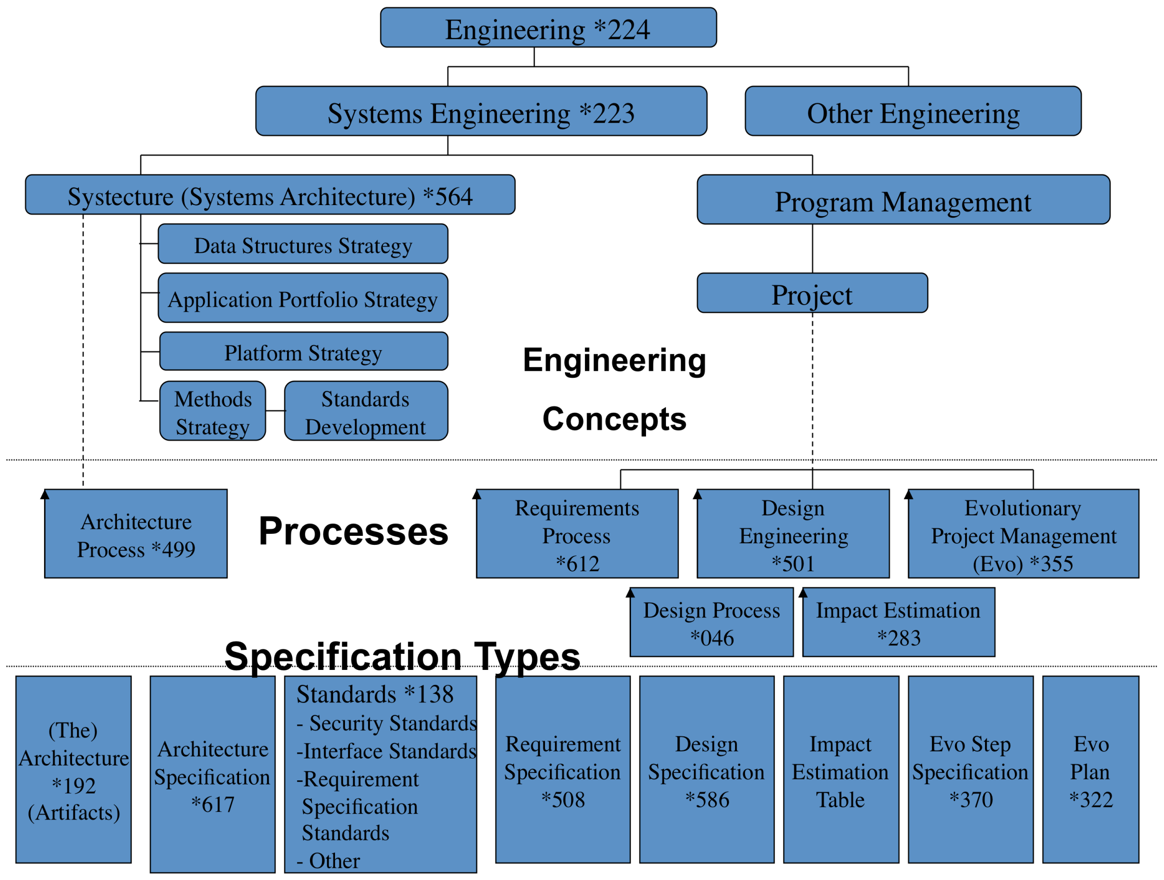


Diagram: my 'Competitive Engineering' view of systems engineering [10]. This set of disciplines figures out what the crafts need to do in order to deliver results. Serious and large software systems need to be developed using these disciplines. OK, this is a bit much for most software IT projects, but we do need something more than badly trained, motivated, and managed 'Product Owners' to run the show?

Summary

The Agile Manifesto [2] was never a well formulated document, from the point of view of making sure we did the right things right. I have at least, in my principles and values here, tried, in my not *too* humble opinion, to show a specific alternative – how it might have been. The manifesto has of course been successful in getting a wave of change. And moving to rapid iteration is a 'good thing'. But rapidly iterating in wrong directions is *not* progress. The key idea of intelligent iteration towards well-defined stakeholder values, was clearly and extensively spelled out in Principles of Software Engineering Management [4 in 1988], which most of the leading agile gurus point to as a source of some of their agile ideas [5 to 12]. But as some of them reflect today, they missed at least one simple but essential point – 'stakeholder value' needs to be the guiding light for the iterative process – not functions and stories.

Hey, we are still a young discipline. We only wasted about a decade getting this wrong. Software has been seeing failed fads come and go for 50 years. We have so many experienced intelligent people now – maybe we can get it right in the next revolution?

If the IT-project failure rate (total plus partial) goes down from about 90% (Standish) to less than 2%, you will know we got it right, for a change. Do you think we can do it right by my 100th Birthday?

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.
*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-gilbsevolutionary-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/tikidownload_ 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. Gilb, "**Decomposition of Projects: How to Design Small Incremental Steps**" INCOSE 2008,*

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

Here is a summary of Gilb's Agile principles from the Part 1 of this paper.

1. **Control projects by quantified critical-few results. 1 Page total ! (not stories, functions, features, use cases, objects, ..)**
2. **Make sure those results are business results, not technical**
- **Align your project with your financial sponsor's interests!**
3. **Give developers freedom, to find out how to deliver those results**
4. **Estimate the impacts of your designs, on your quantified goals**
5. **Select designs with the best impacts for their costs, do them first.**
6. **Decompose the workflow, into weekly (or 2% of budget) time boxes**
7. **Change designs, based on quantified experience of implementation**
8. **Change requirements, based on quantified experience, new inputs**
9. **Involve the stakeholders, every week, in setting quantified goals**
10. **Involve the stakeholders, every week, in actually using increments**

[24] Gilb, T., Value-Driven Development
Principles and **Values** – Agility
is the Tool, Not the Master. Part 1. The first half of this paper.
http://homepage.mac.com/tomgilb/filechute/Agile%20Principles%20and%20Values%20for%20Agile%20Record%202010%20Gilb.doc
Above link to paper agilerecord.com Summer 2010

[25] http://www.gilb.com/tiki-download_file.php?fileId=391
Above Link to my June 16 2010 NDC Oslo Slides on these Principles and Value.

[26] Calaprice, A. (Editor), The New Quotable Einstein, 2005, and "The Expanded Quotable Einstein" (see 'Attributed to Einstein', page 311 for 'Simpler' quote disclaimer). Gilb has had extensive correspondence with AC on this quote.

- http://press.princeton.edu/titles/7921.html
- http://www.amazon.com/Expanded-Quotable-Einstein-Albert/dp/0691070210

[27] References on **Agile Specification Quality Control.**

- 27a Test Experience Paper http://www.gilb.com/tiki-download_file.php?fileId=264
- 27b. Javazone slides http://www.gilb.com/tiki-download_file.php?fileId=333

[28] Gilb, **Estimation or Control**, draft paper, http://www.gilb.com/tiki-download_file.php?fileId=433

[29] Evolutionary Chapter 10, of CE [10], http://www.gilb.com/tiki-download_file.php?fileId=77

[30] CE book , free chapter on Scales of measure. This is about practical quantification of qualities like Usability. http://www.gilb.com/tiki-download_file.php?fileId=26

[31] *] **David Craig and Richard Brooks***

Plundering the Public Sector: How New Labour are letting consultants run off with £70 billion of our money

http://www.amazon.co.uk/Plundering-Public-Sector-David-Craig/dp/1845293746

[32] Upadhyayula2001. A quantitative study of Gilb'd Evo method in use at HP. Sharma Upadhyayula MIT Thesis. "RAPID AND FLEXIBLE PRODUCT DEVELOPMENT: AN ANALYSIS OF SOFTWARE PROJECTS AT HEWLETT PACKARD AND AGILENT" supadhy@mit.edu. http://www.gilb.com/tiki-download_file.php?fileId=65

[33] MAY96: Elaine L. May and Barbara A. Zimmer, **"The Evolutionary Development Model for Software"**, Hewlett-Packard Journal, August 1996, Vol. 47, No. 4, pages 39-45.

http://www.gilb.com/tiki-download_file.php?fileId=67. Elaine is at Tektronix now.

[34] Todd Cotton, **"Evolutionary Fusion: A Customer-Oriented Incremental Life Cycle for Fusion."** todd_cotton@stanfordalumni.org

http://www.hpl.hp.com/hpjournal/96aug/aug96a3.pdf

[35] Decomposition Slides Aug 2010

http://www.gilb.com/tiki-download_file.php?fileId=350

[36] Value Planning for Scrum: Quantified Quality for Scrum

http://www.gilb.com/tiki-download_file.php?fileId=353

I should stress that this is only our initial draft, and current practice is tailoring these ideas to the real environment.