Agile Record

The Magazine for Agile Developers and Agile Testers

Distributed Team Management

August 2013

issue 15

www.agilerecord.com free digital version made in Germany ISSN 2191-1320

Column

Agile Contracting for Results

The Next Level of Agile Project Management

by Tom & Kai Gilb

Agile development presents an opportunity for better agile project management that not many seem to appreciate or practice yet. The fact that we agilistas all seem to agree to 1) deliver iteratively, 2) increment the product, and 3) learn and change evolutionarily means that there is an opportunity we did not have in the days of waterfall.

This opportunity is available when the work is outsourced. Consequently, it is not considered in general agile models which focus on the development process, with the apparent assumption that work is done in-house. Please consider that the ideas here could also be applied to reward and motivate in-house teams, without a legal contract framework.

We can write the contract so that work is paid for as useful, proven results are delivered.

Why should we pay a supplier just because time has been spent on a project? Why not pay only for real delivery of useful results?

Of course, unethical suppliers, by definition, would be happy to take your money, even if they fail to deliver useful results. And that seems to be common practice. Project failure is common [1]. But we have only ourselves to blame if we allow this misuse of resources and confidence to happen. We believe that if you are contracting out your development work, then you are ethically responsible for getting good results.

Massive payments for failed projects and for non-delivery of necessary results should NEVER happen! Nevertheless, it is too common. Our contracting model is broken, yet any one of us can fix it immediately.

We think the best way to ensure good results is to motivate the supplier/developer by paying them well for results, and paying little or nothing for lack of results. We think that a contract should be discontinued, at any time, when planned results are not forthcoming. Continuation of a contract should be based on a reliable series of planned results.

Ideally the results are something more than "code without too many bugs" or "user stories." The results should primarily be high-priority business and organizational improvements: based on the top few critical reasons for the project [2]; based on measurable improvements in cost, speed, and qualities such as usability, security, maintainability, reliability, and many others, as prioritized.

But most developers and product owners are not trained or motivated to do this. They have neither the vision ("no cure, no pay" [3]) nor the technical ability to carry it out. In fact, they lack results leadership by their managers.

The Necessary Ingredients

- Ability to define results so they are testable and usually measurable [4]

  We are not talking about the mere technical software product of the development effort. We are talking about the results – improvements for people (like ease of learning a new application) that result from use of the product.

  This raises the question of whether to contract for the technical characteristics of the technical product itself (usability, for example), or its knock-on effects in the organization (time savings, fewer errors, etc.) when deployed.

  In practice, the technical supplier (e.g. software developer) can really only be responsible for the technical product, not how well it works with your own customers and employees. You as a technical product customer need to translate those technical characteristics (like usability) into organizational results (like training savings).

  But we should ideally be able to contract out even that higher organizational level of results for payment, too. This is a systems engineering level, not a software engineering level.

- The ability to think of the real total system – the people, their motivations, the data, the hardware and software

  Current agile thinking is far too code-centric. Someone needs to worry about designing, planning, and managing the total package needed to deliver useful and long-term results. We need a level of systems engineering to manage the programming. We need a result owner level of management.

A New Open Agile Contractual Framework Initiative

There is an effort by some of our professional friends, with our active participation, to help with the legal framework for this [5].

They have published a free contract template to provide a legal framework for results contracting.

It has two major components:

1. a contract framework, taking care of the long-term overall legal statements; and
2. a contract sub-component – the result specification cycle – that defines the customer-level results of technical implementation, for one or more consecutive delivery cycles.

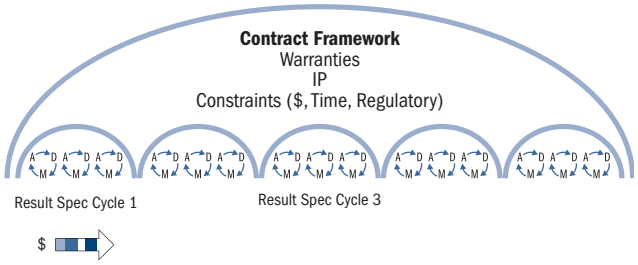

Figure 1. The contract framework applies to all result specification cycles. The result specification cycles are not initially specified in the contract. They are specified as experience and need dictates. Cycles can be terminated when needs are fulfilled or when the supplier does not perform successfully. Payment, in part or whole, is based on delivery of specified customer results, not on delivery of technology itself.

The flexible results contracting idea is simple – the relationship between the customer and the supplier can exist only for as long as the defined results keep on coming through the pipeline. If results are not happening, the customer is free to stop further activity. You are no longer bound by contract to a supplier who produces bad or costly work. And they know that, so they will be more motivated, at all times, to keep the customer happy. This of course can also be viewed as a benefit for the supplier organization too! They are motivated to be better suppliers.

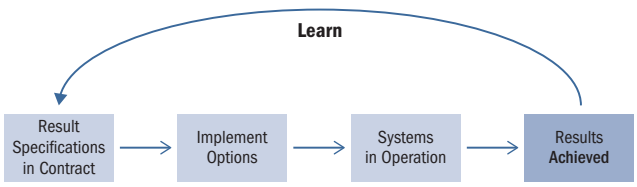

Figure 2: The result contract structure is one of learning from actual results achieved, in operation, in order to decide more intelligently on the next set of contractual result specifications in detail. The requirements are not in the contract framework. Requirements are developed gradually, based on feedback from the receiving organization and on feedback from the customer's changing environment. Options are the designs, strategies, or architectures necessary to deliver the requirements for that result cycle.

Traditional Contract Model	Result Contract Model (Agile)
Requirements are contractual and specified up-front in the main contract.	Requirements are specified at the start of each result cycle.
Changes are managed by means of the change control mechanism.	Requirements are more resistant to change than traditional output requirements. Target outcomes are only specified at the start of each result cycle, are operational for shorter periods of time, and therefore are exposed to less change.
Analysis, design, development, and testing occur sequentially. Big Bang or Waterfall.	Each cycle must deliver value, so design and development occur concurrently. A systems view must be taken, providing real results in real life.
An all or nothing solution.	The solution evolves as a serious of result deliveries.
Constituent modules of software are worked on independently until integration takes place.	There is continuously working and stable software and hardware system.
Testing is used as a contractual tool at the end of the development process.	Testing occurs throughout the development process, providing feedback for improvements.
Success is measured by reference to conformance with the change-controlled contract.	Successs is measuered, cycle by cycle, by requirements delivered, driving value to the customer.

Figure 3: Comparing waterfall contracting to agile result contracting.

Agile Architecture: Proving and Changing Design Gradually to Meet Real Needs

Flexible results contracting will require us to manage software projects at a higher level of concern. Instead of paying suppliers for the implementation of our design (i. e. the technical solutions we hope will meet our business or organizational requirements), we will contract and pay for a higher level of requirement, such as degrees of usability, security, maintainability, and performance. We can free up the supplier (perhaps after trial and error to see what really delivers our needs) to ultimately select, on our behalf, the most cost-effective architecture to meet our technical requirements.

Be careful what you ask for! They may deliver the bad architecture you demand, which does not provide what you hoped to get from it!

We should not put ourselves in the position of fixing technical designs prematurely in a contract. We often falsely call these designs requirements, when they are not our organizational technical requirements (like better security) at all. There is usually no real proof at that early (initial contract) stage that the specified designs or architecture will deliver what we really want.

This must be proven incrementally. We must build our systems on a succession of proven designs that really deliver the technical characteristics of quality, performance, and cost we really want. To do that, we need to test our design assumptions in real systems in practice, and we need to delegate technical design decisions to capable architects and designers, hopefully a competent part of the supplier service. And we need to do so on the shop floor of reality, not in the ivory tower of the

enterprise architect. It is a common bad practice to make all key design decisions too early, unproven, and too much at once [6].

Summary

We need to acknowledge the complex systems and the complex environment we all work in by taking another step in the agile direction: agile contracting for results.

Customers need to focus on the business and organizational results they really need in their technical systems and motivate suppliers to deliver these levels by paying for good results only. All solutions, strategies, designs, architecture, and means to deliver what we as customers really value must be proven gradually. We must learn what works for technology and suppliers rapidly, continuously, incrementally, and assess the consequences quickly. That's agility!

We are not yet, as a culture, truly agile in contracting or in systems architecture. But we can be, if responsible leaders take the lead in doing so.

Is that the *you* we are talking to?

References

[1] Highly recommended in-depth analysis of good and bad agile practices, even if you are NOT in the public sector: Wernham, Brian. *Agile Project Management for Government.* Maitland and Strong.

[2] Gilb, Tom. "The Top 10 Critical Requirements are the Most Agile Way to Run Agile Projects". *Agile Record*, August 2012, 11: pp. 17–21. *http://www.gilb.com/dl554*

[3] Gilb, Tom. "No Cure No Pay." *http://www.gilb.com/tiki-download_file.php?fileId=38*

[4] Gilb, Tom. "Chapter 5: Scales of Measure." *Competitive Engineering*. *http://www.gilb.com/tiki-download_file.php?fileId=26*

[5] This initiative is a draft idea and would welcome cooperation and feedback from people who would like to try it out in practice! *www.flexiblecontracts.com*

[6] Gilb, Tom. "Real Architecture Engineering." Lecture slides from ACCU Bristol, April 2013. *http://www.gilb.com/dl574*

■

> about the authors

Tom Gilb and Kai Gilb

Tom Gilb and Kai Gilb have, together with many professional friends and clients, personally developed the agile methods they teach. The methods have been developed over five decades of practice all over the world in both small companies and projects, as well as in the largest companies and projects. Their website *www.gilb.com/downloads* offers free papers, slides, and cases about agile and other subjects.
There are many organisations, and individuals, who use some or all of their methods. IBM and HP were two early corporate-wide adopters (1980, 1988). Recently (2012) over 15,000 engineers at Intel have voluntarily adopted the Planguage requirements specification methods; in addition to practicing to a lesser extent Evo, Spec QC and other Gilb methods. Many other multinationals are in various phases of adopting and practicing the Gilb methods. Many smaller companies also use the methods.

Tom Gilb

Tom is the author of nine published books, and hundreds of papers on agile and related subjects. His latest book 'Competitive Engineering' (CE) is a detailed handbook on the standards for the 'Evo' (Evolutionary) Agile Method, and also for Agile Spec QC. The CE book also, uniquely in the agile community, defines an Agile Planning Language, called 'Planguage' for Quality Value Delivery Management. His 1988 book, Principles of Software Engineering Management (now in 20th Printing) is the publicly acknowledged source of inspiration from leaders in the agile community (Beck, Highsmith, and many more), regarding iterative and incremental development methods. Research (Larman, Southampton University) has determined that Tom was the earliest published source campaigning for agile methods (Evo) for IT and Software. His first 20-sprint agile (Evo) incremental value delivery project was done in 1960, in Oslo. Tom has guest lectured at universities all over UK, Europe, China, India, USA, Korea – and has been a keynote speaker at dozens of technical conferences internationally.

Kai Gilb

Kai Gilb has partnered with Tom in developing these ideas, holding courses and practicing them with clients since 1992. He coaches managers and product owners, writes papers, develops the courses, and is writing his own book, 'Evo – Evolutionary Project Management & Product Development.' Tom & Kai work well as a team; they approach the art of teaching their common methods somewhat differently. Consequently the students benefit from two different styles.