Agile Record

The Magazine for Agile Developers and Agile Testers

Refactoring

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

November 2013

issue 16

Column

Agile Architecture Engineering:
Dynamic Incremental Design Selection and Validation

by Tom Gilb & Kai Gilb

Agile project management offers us a whole new method for approaching architecture and design engineering, of both IT systems and software.

Agile is *iterative* (cyclical, repetitive), and *incremental* (cumulating stakeholder value delivery), and *evolutionary* (learning from experience, and changing plans). This means we have very useful opportunities to manage systems and software architecture, and design, better.

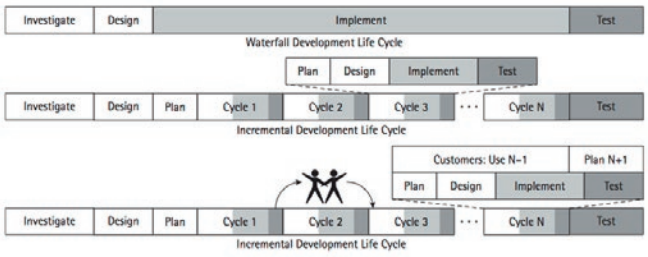

Figure 1

Architecture and design is complex, and we really know very little about the impact on values and costs of most of our initial design suggestions. They need to be considered as mere hypotheses – to be proven or disproven. The outcome is only roughly understood in advance, and our scope for estimation error is intolerably wide (at least in terms of order of magnitude) [8].

Agile offers a useful practical solution. But most Agile cultures, as taught and practiced today (Agile Manifesto, Scrum, XP for example) do not take a position on the measurement and management of architecture and design. But some earlier Agile methods, such as Evo (1970–2013 [3, 6, 10]) and Cleanroom (1970–1980s [1, 2, 6]), have long since exploited the architecture management opportunity inherent in cyclical incremental evolutionary system delivery to successfully manage the architecture and design itself.

Harlan Mills, IBM Federal Systems Division, comments on the ability of the Evolutionary "Cleanroom" method to control projects perfectly: "LAMPS software was a four-year project of over 200 person-years of effort … in 45 incremental deliveries. There were few late or overrun deliveries in that decade, and none at all in the past four years" – Harlan Mills, in 1980 [1].

His colleague in the "Cleanroom" method, one of the first "Agile" methods, Robert Quinnan, comments on the "dynamic design-to-cost" aspects: "The method consists of developing a design, estimating its cost, and ensuring that the design is cost-effective." (p. 473, [2])

He goes on to describe a design iteration process that tries to meet cost targets either through redesign or by sacrificing "planned capability". When a satisfactory design at cost target is achieved for a single increment, the "development of each increment can proceed concurrently with the program design of the others".

"Design is an iterative process in which each design level is a refinement of the previous level."

But they iterate through a series of increments, thus reducing the complexity of the task and increasing the probability of learning from experience, won as each increment develops and as the true cost of the increment becomes a fact.

"When the development and test of an increment are complete, an estimate to complete the remaining increments is computed" (Quinnan, [2]).

The "developers" in the current Agile culture are not going to do anything about this. The just want to "code". The responsible architects, such as IT Architects, are going to have to figure out how to exploit Agile for *their* purposes.

The first stage of this is to recognize that *architecture* (the overall discipline of managing a system development through design) and *design* (which includes specialist disciplines such as Human Interfaces Design, Security Engineering, Performance Engineering, and other such disciplines supervised by the overall architect) need to be conducted as an *engineering* discipline. Not as art or poetry.

"Engineering" means managing a numeric set of objectives and constraints, which are our architecture requirements. *Engineering* then implies managing the corresponding numeric attributes of all design and *architecture* (defined as the things we do to achieve our performance and quality requirements, within our resource constraints).

One interesting side-effect of managing architecture as an *engineering* discipline, is that we not only get control over performance and quality aspects of the system, but we simultaneously get far better control over our budget and deadline [1,2, 6, 8], as Cleanroom experience proved long ago.

Technical Prerequisites

In order to do this, (and several groups have done it for decades, so this is not idle speculation, but observation of known methods!) we need to learn and practice the following architecture engineering disciplines:

1. Quantification of all critical *quality* aspects (security, maintainability, usability etc.) [9].
2. Design of suitably cheap processes for measuring at least leading indicators, then better final indicators, of the incremental delivery of technical qualities (like degrees of security or usability) and then of their intended knock-on effects to a higher level of stakeholder interest (e. g. stakeholder perceptions such as "saving time", "feeling confident").
3. Ability to decompose [10, 11] our larger high-level architecture ideas into smaller implementable components (so we can get earlier delivery of their value).
4. Ability to test design-component hypotheses in a safe, but realistic, way before confidently scaling up, once they are working as required.
5. Contracts for outsourcing that envision, allow for, and assist our ability to do all the above engineering and exploration; with the ability to be agile and exchange what does *not* work for that which *does*! The *real* heart of agility [7].

The *Agile* Architecture Engineering Process [10, 13]

The Architecture Engineering process using "Planguage" as a planning language, or any other way to express qualities quantitatively, goes like this. I use a week to get through these initial plans, before diving into cycles of delivering real value from the implementation of architecture components [5].

1. Quantify the top few critical performance and quality objectives for the system. This means a defined scale of measure and at least one level of performance expected in the future. The agenda is that the project will be done, and successful, when these levels of requirements have been reached. Day 1.
2. List and define in some detail (maybe a page each of ten major architecture ideas) [3, the CE book, Design Chapter template for detail] the major architecture components. These should be the set of ideas you believe will enable you to reach the critical requirement levels in the first step above. Day 2.
3. Rate the expected effectiveness of each architecture component on all critical objectives, as well as on critical resources such as money and time. Use an Impact Estimation Table [14]. One rating is to estimate the % effectiveness expected by the deadline. (100 % means we reach numeric goals on time). Day 3.
4. Using the information in the Impact Estimation Table, find subsets of the architecture that are estimated to give very high value (performance and quality requirements level) in relation to resources used [15]. The most efficient designs. Schedule these for early value-delivery cycles [10]. Day 4. Day 5 is presentation to management.
5. Evaluate results (feedback from delivery cycles, on measures of value and costs). Decide what you need in order to improve or learn. Plan the next steps, with a view to maximizing fast progress towards your requirements levels.

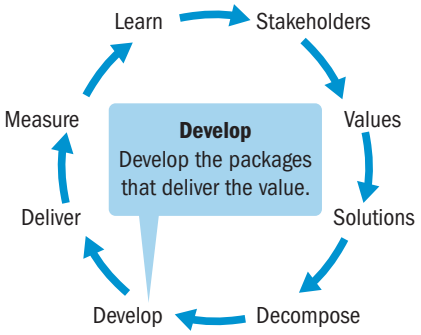

Figure 2. The Evo Cycle [16] – an extension of the Shewhart/Deming "Plan Do Study Act" cycle of SPC methods. Developed by Kai Gilb.

In Summary: we can *engineer* the architecture in *incrementally*. We get earlier results and better results, as a result [4].

References

[1] Mills, H. 1980. "The management of software engineering: part 1: principles of software engineering", *IBM Systems Journal* 19, issue 4 (Dec.): 414–420. Reprinted 1999 in *IBM Systems Journal*, Volume 38, Numbers 2 and 3.
A nice sample, in slides, of how Cleanroom reaches the highest military and space performance and quality levels "always on time and under budget": *http://www.gilb.com/dl602*, "Architecture Prioritization", Oct 2013. See ref. [6] for source of Mills and Quinnan.

[2] Robert E. Quinnan, "Software Engineering Management Practices", *IBM Systems Journal*, Vol. 19, No. 4, 1980, pp. 466–77.
A nice example in slides for Oct 9 2013 1.5 hour talk at London "Software Architect" conference.: *http://www.gilb.com/dl602* "Architecture Prioritization" Oct 2013. Quinnan goes into detail on his dynamic design to cost practice within Cleanroom. See ref. [6] for source of Mills and Quinnan.

[3] Gilb, T. 2005. *Competitive engineering: A handbook for systems engineering, requirements engineering, and software engineering, using planguage*. Oxford: Elsevier Butterworth-Heinemann. **Free digital copy for first 50 Agile Record readers who email me with a request**

within one week after publication of this paper. After that, see 9 and 10 below.

[4] Gilb: "What's Wrong with Software Architecture". Keynote Software Architect Conference, London 10 Oct 2013. *http://www.gilb.com/dl603*
Slides in PDF: *http://vimeo.com/28763240*

[5] "Confirmit" Company Cases (use of Evo). Johansen, T., and Gilb, T. 2005. *From waterfall to evolutionary development (Evo): How we rapidly created faster, more user-friendly, and more productive software products for a competitive multi-national market*. Available at: *http://www.gilb.com/tiki-download_file.php?fileId=32*.

And: *The Green Week: engineering the technical debt reduction in the Evo Agile Environment* by Confirmit.
http://www.gilb.com/dl575
A Gilb's Mythodology column published in May 2013 in Agile Record No. 14, *www.agilerecord.com*. This paper highlights a case of reengineering a legacy system to give reduced technical debt, using Evo, in a small Norwegian international market software package house.

[6] Gilb, T. 1988. *Principles of software engineering management*. Boston: Addison-Wesley. See *http://www.gilb.com/tiki-list_file_gallery.php?galleryId=15*. "Some deeper and broader perspectives on evolutionary delivery and related technology", chapter 15 of the book.

[7] *Agile Contracting for Results The Next Level of Agile Project Management*: Gilb's Mythodology Column Agilerecord No. 15, August 2013, *www.agilerecord.com*.
http://www.gilb.com/dl581

[8] Estimation Paper: *Estimation: A Paradigm Shift Toward Dynamic Design-to-Cost and Radical Management*. SQP VOL. 13, NO. 2/© 2011, ASQ. *http://www.gilb.com/dl460*

[9] CE book. Chapter 5: *Scales of Measure*:
http://www.gilb.com/tiki-download_file.php?fileId=26

[10] CE Book. Chapter 10: *Evolutionary Project Management*:
http://www.gilb.com/tiki-download_file.php?fileId=77
Evo Standard 2012 for DB, Non-Confidential
http://www.gilb.com/tiki-download_file.php?fileId=487

[11] Gilb, T. 2010b. *The 111111 or Unity Method for Decomposition*, presented at the 2010 Smidig (Agile) Conference, Oslo. Available at: *http://www.gilb.com/tiki-download_file.php?fileId=350*

[12] *Agile Contracting for Results The Next Level of Agile Project Management*: Gilb's Mythodology Column Agilerecord August 2013
http://www.gilb.com/dl581

[13] Evo standards Feasibility Study paper. *"Project Startup" for agile architecture*. *http://www.gilb.com/dl568*
Our column in Agile Record No. 13, *www.agilerecord.com*, as published 7 March 2013

[14] *Impact Estimation Tables: Understanding Complex Technology Quantitatively*. *http://www.gilb.com/dl23*

[15] On Decomposition. See ref. 11 above and: *Decomposition of Projects: How to Design Small Incremental Steps*
http://www.gilb.com/dl41

[16] See *gilb.com* for a dynamic version of the Evo cycle and for more explanation. *http://www.gilb.com/Site+Content+Overview*

■

> 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.
Twitter: *@imtomgilb*

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.