# Beyond Scaling: Scale-free Principles for Agile Value Delivery - Agile Engineering.



http://www.gilb.com//dl865

Version March 14 2016, Modified April 11 2016 (XP)

## Summary

There is widespread interest in how to make Agile (including Scrum) methods, work better, on a large scale.

Mike Beedle's paper [1] gives a good *overview (references to much of the agile scaling literature) of many different proposed methods*. I am not going to argue whether these methods are good or bad. No doubt most of the techniques have some value in some circumstances. My concern is *not* this set of 'conventional agile scaling' ideas'. My concern is the large collection of *other* possible ideas for 'scaling up agile' [18]. I will list them here, and reference the detail.

As a leader of practicing my ideas at Intel says: "scale-free things do not scale (because they just don't need to, and scale is irrelevant)" [25, Erik Simmons, 8 Jan. 2016]. So it is not about scaling methods that initially are best suited for small scale. It is about methods that work at any scale.

When the conventional and new ideas are looked at all together, as a larger set of ideas, I suspect that many conventional ideas will not seem as useful, or as necessary, or as cost effective as they seem to be today; in the absence of my supplementary set of ideas. You judge.

The general nature of most of my agile scaling ideas is that they are based on *engineering* concepts: that means systematic *quantification* of project-and-architecture values and costs [22, 20]. These critical ideas seem totally absent from the mainstream agile literature, and public presentation that I can register [1]. But I have been publishing these engineering tools **long** before 'Agile' (1976 Software Metrics, 1988 Principles of Software Engineering Management [6, which is deep on Agile delivery cycles)

This is not surprising since Scrum and similar variants of Agile (Cyclical delivery) are admittedly almost completely lacking any notion of engineering, and of consequent quantification of values and costs [18].

If we want to move from 60% project failure rates [1], towards better than 99% success rates: we need engineering techniques to tackle large projects (in small 'value' increments).

Scaling up a 'craft' discipline has up to now been attempted, in 'conventional scaling' [1], by scaling up the *craft* process. The normal way to scale up most other types of building processes is by using an engineering paradigm (think: aircraft, computers, buildings, telecoms, cities, roads).

But Agile Manifesto and XP were developed by 'Programmers', not Engineers. If some were not actually programmers, the end result is extremely focussed on programming. And they, the 'coders' have consistently failed to even *discuss* an engineering paradigm [F1]. [18, 19 and most references [2 to 17] ].

The Joke is, that they like to call themselves ‘software engineers. But they don’t do software (like ‘data’) just code. And they don’t do engineering, just code hacking. We are allowing ‘carpenters’ to build and plan skyscrapers, and ‘ditch diggers’ to build and plan super-highways. It cannot work.

So, here is the bottom line:

**The main keys**
**to any scale of software development in agile**
**lie in ‘*engineering* methods’,**
**not ‘programming methods’.**

## THE BASIC TOOLS WITH [REFERENCES] TO PRACTICAL DETAIL

Now, rather than write a long-winded paper with detail, I am going to summarize 10 basic principles, methods, tools, and policies: and for each cite [writings] where practical detail, and more-convincing arguments, will be found. The references in *this* paper are very ‘Agile’ oriented. The ‘VP’ References from my Value Planning book [20] are the best updated detailed practical source of ‘how to do it’. So, I prefer to reference VP frequently, even though it is written for ‘managers’ not Agilistas. But VP is all about Enterprise Agile [1].

## *Principles* for Any Scale Software Projects

1. Keep focus on measurable delivery of critical values and their costs. [3, 4, 5, 6, 9, 10, 12, VP (20) Part 1**, VP 10.6** ]
2. Deliver value early, quickly and regularly: in roughly 2% increments. [14, 11, **VP Ch.4,** 2**,** 5 ]
3. Do NOT focus on code delivery; focus on overall system value and costs. [ **VP Ch.4,** 10D, 10F, 13, VP 3.4, VP 2.10, VP 9.8, 4, 12]
4. Focus on quantified *critical stakeholder* values. [**19**, VP 3.4, VP 3.7, VP 3.9, VP 3.10 VP 4.2, 10 ]
5. Synchronize all teams in terms of measurable value delivery. [VP 3.3, VP 3.4, VP Part 1, VP 3.6, VP 3.8, VP 8.4 , 11, 12, 13 ]
6. Solve big problems through ingenious architecture; not through coding faster. [VP 4.5, VP 5.1, VP 5.3, VP 7.2, 15 ]
7. Decompose the large problems by incremental value deliveries: not code deliveries. [**7**, VP Ch. 5, VP 5.1, VP 5.6 , 10, 11, 13, 15]
8. The software component needs to be integrated into the total system of hardware, data, people, culture. [ VP 5.2, 10 ]
9. If your team cannot deliver small increments of real value early, frequently, and predictably; they are incompetent and need to be abandoned for those who can deliver. [7, VP 2.8, 10]
10. Never commit to contacts for *work done* or *code delivered* alone: there must always be a sufficiently large contractual protection, of paying for measurable value delivered. [12, 15 ].

- *"As to methods, there may be a million and then some, but principles are few.*
- *The man who grasps principles can successfully select his own methods".*
- -

Ralph Waldo **Emerson,** -1803-1882, USA

# *Methods* for Any Scale Software Projects

1. Quantification of Values [10, VP 1.1].
2. Quantification of short term and long term costs [VP 3.4, VP 4.5, VP 6.7 ].
3. Design to Cost: Top Level Architecture [ VP 7.9, 10 ].
4. Dynamic Design to Cost: Each Delivery Cycle [12 C, VP 4.5, VP 2.5, VP 2.3, 5, 10, 12 ].
5. Quality Control of Plans, Contracts, Code and all written artifacts [VP Part 2, VP Part 4, VP 7.7 ].
6. Flexible Contracting [12, VP 4.5].
7. Value delivery Cycle Measurable Feedback, Learning and Change [4, VP 7.3, VP 9.8, VP 6.7, VP 8.6, 2, 9, 10, 11, 14 ].
8. Value Decision Tables (Impact Estimation Tables) [**9**, VP 2.3, VP 4.4, VP 5.3, 13 ].
9. Risk Management in all aspects of planning and Management [ VP Ch. 7], 12.
10. Intelligent Prioritization Policies: for short term and long term [ VP Ch. 6, 12, 13, 14].

# *Engineering Tools* for Any Scale Software Projects

1. The Planning language: 'Planguage' [ 22, VP, 8, 9].
2. The 111111 Decomposition Method [7B, 7C, 3 ].
3. Flexible Contracts [12 ].
4. The 'Needs and means Planning' tool [**16**, 9 ].
5. Quantification of Values processes: Scales, Meters, Past, Tolerable, Wish, Goal. [VP 10.7 ].
6. The Agile Spec QC measurement process, Exit Processes, Rules [VP 10.4, VP Part 4 ].
7. Multiple Relationship Management technology [9, VP Ch.3, VP Ch. 6, 13 ].
8. Continuous Architecture adjustment based on delivery cycle feedback (Cleanroom) [ 5, 14, 8].
9. Graphic Visibility of Values, Costs, and Risks [16 ].
10. Design to Cost Practices: initially and continuously [14, 12 C, VP 4.5, VP 2.5, VP 2.3, 5 ].

# *Management Policies* for Any Scale Software Projects

1. We will primarily manage critical stakeholder value improvements [VP Ch. 3, 8, 19 ].
2. We will simultaneously manage the short term and long term resources [VP 2.5, VP 9.9, VP 10.6, ].
3. We will contract for measurable values for money, rather than 'work done' [**12**, VP 8.4].
4. We will manage all basic system qualities in a quantified engineering manner. [VP Part 1, 3, 4, 8, 10, 17 ].
5. We will prioritize delivery of measurable value, early, frequently, predictably [VP 1.2, **VP Ch. 6**, 6, 12 ].
6. We will not lock ourselves into investments or expenditures of any kind that cannot be reversed if they do not produce expected value for money [VP Ch. 7, VP 8.10 ].
7. We will make the risks of all strategies, designs, actions, and relationships visible numerically; and make decisions with regard to worst-case risks [VP Ch. 7, 12, 17 ].
8. We will empower the 'troops' to make real-time project decisions, based on current numeric feedback, about real values and real costs [**VP 8.1, VP 8.2**, 10, 11, 13 ].
9. Every team or set or related teams will be judged by their ability to deliver a measurable, predictable value improvement stream [ VP 8.7].
10. Decisions will not be made on badly-defined-package costs: decisions will be made continuously, and if necessary retracted, on provable values for costs, with regard to risks. [VP 7.2, 3, 6, **17**]

# Why Engineering methods help projects to scale up.

I would like to attempt an explanation as to why the above principles, methods, tools and policies help us deal with very large systems.

1. **Value quantification** allows us to focus on the stakeholder results, the main objectives of any project. All other activity, below this level should be contributing to delivery of the planned values. This means we can delegate the activity to any combination of specialist teams of any size and complexity: yet we can judge whether things are 'working'. We keep our eyes on measured value delivery. We can judge whether both our organization and our architecture are delivering as expected and needed. If not we can adjust (**dynamic design to cost**) and go with things that are actually delivering necessary value.
2. **Contracting for value** relates to the above explanation, with the added benefit that outside contractors are now motivated to focus on value delivery, not just 'doing work', or 'programming'. It does not matter so much about the underlying complexity. That underlying complexity either works (delivers contracted value measurably) or not. If not, we change it until it does, or give up if we cannot change to satisfy value delivery needs.
3. **Decomposition by small 2% deliverable value architecture components**: this is a very basic attack on large size and consequent complexity. We can see the incremental impact of each step on the whole system, regarding both value delivery and costs. If it is not good enough we try new ideas. If we run out of ideas that work, we need to stop.
4. **Risk Management:** our methods, including 1-3 above, are really all about managing the risk of failing to deliver value for money, on time. In addition we have suggested a number of additional risk management ideas. For example estimating the ± uncertainty of a design impact on values and costs [9]. For example asking for specific evidence [9] that any given design, or strategy will deliver the values and costs we need. The more engineering effort we put in to planning for risk up front, the less likely we are to get nasty surprises later (and then blame them on 'project size and complexity'; rather than our own lack of decent engineering planning).
5. **Delegation of decision-making [23]**. Delegating the power to make decisions to a grass roots level, and in addition to do so incrementally while keeping any eye of their level of concern (in terms of value and costs), should obviously help us make better decisions, in an evidence-based situation.

I have personally used these methods, with remarkable success, on projects involving for example 1,000 programmers and 1,000 hardware engineers (example HICOM (which was in total failure mode after 2 years, at Siemens. Boeing Aircraft projects [thousands of employees involved. To mention just a couple of many). There is no doubt for me that they work, and why they work.

# Conclusion

I believe, based on my long term client experience, and other peoples experiences [5, 6, 10, 11, 22, VP (dozens of case studies) for example], that these ideas above are the basic engineering ideas we need to take seriously, if we are to get control over large-scale software projects.

The ideas above are described in practical detail in the references [especially 20, 22], have extensive documented practical international experience, and can be tried out one-by-one. They can be added to any other practices, that are, or will be successful for you, They are free ideas.

‘This stuff works!” (Erik Simmons, Intel, [22, 25])

Just do it!

# References


[1] "**Enterprise Scrum- Executive Summary:**

**Agile Management for the 21st Century"**

Authored, Developed and Sustained by Mike Beedle Enterprise Scrum Inc.

http://static1.1.sqspcdn.com/static/f/608893/25858383/1441617422010/Enterprise+Scrum+Executive+Summary.pdf?token=pl51QbvOlQDj0jML9CJdAuX08GM%3D


'Scaling' is **not** the primary purpose of this work, but the overview of references does give a fairly complete picture of the Scaling Writings.

Beedle says (7 Jan 2016 to me):
Enterprise Scrum is about 3 things: 1) genericity - agile management of ANY business process, 2) Techniques (some business–like) to work with the Enterprise Scrum framework, 3) Scaling (not only for software) — businesses, business processes, programs or projects.

In this 4th generation industrial revolution that we live in — I call it the the Innovation Revolution, where some companies are experiencing exponential growth after founding an MTP (massive transformative purpose), and are getting to get to 1 billion dollar valuations in record times … they need a different type of management that is more agile.


[2]

**A. The Agile Evo Project Startup Week Standard**
http://www.gilb.com/dl562

This is a detailed standard for conducting an 'Evo' (Evolutionary Project Management, Gilb's Agile Method) as described in my book Competitive Engineering, Chapter 10
http://www.gilb.com//tiki-download_file.php?fileId=77

**B. Evo Project Management** Standard, Jan 12 2013
http://www.gilb.com/dl563

[3]: **ONE WEEK STARTUP PLANNING** FOR PROJECTS; FRONT END TO EVO

- [3] A. **'An Agile Project Startup Week': Papers and slides**
  - Talk **slides** pdf from ACCU Conference, Bristol UK, April 9 2014
  - 90 minutes talk. Includes Startup Planning for Business Startups, Confirmit, US DoD case, 2 Bank cases, Detailed Startup week outlines and links to sources.

  http://www.gilb.com/dl812
  - 
- [3] B. See Persinscom Case
  - **"111111 Unity Method of Decomposition into weekly increments of value delivery"**. (10 min. talk slides)

  http://www.gilb.com/tiki-download_file.php?fileId=451
  - Includes Persinscom case US DoD.

o

- [3] C. **Software Plans in Less Than a Week**

  http://www.dtic.mil/ndia/2004cmmi/CMMIT7Tue/MelissaOlson.pdf
  - o Melissa Olson
    Raytheon Company McKinney, TX
    972-952-4502 Melissa_olson@raytheon.com Abstract #1196

[3] D. "An Agile Project Startup Week."
Gilb's Mythodology Column
www.gilb.com/dl568

[4] The **Top 10 Critical Requirements** are the Most Agile Way to Run Agile Projects
http://www.gilb.com/dl554

[5] **QUINNAN AND MILLS CLEANROOM**
**QUINNAN AND MILLS CLEANROOM**
http://www.gilb.com/dl821
is contained in these slides.
See reference [14]

[6] **'DEEPER PERSPECTIVES ON EVO DELIVERY** '
Chapter 15 in (1988) Principles of Software Engineering management
www.gilb.com/dl561
"Deeper Perspectives on Evolutionary Delivery"
plus a page extra of quotations from Agile Gurus crediting it as inspiration for them, and it being first.

http://www.gilb.com/dl821
is contained in these slides.

[7] **Decomposition of strategies by Value**
**A. "Decomposition of Projects - How to design small incremental result steps"**, 2008 Paper
http://www.gilb.com/tiki-download_file.php?fileId=41

The 20 decomposition principles alone, from CE [1] Ch. 10 on Evo are [in the VP book at Figure 5.6 A]

**B. "The Unity Method of Decomposition"**
Column 2 of Gilb's Mythodology
in Agile Record
http://www.gilb.com/dl826

**C. "111111 Unity Method of Decomposition into weekly increments of value delivery"**. (10 min. talk slides)
http://www.gilb.com/tiki-download_file.php?fileId=451
Includes Persinscom case US DoD.

[8] Competitive Engineering, Chapter 10: **Evolutionary Project Management:**
http://www.gilb.com//tiki-download_file.php?fileId=77

[9] **Impact Estimation**
Impact Estimation Table MASTER.ppt (8.49 Mb)
You can download this file using: http://www.gilb.com/tiki-download_file.php?fileId=146

**Design Evaluation** Paper
http://www.gilb.com/tiki-download_file.php?fileId=58

See **IE Table Chapter in CE Book** [22]

**Impact Estimation Tutorial** MASTER 2012_compressed.pdf
http://www.gilb.com/dl553

**Impact Estimation Tables**
**Understanding Complex Technology Quantitatively**
http://www.crosstalkonline.org/storage/issue-archives/1998/199812/199812-Gilb.pdf
Crosstalk, US DoD, December 1998.

[10] confirmit.com
**Confirmit** Case Study.

**A: The Green Week** Slides
http://www.gilb.com/dl660
Smidig/Agile Conference 2013 Oslo
Nov 5 2013

**B: The GREEN WEEK- Agile Technical Debt Engineering beats Refactoring.**
http://vimeo.com/78635151
Agile Oslo 2013, Video 10 minutes.
IN NORWEGIAN, English slides

C: Gilb, co author Trond Johansen **Confirmit case** paper
http://www.gilb.com/tiki-download_file.php?fileId=32

**D: Value Driven Project Management**
17.5MB *slides* 2008
Includes Firm Case
http://www.gilb.com/tiki-download_file.php?fileId=152

**E. 'What's Wrong With Agile Methods? Some Principles And Values To Encourage Quantification'** with Confirmit Case. http://www.gilb.com/dl50

[11] Hewlett Packard Case
**HP Evo**
**A. The Evolutionary Development Model for Software**
by Elaine L. May and Barbara A. Zimmer
August 1996 Hewlett-Packard Journal
http://www.gilb.com/tiki-download_file.php?fileId=67

**B. Evolutionary Fusion: A Customer- Oriented Incremental Life Cycle for Fusion**
by Todd A
http://www.gilb.com/tiki-download_file.php?fileId=35

August 1996 Hewlett-Packard Journal

**C. RAPID AND FLEXIBLE PRODUCT DEVELOPMENT: AN ANALYSIS OF SOFTWARE PROJECTS AT HEWLETT PACKARD AND AGILENT**
(2001)
by
Sharma Upadhyayula
http://www.gilb.com/tiki-download_file.php?fileId=65

M.S., Computer Engineering University of South Carolina, 1991
And
Massachusetts Institute of Technology
January 2001

**D. Best Practices for Evolutionary Software Development**
by
Darren Bronson
http://www.gilb.com/dl825

57 pages., 1999.

URI: http//hdl.handle.net/1721.1/80490

[12] **PRIORITIZATION (SEE CHAPTER 6 of Value Planning book)**

**A. Choice and Priority Using Planguage:**
**A wide variety of specification devices and analytical tools.**
Copyright © 2006 by Tom Gilb.
http://www.gilb.com/tiki-download_file.php?fileId=48

**B. Managing Priorities:**
**A Key to Systematic Decision-Making**
Tom Gilb Tom@Gilb.com Mark W. Maier Mark.w.maier@aero.org
Copyright © 2005 Tom Gilb and Mark Maier. Used by Permission of the authors by INCOSE.
http://www.gilb.com/tiki-download_file.php?fileId=60

**C. Dynamic Design Prioritization in the 'Evo' Agile Framework for Scrum or other Iterative Methods**
http://www.gilb.com/dl602
Slides up pdf 122 slides, for London Software Architect Conference Oct 9 2013

[13] Bring Case (Kai Gilb)
**Bring Case** and more: Hierarchical Impact Estimation Tables
http://www.gilb.com/dl500

**"Value-Driven Development: Principles and Values."**
Slides for , 50 minute talk, Software Passion Conference 20 March 2012 Gothenburg, Sweden

**Value Management**
**(Evo)**
**with Scrum development**, March 2010 English Version , Kai Gilb

www.gilb.com/tiki-download_file.php?fileId=277

**The Inmates are running the asylum, Construx Summit talk Oct 25 2011 Seattle**
**Contains considerable Bring Case slides**
www.gilb.com/tiki-download_file.php?fileId=488

Norwegian Version, Bring Case
www.gilb.com/tiki-download_file.php?fileId=279

[14] IBM Cleanroom (Large Scale Agile in 1970s)
**Cleanroom, See [5]**
A. Mills, H. 1980. The management of software engineering: part 1: principles of software engineering. *IBM Systems Journal* 19, issue 4 (Dec.):414-420.
Direct Copy
http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan
Library header
http://trace.tennessee.edu/utk_harlan/5/

B. Mills, Harlan D.; Dyer, M.; and Linger, R. C., "Cleanroom Software Engineering" (1987). The Harlan D. Mills Collection. http://trace.tennessee.edu/utk_harlan/18

C. Mills Generally
http://trace.tennessee.edu/utk_harlan

[15] **User Stories**
A. User Stories paper by Tom and Kai Gilb
In Gilbs' Mythodology Column, Agilerecord.com March 2011
http://www.gilb.com/tiki-download_file.php?fileId=461
**"User stories and the conversations provoked by them comprise *verbal communication*, which is clearer than written communication." (Mike Cohn via Denning)**
B. Stephen. Denning, The Leaders Guide to Radical Management.
http://stevedenning.typepad.com

[16] Richard Smith's Planguage Tool: '**Needs and Means**'
http://needsandmeans.com
There have been several tools supporting Planguage. This tool by Richard Smith
rsmith@rsbatechnology.co.uk

is emerging in 2015. We have used it on training courses in 2015. I am impressed by it capabilities and ease of use. It is very helpful on the *dynamic prioritization* methods.
Free Trial Use is bundled with the VP book [20]


[17]
The Logic of Design: Design Process Principles.
http://www.gilb.com/dl857

T Gilb
14 Oct 2015

[18]
What are the Dangers of Current Agile Practices, and How Can We Fix Them?
Values for Value
http://www.gilb.com/tiki-download_file.php?fileId=456

by Tom Gilb & Lindsey Brodie

[19] Some Alternative Ideas On Agile Values For Delivering Stakeholder Value Principles and Values – Agility is the Tool, Not the Master.
Part 2 "Values for Value"
http://www.gilb.com/tiki-download_file.php?fileId=448

Agile Record 2010, www.agilerecord.com, October 2010, Issue 4

[20: VP] "**Value Planning: Practical Tools for Clearer Management Communication**"

leanpub.com/ValuePlanning

Free sample (Part 0 to 5), and ridiculously cheap download with bundled tools and updates.

If any reader cannot afford, or has technical problems paying the minimum $1 fee, I'll send them a free link to the text so they can explore all references. Email me.

[21] Contracting

**No Cure No Pay Contracting**

A. Agile Contracting for Results The Next Level of Agile Project Management: Gilb's Mythodology Column, in Agile Record August 2013.
http://www.gilb.com//dl581
see in that respect
www.flexiblecontracts.com

B. No Cure Slides
http://www.gilb.com/tiki-download_file.php?fileId=85

C. NO Cure Paper
http://www.gilb.com/tiki-download_file.php?fileId=38

D. Contracting For Value. Slides 2015
http://www.gilb.com/dl864

[22] Gilb: **Competitive Engineering**: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage, ISBN 0750665076, 2005, Publisher: Elsevier Butterworth-Heinemann., 2005
http://www.gilb.com//dl540
For your *personal* use only. Do not refer to or publicize this URL.
The Paper Book, and E versions Book are for sale. This is a simple pdf.
See Download [24]

[23] Power to the Programmers

"**POWER TO THE PROGRAMMERS"** TALK SLIDES AND VIDEO

Note Paul Klipp has transcribed this and published my talk in a book. 21.7.14 mail hpps://leanpub.com/ACE2014

http://www.gilb.com/dl821
(Slides)
**Power to The Programmers**, as held Krakow ACE Conference June 2014
Video: http://vimeo.com/98733453

[24] **Case of Siemens 1,000 Programmer project.**
Additional Text (to the CE book [22]).
Chapter 10: Evolutionary Project Management: see page 315
http://www.gilb.com//tiki-download_file.php?fileId=77

There is a second story here, related to our methods. They had focussed on a main quantified goal of low bugs per thousand lines of code. Three people were set aside to manage and report on that alone. I actually met them.
I suggested this was the wrong major metric. I suggested that they needed to focus on the Availability metric. They agreed, but said they did not know how to measure this in software. So they chose a metric they knew how to measure! Bad mistake.

I pointed out the Books of my friend John Musa (ATT) on measurements of availability in telecoms software. And suggested they use common sense (lift up handset 100 times and count % dial tone!). They agreed to focus on their Availability Metric. They were in danger of failing to deliver one of their critical values.

[25] with his kind permission, you might like the views of a large Scale (20,000+ Intel Employees over many years) adopter of these methods: Erik Simmons

On 08 Jan 2016, at 19:30, Simmons, Erik <erik.simmons@intel.com> wrote:
Just a couple of things come to mind after reading this:

I've not been a fan of the scaling movement since it started. There are very few things that scale well, and economies of scale are often pursued without adequate understanding of the accompanying diseconomies of scale. SW development does not scale well because of the diseconomies of complexity, such as the number of communication pathways, cognitive load on programmer brains, etc. That is among the core reasons for Brook's Law.

What makes us think that scaling Scrum, which is successful in small teams and projects, is a good idea? A grown-up is not a scaled baby. Scaling as a concept is selling a lot of books, consulting, and certifications right now. But I don't think it is a valuable concept.

Instead, I believe that the majority of what you have included for ideas, principles, etc. from CE and VP are in fact scale-free. They are not dependent on project or organization size. They are good heuristics for almost any project, and nearly universally applicable (nearly universal because I hear Koen in my head, and all is heuristic). So, CE and VP are not about scaling so much as they should be taught and understood as scale-free. Size is not a reason to choose (or not choose) to use CE, Evo, Planguage, etc. As you quoted me in the paper – this stuff works. It works on small projects. It works on large projects. Evo on a 5-person team is not really much different than Evo on a 100-person team, except there are more people. The principles apply without alteration (or "scaling"). Anyone who sees a random page of your new paper would probably not guess the topic is scaling (unless you happen to mention that in the text on that particular page). CE does not scale. *It doesn't need to*.

There's no doubt that large projects are different. There's no doubt that we should approach them differently. We still don't have a recipe for large projects, and probably never will. But all that does not lead me to think that the answer to large projects can be found in scaling successful practices for small projects. Instead, it must be found in use of principles and practices that are scale-free, coupled with use of particular practices that are effecting on large projects. If something that works on small projects also works on large projects, then I'd propose we call it a scale-free practice, not a scaled practice.

I'm deeply interested in scale-free practices. I'm also interested in specific practices tuned to large, small, complicated, and complex projects, but I find particular power in scale-free practices. Your work for decades has been focused on a very good set of these. SQC, for example, works on any size specification. It does not (need to) scale.

BTW, I think the agile principles are also quite scale-free. But most Scrum practices are definitely not.

So, perhaps you can chart a better course by advocating for use of scale-free core practices, augmented with a set of specific, tailored practices that are effective for the size of the project in question.

Cheers,
e

FOOTNOTES

F1: Reaction to the paper from a major author and developer of Agile Scaling Methods: Demonstrates my point!

Hi,

Thanks for sharing. I do not agree with them though. I do not think rigorous measurements is going to improve things.

We do need more programmers as most aren't able to write proper code yet.

To extend your analogy. We wouldn't want to focus on "engineering skyscrapers" if we don't even have proper carpenters.

Anyways, thanks for sharing!

paper: version January 8 2016, Edit 20:46 CET

on gilb.com papers
http://www.gilb.com/dl865
Has been updated several times already



Thanks to the following for advice which i have taken in editing the paper:
Mary Poppendieck
Erik Simmons