

‘Agility is the TOOL, not the Master’ : Practical Agile *Systems Engineering* Tools

including

My Ten Key Agile Principles
and several case studies



Tom Gilb

tomsgilb@gmail.com

@imtomgilb

www.gilb.com

These slides are on gilb.com downloads, and twittered at #BuildstuffLT
The Talk was videoed.

Paper: Value-Driven Development Principles and Values - Agility is the Tool, Not the Master
July 2010 Issue 3, Agile Record 2010 (www.AgileRecord.com)

http://www.gilb.com/tiki-download_file.php?fileId=431

<http://www.slideshare.net/tomgilb1/agility-is-the-tool-gilb-vilnius-9-dec-2013>

Agile Principles : Agile Record Paper as Published

http://www.gilb.com/tiki-download_file.php?fileId=431

Value-Driven Development Principles and Values - Agility is the Tool, Not the Master
July 2010 Issue 3, Agile Record 2010 (www.AgileRecord.com)

http://www.gilb.com/tiki-download_file.php?fileId=431

June 9, 2014

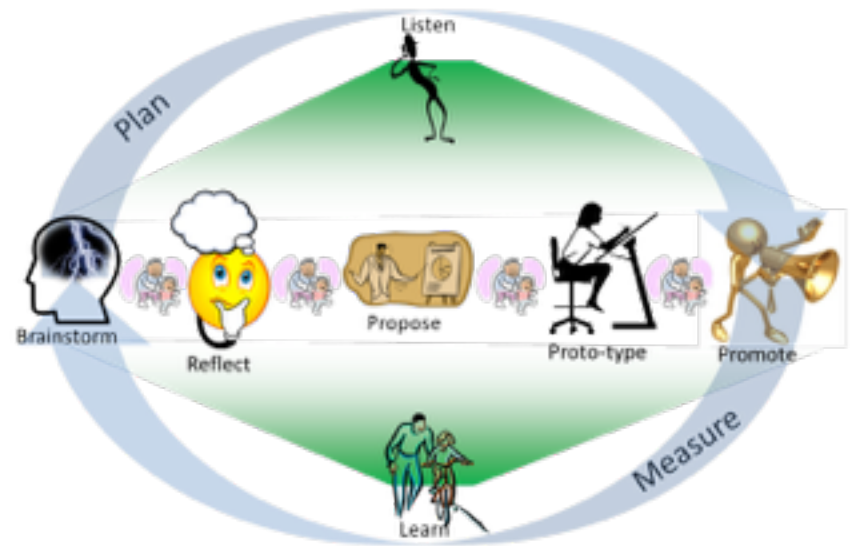
© Gilb.com

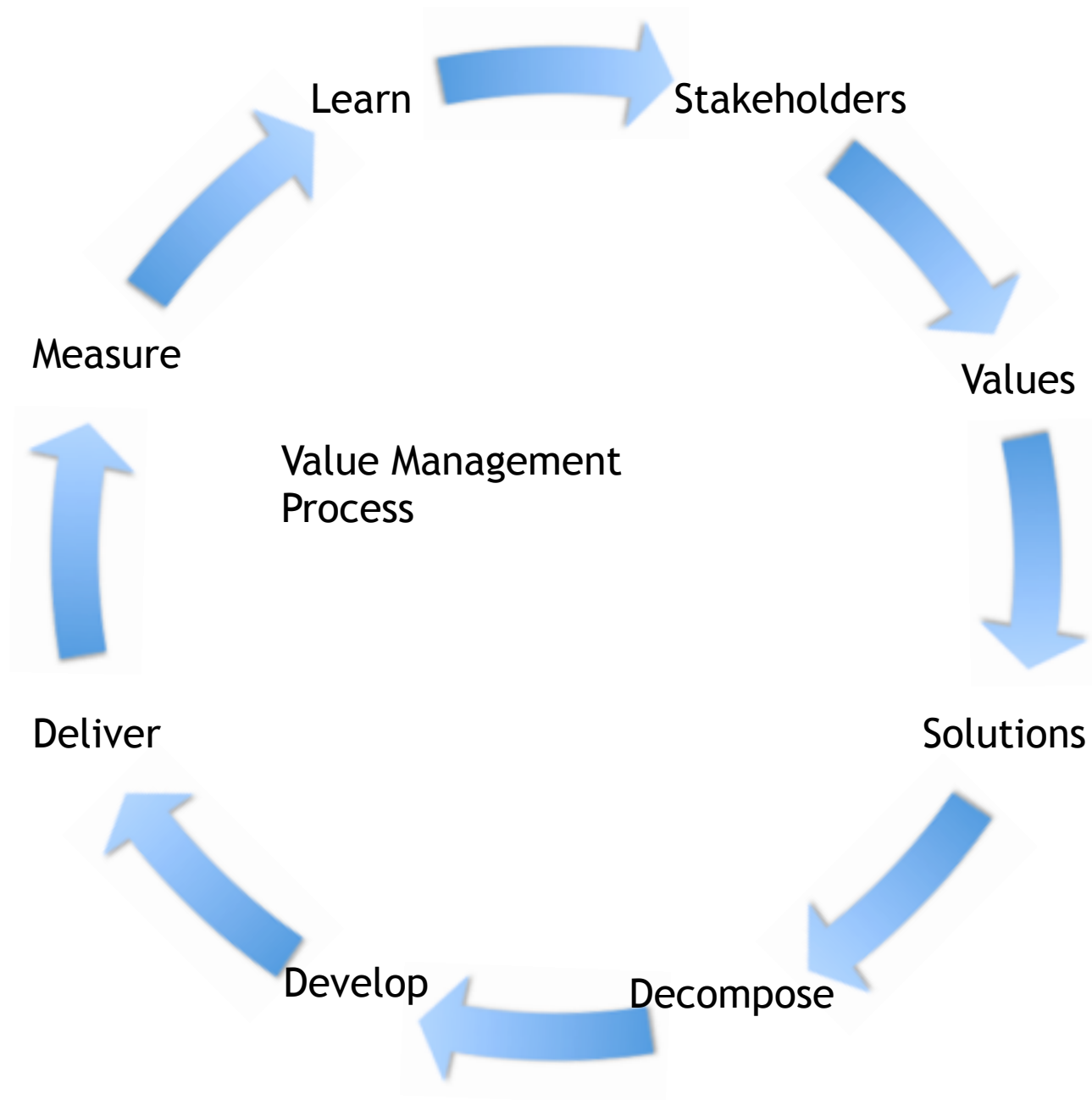
Agility is the Tool

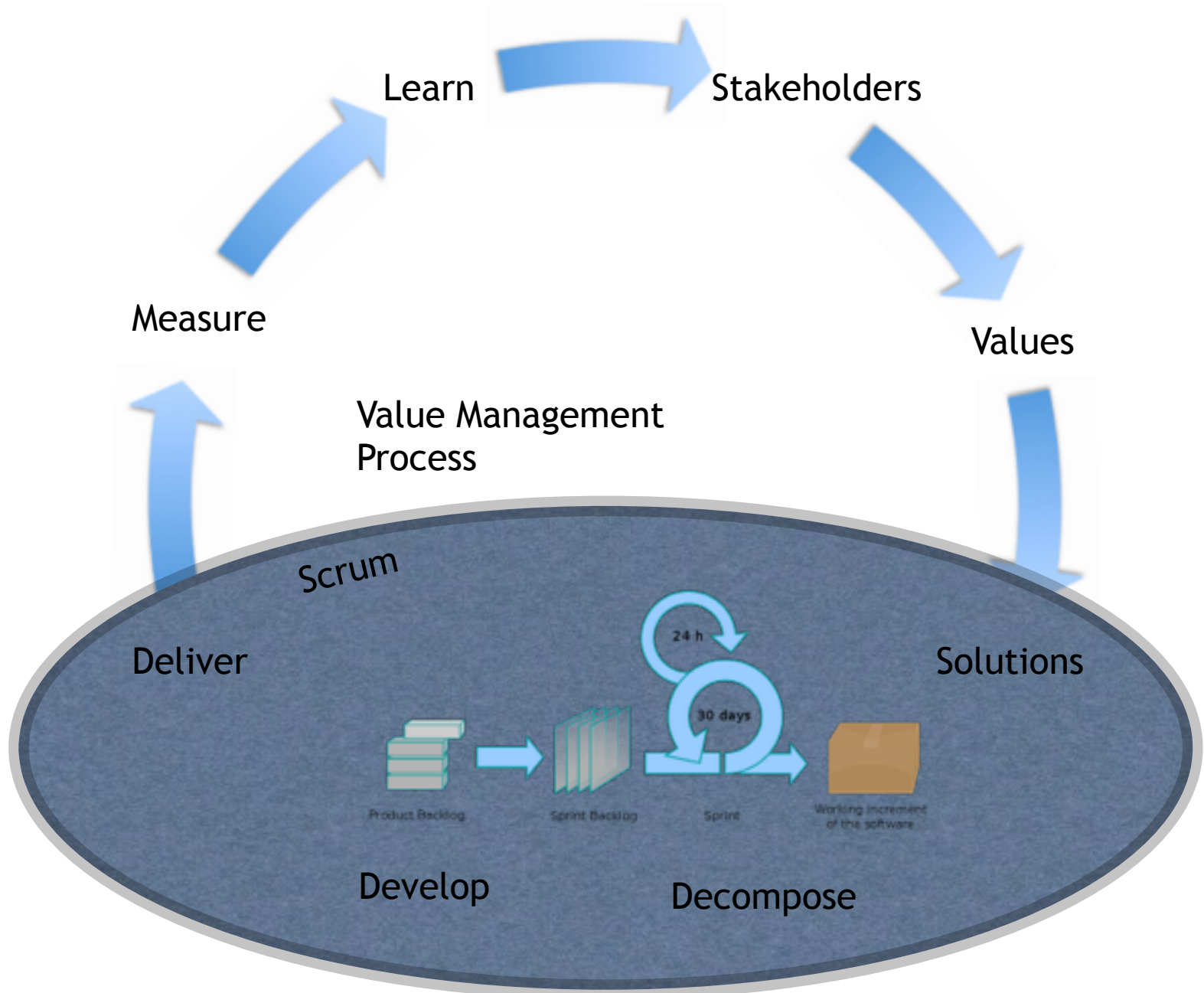
Defining ‘Agile’

- “Any set of tactics that **enable a prioritised stream of useful results, *in spite of a changing environment***”
 - TsG 7 June 2013
- A focus on ‘Agile’, is the *wrong* level of focus.
 - Using agile tactics that ‘work’, is a good idea.
- **Focus on *results*, no matter what.**
- As a government minister, I was asked to give ideas to, put it “Value for Money”

The Generic Agile Concept

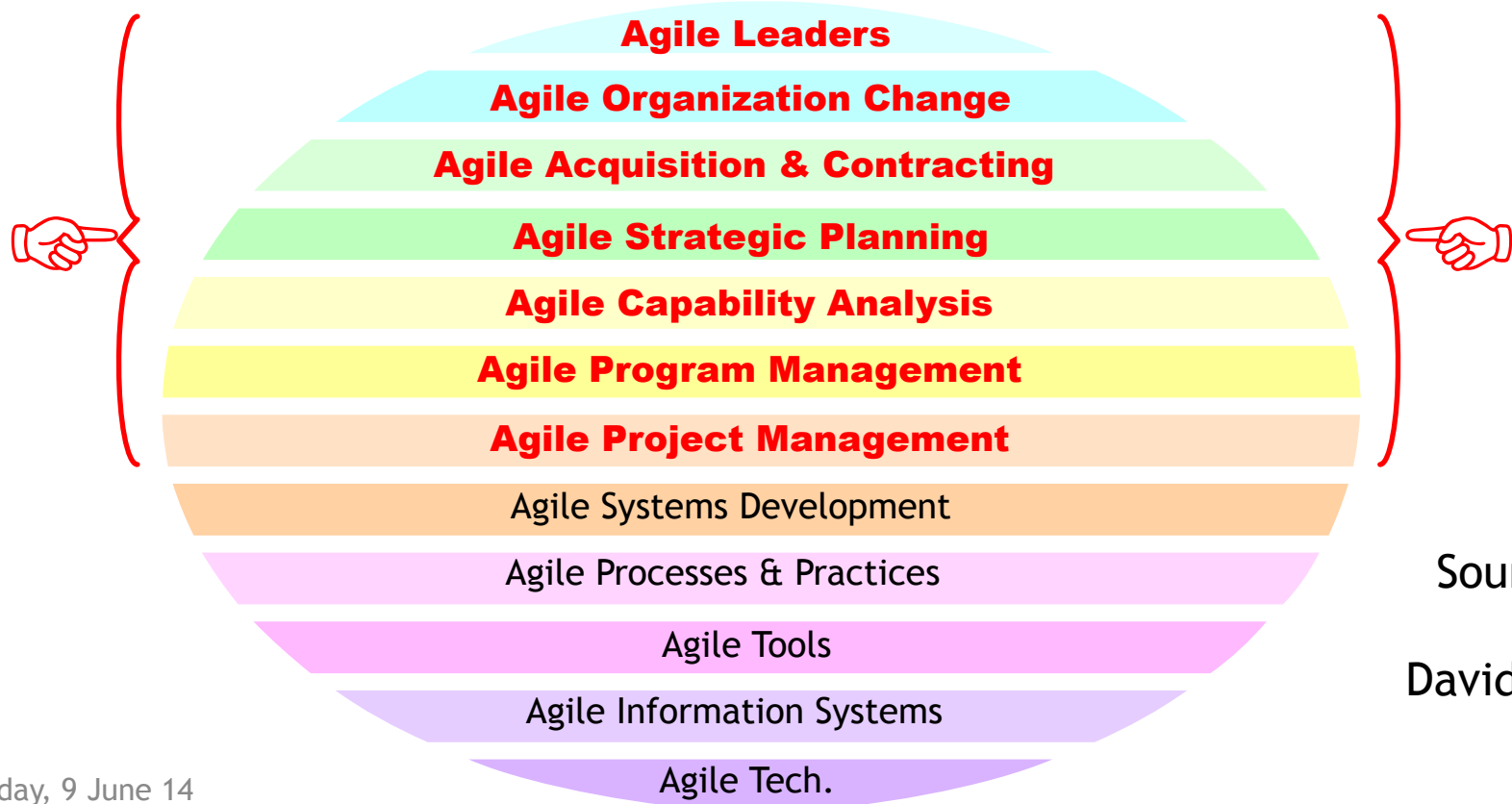






Agile World View

- “Agility” has many dimensions other than IT
- It ranges from leadership to technological agility
- The focus of this brief is program management agility



Agile Recap

- Agile methods **DON'T** mean deliver it now & fix it later
- Lightweight, yet disciplined approach to development
- Reduced cost, risk & waste while improving quality

What	How	Result
Flexibility	Use lightweight, yet disciplined processes and artifacts	Low work-in-process
Customer	Involve customers early and often throughout development	Early feedback
Prioritize	Identify highest-priority, value-adding business needs	Focus resources
Descope	Descope complex programs by an order of magnitude	Simplify problem
Decompose	Divide the remaining scope into smaller batches	Manageable pieces
Iterate	Implement pieces one at a time over long periods of time	Diffuse risk
Leanness	Architect and design the system one iteration at a time	JIT waste-free design
Swarm	Implement each component in small cross-functional teams	Knowledge transfer
Collaborate	Use frequent informal communications as often as possible	Efficient data transfer
Test Early	Incrementally test each component as it is developed	Early verification
Test Often	Perform system-level regression testing every few minutes	Early validation
Adapt	Frequently identify optimal process and product solutions	Improve performance

Source:
David Rico

Rico, D. F. (2012). *What's really happening in agile methods: Its principles revisited?* Retrieved June 6, 2012, from <http://davidfrico.com/agile-principles.pdf>

Rico, D. F. (2012). *The promises and pitfalls of agile methods.* Retrieved February 6, 2013 from

Rico, D. F. (2012). *How do lean & agile intersect?* Retrieved February 6, 2013, from <http://davidfrico.com/agile-concept-model-3.pdf>

14 PITFALLS OF AGILE METHODS

- **Change** – Use of top-down, big-bang organization change, adoption, and institutionalization.
- **Culture** – Agile concepts, practices, and terminology collide with well-entrenched traditional methods.
- **Acquisition** – Using traditional, fixed-price contracting for large agile delivery contracts and projects.
- **Misuse** – Scaling up to extremely complex large-scale projects instead of reducing scope and size.
- **Organization** – Unwillingness to integrate and dissolve testing/QA functional silos and departments.
- **Training** – Inadequate, insufficient, or non-existent agile training (and availability of agile coaches).
- **Infrastructure** – Inadequate management and development tools, technologies, and environment.
- **Interfacing** – Integration with portfolio, architecture, test, quality, security, and usability functions.
- **Planning** – Inconsistency, ambiguity, and non-standardization of release and iteration planning.
- **Trust** – Micromanagement, territorialism, and conflict between project managers and developers.
- **Teamwork** – Inadequate conflict management policies, guidelines, processes, and practices.
- **Implementation** – Inadequate testing to meet iteration time-box constraints vs. quality objectives.
- **Quality** - Inconsistent use of agile testing, usability, security, and other cost-effective quality practices.
- **Experience** - Inadequate skills and experience (or not using subject matter experts and coaches).
- *(Note. Firms may prematurely "revert" to inexorably slower and more expensive traditional methods or "leap" onto lean methods that may not adequately address common pitfalls of adopting agile methods.)*
- Source: David Rico <http://davidfrico.com/agile-pros-cons.pdf> 2012



Value Driven Scrum

8

System Owner

- **Stakeholders Values**
- **Business Values**
- **System Functions**



Product Owner

- **Build**
- **Test**
- **Maintain**
- **Detailed Technical Design**

+

Value Decision Tables

9

Business Goals	Training Costs	User Productivity
Profit	-10%	40%
Market Share	50%	10%
Resources	20%	10%

Stakeholder Val.	Intuitiveness	Performance
Training Costs	-10%	50 %
User Productivity	10 %	10%
Resources	2 %	5 %

Product Values	GUI Style Rex	Code Optimize
Intuitiveness	-10%	40%
Performance	50%	80 %
Resources	1 %	2 %

Jeffsutherland
Twitter: Very cool
product backlog
management
by Tom and Kai
Gilb <http://ad.vu/2h4d> Sat 28
March 2009



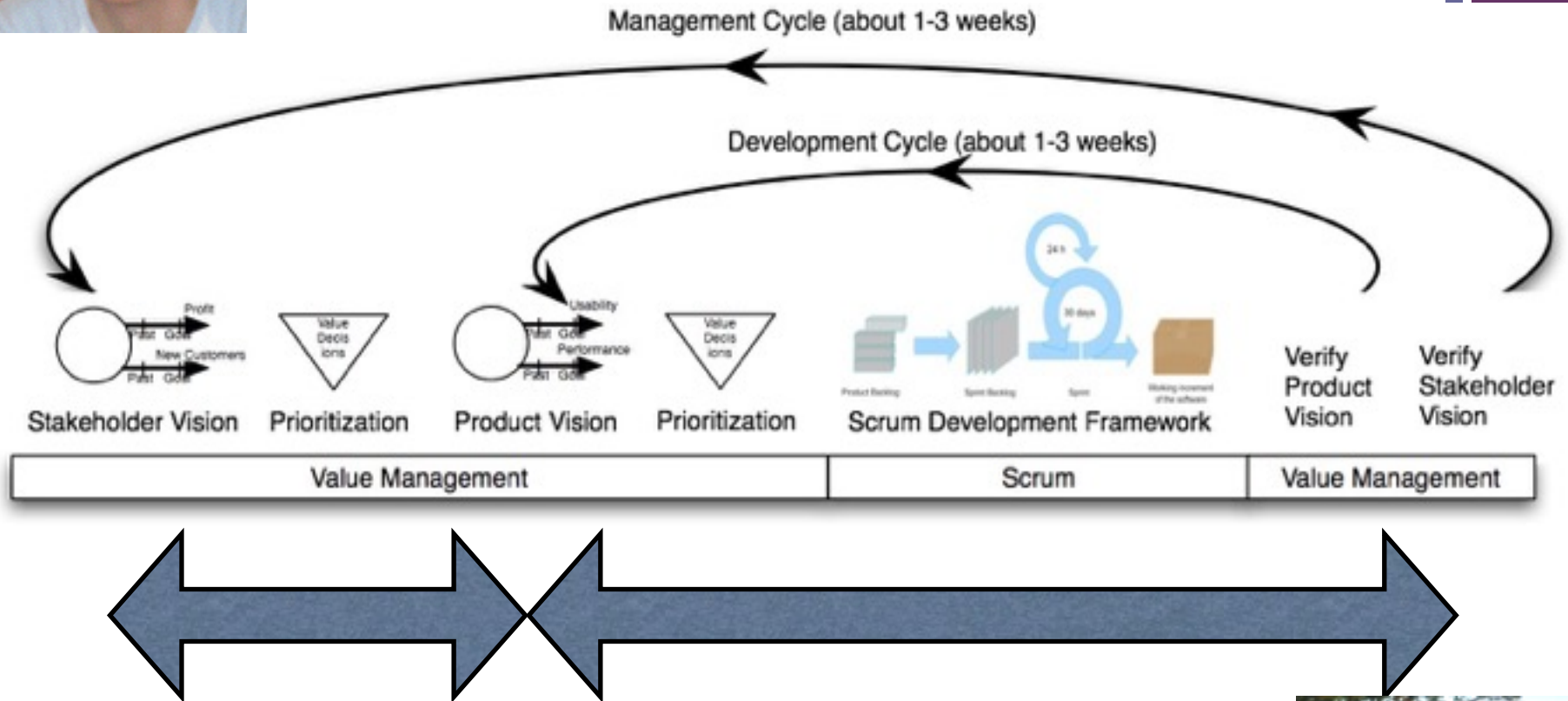
We measure
 improvements
 Learn and Repeat

Scrum Develops



Prioritized List
1. Code Optimize
2. Solution 9
3. Solution 7





June 9, 2014

Jeffsutherland Twitter: Very cool product backlog management by Tom and Kai Gilb <http://ad.vu/2h4d> Sat 28 March 2009

Gilb's Ten Key Agile Principles

to avoid bureaucracy and give creative freedom

(see Polish & Eng. Paper on this!) Core, Agilerecord.com, Gilb.com

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 in relation to 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 in quantified experience, new inputs

9. Involve the stakeholders, every week, in setting quantified goals

10. Involve the stakeholders, every week, in actually using increments



Gilb's Agile Principles


to avoid bureaucracy and give creative freedom (1 sentence summary)



Main Idea:

Get early, and frequent, real, stakeholder net-value - delivered

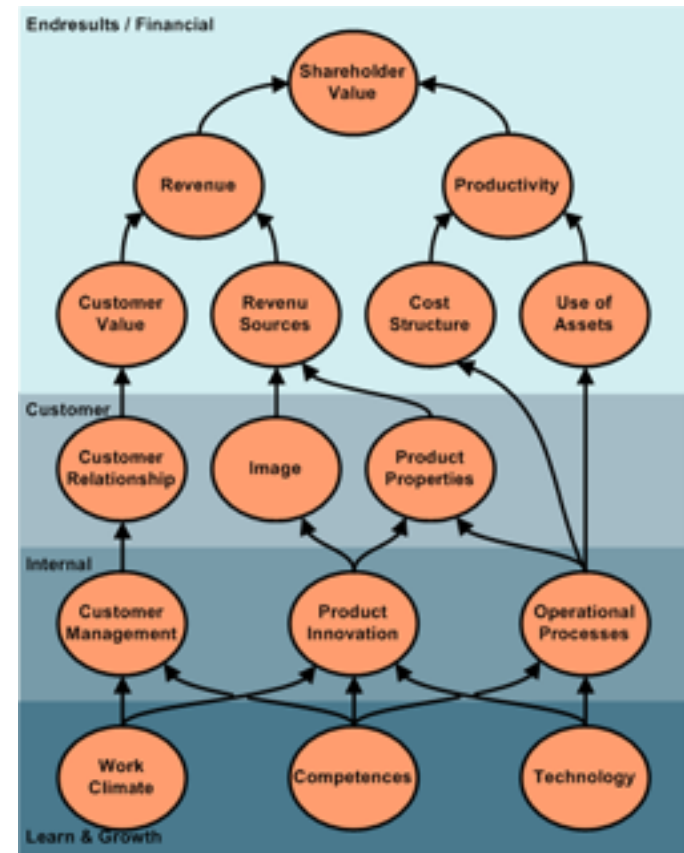
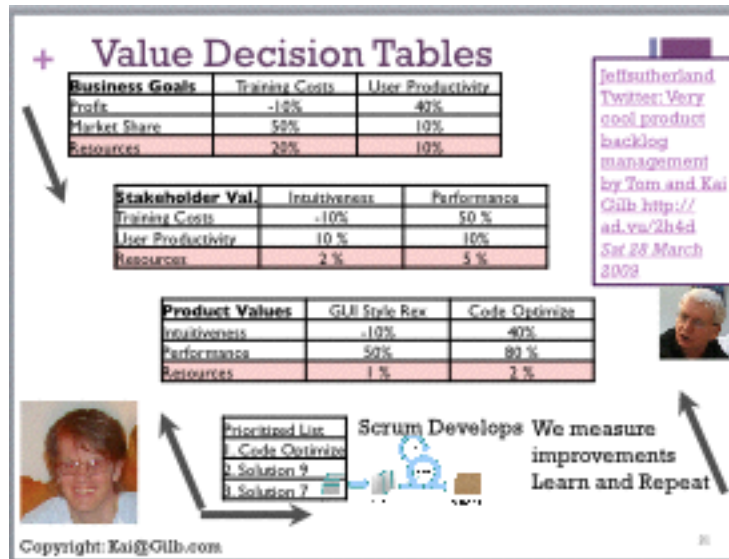
	VALUE TO CREATE	VALUE TO PRESERVE	VALUE TO SACRIFICE
EMPLOYEES			
CUSTOMERS			
SUPPLIERS AND PROFESSIONAL ADVISERS			
INVESTORS			
TRADES UNIONS			
GOVERNMENT			
MEDIA			
COMMUNITY			
OTHER STAKEHOLDER GROUPS			



**Deliver
Value !**

1. Control projects by quantified critical-few results. 1 Page total !

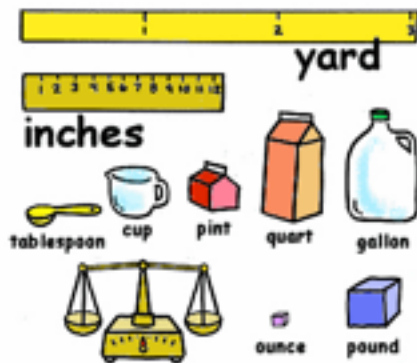
(not stories, functions, features, use cases, objects, ..)



NOT LIKE THIS! Project Objectives

'Unquantified few'

- ▶ **Defined** Scales of Measure:
 - Demands *comparative thinking*.
 - Leads to requirements that are unambiguously **clear**
 - Helps Team be **Aligned** with the Business



Real Example of Lack of CLARITY

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 clarity cost them \$100,000,000

More like this! (Real Example).

Business objective	Measure	Goal (200X)	Stretch goal (0X)	Volume	Value	Profit	Cash
Time to market	Normal project time from GT to GT5	<9 mo.	<6 mo.	X		X	X
Mid-range	Min BoM for The Corp phone	<\$90	<\$30	X		X	X
Platformisation Technology	# of Technology 66 Lic. shipping > 3M/yr	4	6	X		X	X
Interface	Interface units	>11M	>13M	X		X	X
Operator preference	Top-3 operators issue RFQ spec The Corp	1	2	X		X	X
Productivity						X	X
Get Torden	Lyn goes for Technology 66 in Sep-04	Yes		X		X	X
Fragmentation	Share of components modified	<10%	<5%		X	X	X
Commoditisation	Switching cost for a UI to another System	>1yr	>2yrs			X	X
	The Corp share of 'in scope' code in best-selling device	>90%	>95%		X	X	X
Duplication					X		
Competitiveness	Major feature comparison with MX	Same	Better	X		X	X
User experience	Key use cases superior vs. competition	5	10	X	X	X	X
Downstream cost saving	Project ROI for Licensees	>33%	>66%	X	X	X	X
Platformisation IFace	Number of shipping Lic.	33	55	X		X	X
Japan	Share of of XXX sales	>50%	>60%	X		X	X

<- Business Objectives Quantified

Numbers are intentionally changed from real ones

Real EXAMPLE of Objectives/Strategy definitions

US Army Example: PERSINSCOM: Personnel System



Example of one of the Objectives:

Customer Service:

Type: Critical Top level Systems Objective

Gist: Improve customer perception of quality of service provided.

Scale: Violations of Customer Agreement per Month.

Meter: Log of Violations.

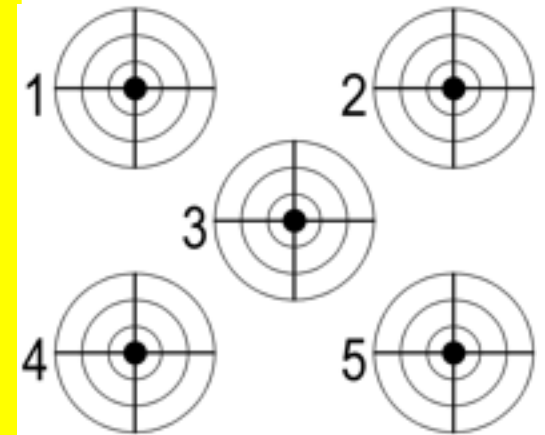
Past [Last Year] Unknown Number ← State of PERSCOM Management Review

Record [NARDAC] 0 ? ← NARDAC Reports Last Year

Fail : <must be better than Past, Unknown number>

←CG

Goal [This Year, PERSINCOM] 0 “Go for the Record” ← Group SWAG



Make sure those results are business results, not **JUST** technical

Align your project with your financial sponsor's interests!

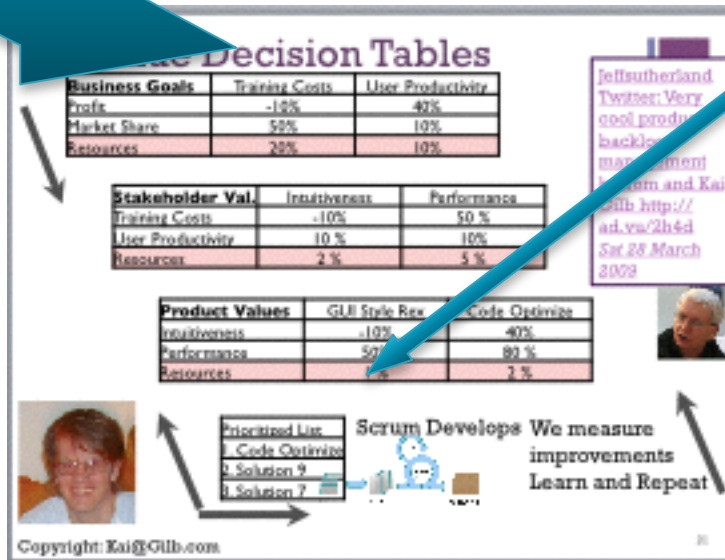


Figure 1. The "Mother of All Models". © 2006 MarketingNPV LLC. All Rights Reserved.

The Strategic Objectives (CTO level) Example from Ericsson Base Stations

- ***the Fundamental Objectives (Profit, survival)***
- ***Software Productivity:***
 - *Lines of Code Generation Ability*
- ***Lead-Time:***
- ***Predictability.***
- ***TTMP: Predictability of Time To Market:***
- ***Product Attributes:***
- ***Customer Satisfaction:***
- ***Profitability:***



‘Means’ Objectives which support Strategic Objectives:
all quantified in practice,
see URL below

- Support the **Strategic Objectives**
 - *Complaints:*
 - *Feature Production:*
 - *Rework Costs:*
 - *Installation Ability:*
 - *Service Costs:*
 - *Training Costs:*
 - *Specification Defectiveness:*
 - *Specification Quality:*
 - *Improvement ROI:*



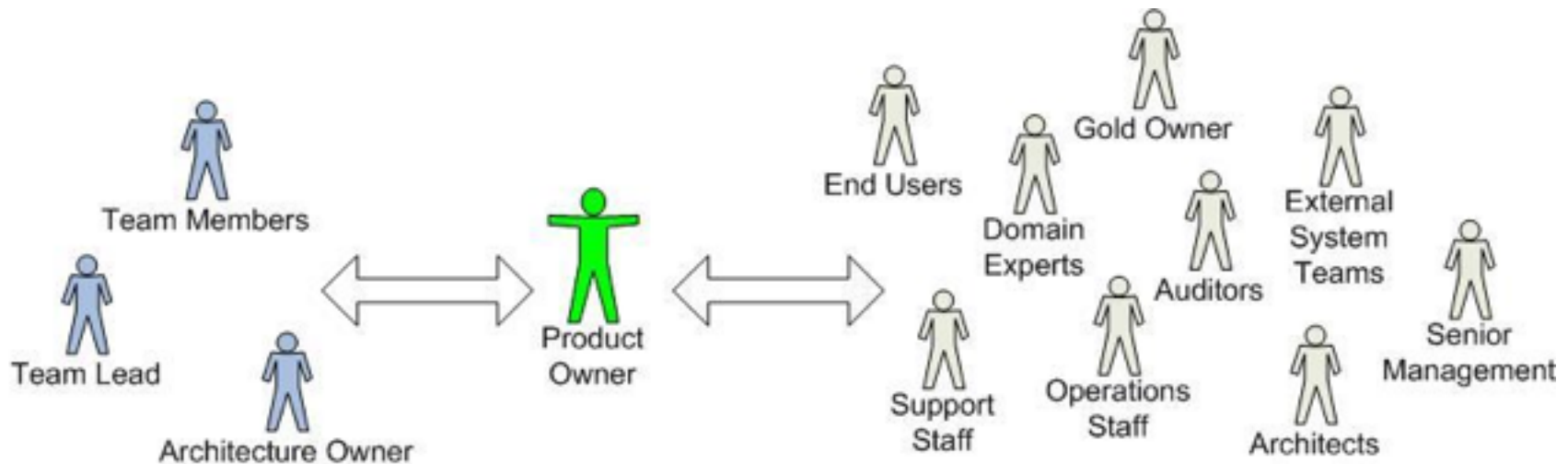
"Let no man turn aside,
ever so slightly,
from the broad path of honour,
on the plausible pretence
that he is justified by the goodness
of his end.

All good ends can be worked out
by good means."

Charles Dickens

Productivity Slides incl Ericsson
<http://www.gilb.com/dl559>

Simple Product Owner (Ambler)



Copyright 2005-2010 Scott W. Ambler

<http://www.agilemodeling.com/essays/productOwner.htm>

‘Advanced Product Owner’ Policy: System ‘Requirements Engineer’ (RE).

Background: this policy defines the expectations for a **‘Product Owner’ (PO) for serious, critical, large, and complex systems.**

1. This implies that it is not enough to manage a simple stream (Backlog) of ‘user stories’ fed to a programming team.
2. It is necessary to communicate with a ***systems engineering*** team, developing or maintaining the ‘Product’.

System implies management of all technological components, people, data, hardware, organization, training, motivation, and programs.

Engineering: means systematic and quantified, ‘real’ engineering processes, where proactive design is used to manage system performance (incl. all qualities) attributes and costs.



New idea being drafted by TG for a Client Bank, 7.12.2013

'Advanced Product Owner' Policy: System 'Requirements Engineer' (RE).

1. COMPLETE REQUIREMENTS:

The RE (Requirements Engineer) is responsible for absolutely all requirements specification that the system must be aware of, and be responsible for to all critical or relevant stakeholders.

In particular, the RE is not narrowly responsible for requirements from users and customers alone. They are responsible for all other stakeholders, such as operations, maintenance, laws, regulations, resource providers, and more.

2. QUALITY REQUIREMENTS:

The RE is responsible for the quality level, in relation to official standards, of all requirements they transmit to others.

They are consequently responsible for making sure the quality of incoming raw requirements, needs, values, constraints etc. is good enough to process. No GIGO.

If input is not good quality, they are responsible for making sure it is better quality, or at least clearly annotated where there is doubt, incompleteness, ambiguity and any other potential problems, they cannot resolve yet.

3. ARCHITECTURE:

The Requirements Engineer is NOT responsible for any architecture or design process itself. This will be done by professional engineers and architects.

They are however very much responsible for a complete and intelligible quality set of requirements, transmitted to the designers and architects.

They are also responsible for transmitting quality-controlled architecture or design specifications to any relevant system builders. These are the designs which are input requirements to builders. Effectively they are 'design constraints requirements'.

4. PRIORITY INFORMATION:

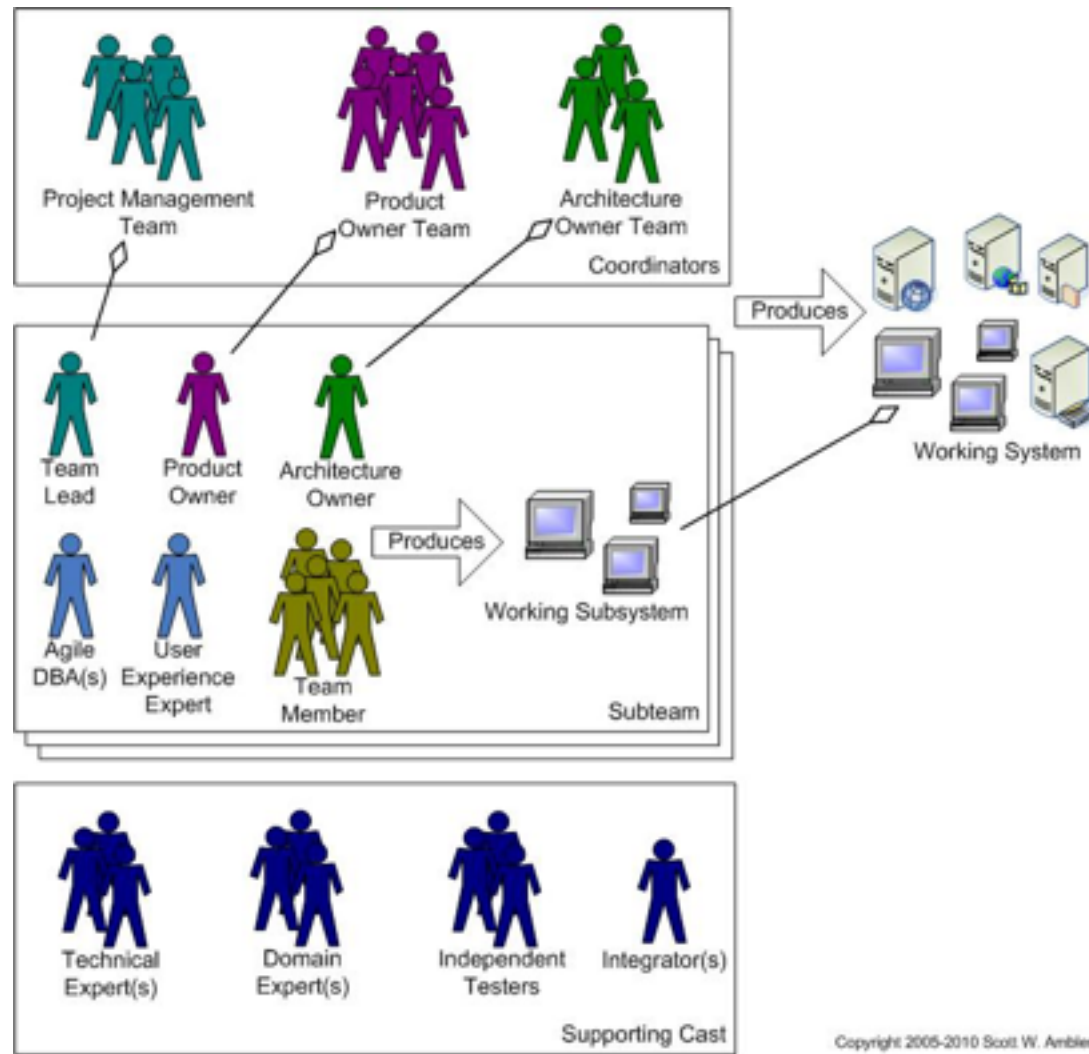
The Requirements Engineer is NOT responsible for prioritization of requirements.

Prioritization is done dynamically at the project management (PM) level, based on prioritization signals in the requirements, and on current feedback and experience in the value delivery cycles (Sprints).

The primary responsibility of the Requirements Engineer, is to systematically and thoroughly collect and disseminate all relevant priority signals, into the requirement specification; so that intelligent prioritization can be done at any relevant level, and at any time.

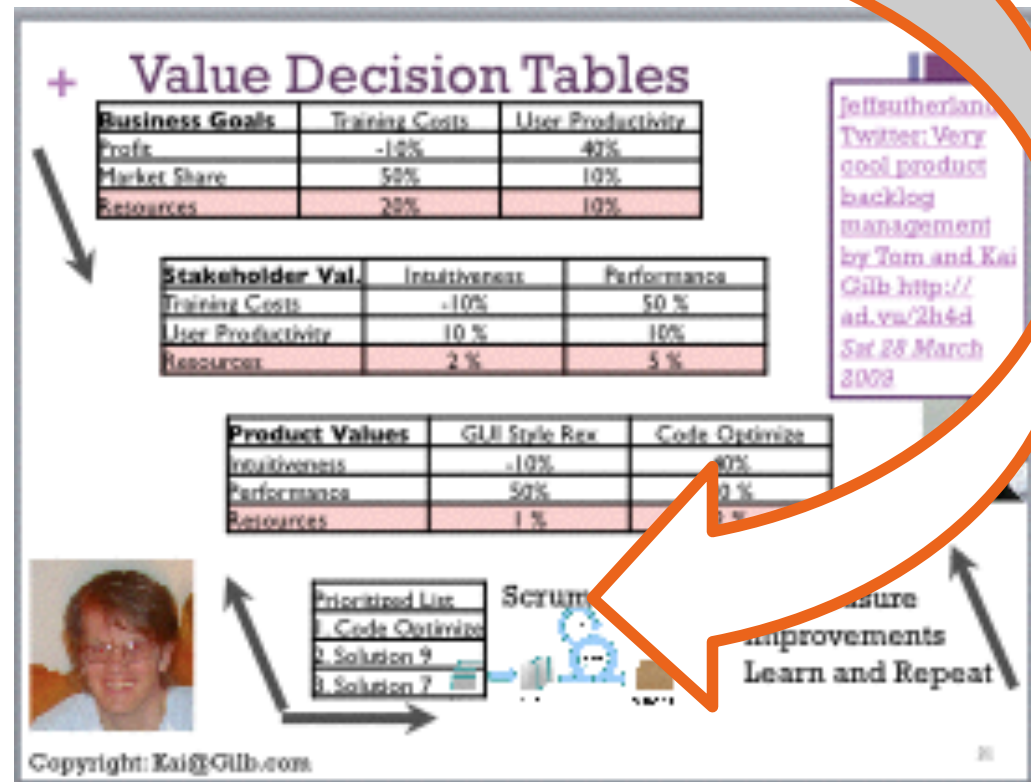
New idea being drafted by TG for a Client Bank, 7.12.2013

Product Owner at Scale (Ambler)



<http://www.agilemodeling.com/essays/productOwner.htm>

3. Give developers freedom, to find out *how* to deliver those results



Principle 4. Estimate the impacts of your designs, on *your* quantified goals



If you cannot, then your knowledge is of a meagre and unsatisfactory kind (Lord Kelvin)

Quantified Value Delivery Project Management in a Nutshell (Confermit Case, Norway)
Quantified Value Requirements, Design, Design Value/cost estimation, Measurement of Value Delivery, Incremental Project Progress to Date

	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
16						Development resources					
17											
18											
19											
20											
21			101,0	91,8		0		110	4,00	3,64	4,00

Estimates

Testing

Weekly

Priority
Next
week
Warning
metrics
based

Cumulative
weekly
progress
metric

Constraint

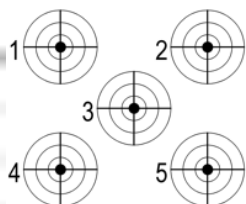
Target

REAL EXAMPLE: Strategy Impact Estimation:
for a \$100,000,000 Organizational Improvement Investment



Technical Strategies

Objectives



Viking Dumbles

Business Objective	hardware adaptation	Telephony	Reference design	Face	Modularity	Defend vs Technology	Tools	User Expertise	GUI & Graphics	Security	Defend vs OCD	Enterprise
Time to market	20%	10%	30%	5%	10%	5%	15%	0%	0%	0%	5%	5%
Mid-range	15%	0%	15%	0%	30%	15%	5%	10%	5%	5%	0%	0%
Platformisation Technology	25%	0%	10%	0%	10%	10%	0%	5%	0%	10%	0%	5%
Interface	5%	15%	15%	0%	5%	0%	5%	0%	0%	10%	0%	10%
Operator preference	0%	0%	0%	15%	5%	20%	5%	10%	10%	20%	5%	10%
Get Torden	25%	10%	10%	15%	0%	20%	0%	10%	-20%	10%	10%	5%
Commoditisation	20%	10%	20%	10%	-20%	25%	15%	0%	0%	5%	10%	5%
Duplication	15%	0%	10%	0%	0%	40%	0%	0%	0%	5%	20%	5%
Competitiveness	10%	15%	20%	0%	10%	20%	10%	10%	20%	10%	10%	10%
User experience	5%	0%	0%	0%	0%	0%	0%	30%	10%	0%	0%	0%
Downstream cost saving	15%	0%	0%	5%	10%	20%	0%	10%	0%	0%	10%	5%
Platformisation I face	10%	10%	20%	40%	0%	20%	5%	0%	0%	0%	0%	5%
Japan	10%	5%	20%	0%	10%	0%	0%	10%	5%	0%	0%	0%
Contribution to overall result	15%	9%	17%	4%	10%	10%	10%	10%	10%	10%	5%	5%
Cost (£M)	2.85	0.49	3.21	2.54	1.92	2.31	0.81	1.21	2.68	0.79	0.62	0.60
ROI Index (100=average)	108	358	100	33	78	133	148	107	10	152	202	174

"Benefits"

Strategy
Impacts
on
Objectives

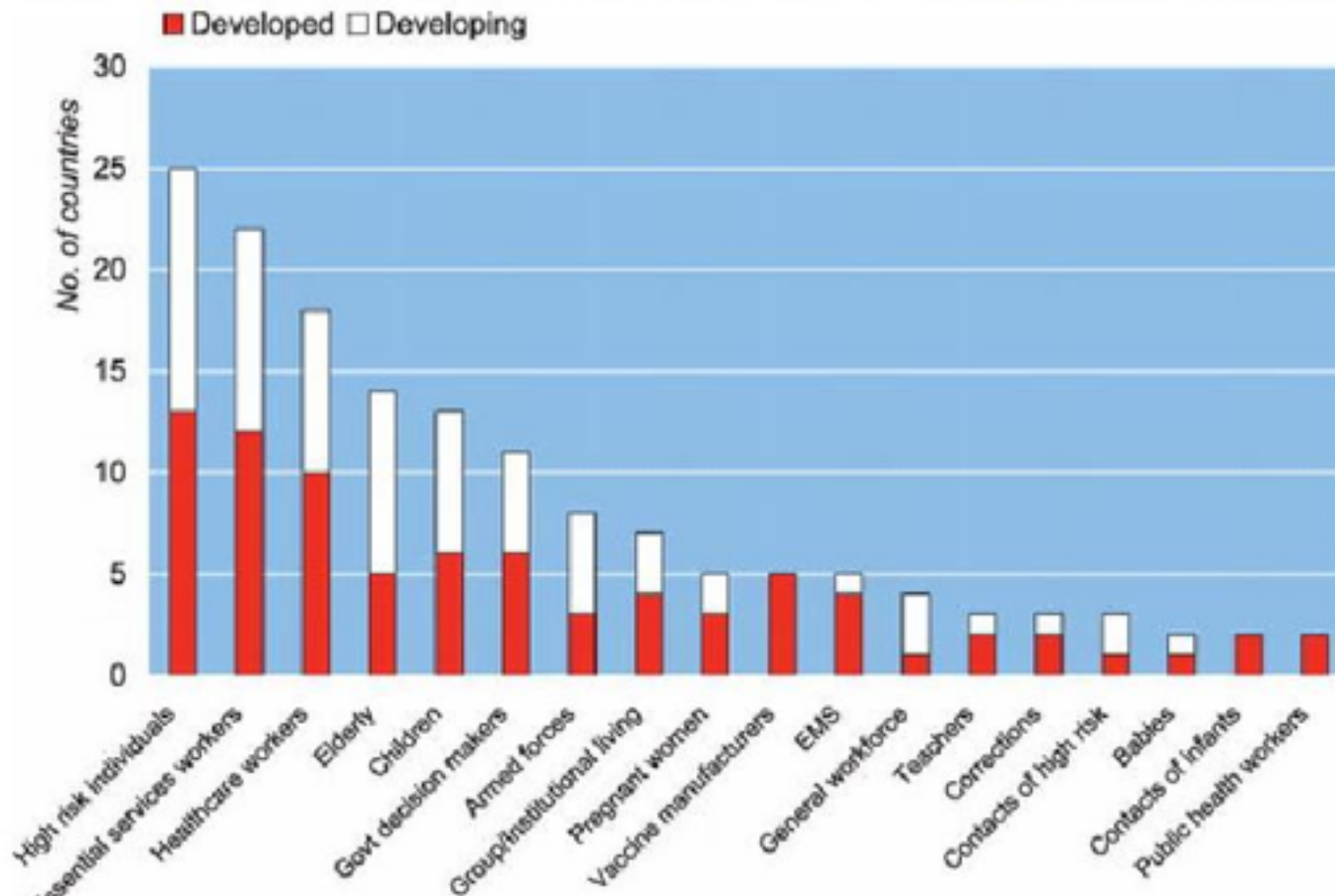
Cost

Benefit/Cost
ratio

358 !

5. Select designs with the best impacts in relation to their costs, do them first.

Figure 1: Vaccine Priority Groups by Development Status - Listed in at Least Two National Plans



Source: Uscher-Pines et al. Priority setting for pandemic influenza: An analysis of national preparedness plans. *PLoS ONE* 2013;8(12):e82115. DOI:10.1371/journal.pone.0082115

June 9, 2014

Impact Estimation: Value Decision Table



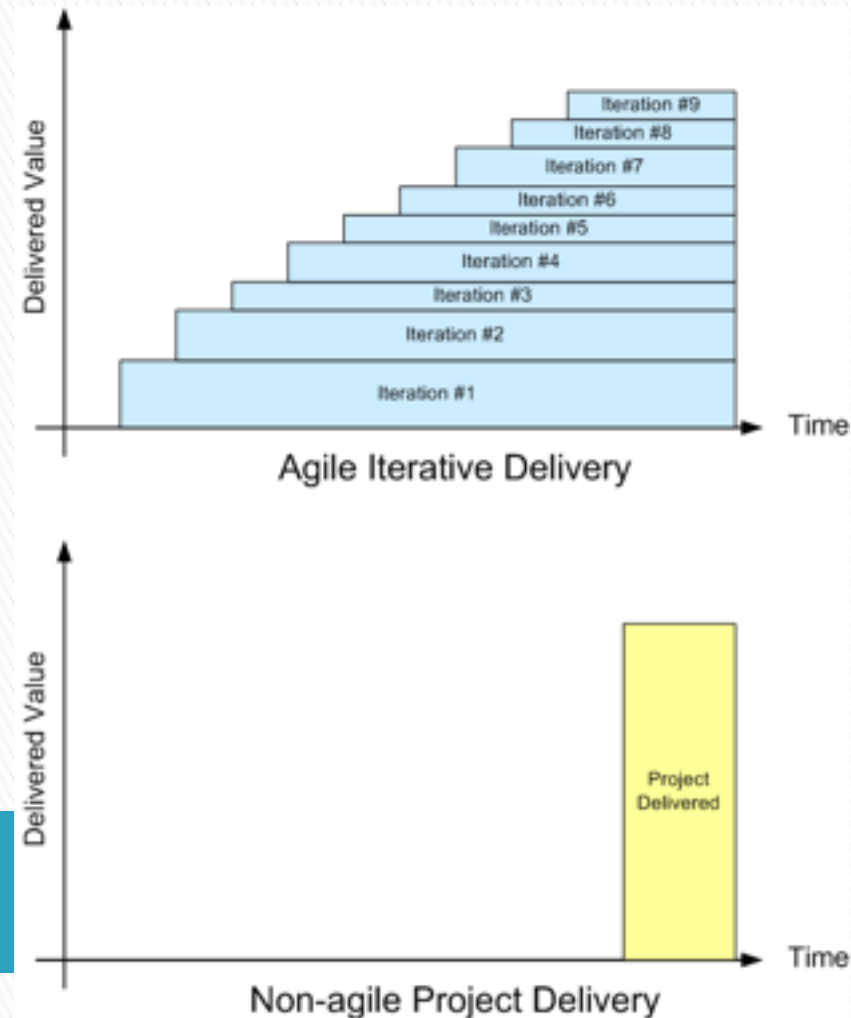
Decomposes Architecture by Value, and Value/Cost “Efficiency”

STRATEGIES → OBJECTIVES	Technology Investment	Business Practices	People	Empow- erment	Principles of IMA Management	Business Process Re- engineering	SUM
Customer Service ? → 0 Violation of agreement	50%	10%	5%	5%	5%	60%	185%
Availability 90% → 99.5% Up time	50%	5%	5-10%	0	0	200%	265%
Usability 200 → 60 Requests by Users	50%	5-10%	5-10%	50%	0	10%	130%
Responsiveness 70% → ECP's on time	50%	10%	90%	25%	5%	50%	180%
Productivity 3:1 Return on Investment	45%	60%	10%	35%	100%	53%	303%
Morale 72 → 60 per mo. Sick Leave	50%	5%	75%	45%	15%	61%	251%
Data Integrity 88% → 97% Data Error %	42%	10%	25%	5%	70%	25%	177%
Technology Adaptability 75% Adapt Technology	5%	30%	5%	60%	0	60%	160%
Requirement Adaptability ? → 2.6% Adapt to Change	80%	20%	60%	75%	20%	5%	260%
Resource Adaptability 2.1M → ? Resource Change	10%	80%	5%	50%	50%	75%	270%
Cost Reduction FADS → 30% Total Funding	50%	40%	10%	40%	50%	50%	240%
SUM IMPACT FOR EACH SOLUTION	482%	280%	305%	390%	315%	649%	
Money % of total budget	15%	4%	3%	4%	6%	4%	
Time % total work months/year	15%	15%	20%	10%	20%	18%	
SUM RESOURCES	30	19	23	14	26	22	
BENEFIT/RESOURCES RATIO	16:1	14:7	13:3	27:9	12:1	29.5:1	

Principle 6. Decompose the workflow, into weekly (or 2% of budget) time boxes

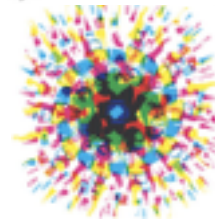
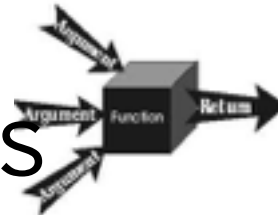
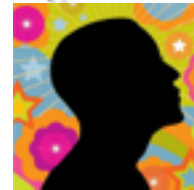
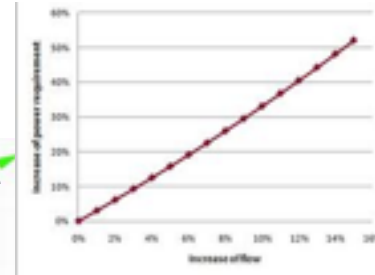
Decomposition of Projects:
How to Design Small
Incremental Steps INCOSE
2008

http://www.gilb.com/tiki-download_file.php?fileId=41



1 1 1 1 1 1 Unity

- 1% increase at least
- 1 stakeholder
- 1 quality or value
- 1-week delivery cycle
- 1 function focus
- 1 design used



http://www.gilb.com/tiki-download_file.php?fileId=451

111111 Unity Method slides

© Gilb.com 9 June 2014

some principles to apply:

Decomposition Principles A Teachable Discipline

Decomposition of Projects into small steps 11/12/2008 13:38

Decomposition of Projects: How to design small, early and frequent incremental and evolutionary feedback, stakeholder result delivery steps, at the level of 2% of project resources.
By Tom Gilb, Norway

Introduction

- The basic premise of iterative, incremental and evolutionary project management [Larman 03 MG] is that a project is divided into early, frequent and short duration delivery steps.
- One basic premise of these methods is that each step will attempt to deliver some real value to stakeholders.
- It is not difficult to envisage steps of construction for a system; the difficulty is when a step has to deliver something of value to stakeholders, in particular to end users.
- This paper will give some teachable guidelines, policies and principles for decomposition. It will also give short examples from practical experience.

A Policy for Evo Planning

One way of guiding Evo planners is by means of a 'policy'. A general policy looks like this (you can modify the policy parameters to your local needs):

Evo Planning Policy (example)

P1: Steps will be sequenced on the basis of their overall benefit-to-cost efficiency.

P2: No step may normally exceed 2% of total project financial budget.

1. *Believe* there is a way to do it, you just have not *found* it yet!
 2. *Identify* obstacles, but don't use them as excuses: use your imagination to get *rid* of them!
 3. Focus on *some usefulness* for the user or customer, however small.
 4. Do **not** focus on the design ideas themselves, they are distracting, especially for small initial cycles. Sometimes you have to ignore them entirely in the short term!
 5. Think; one customer, tomorrow, one interesting improvement.
 6. Focus on the *results* (which you should have defined in your goals, moving toward target levels).
 7. Don't be afraid to use temporary-scaffolding designs. Their cost must be seen in the light of the value of making some progress, and getting practical experience.
 8. Don't be worried that your design is inelegant; it is results that count, not style.
 9. Don't be afraid that the customer won't like it. *If* you are focusing on results *they want*, then by definition, *they* should like it. If you are not, then *do*!
 10. Don't get so worried about "what might happen afterwards" that you stop practical progress.
 11. You cannot foresee everything. Don't even *think* about it!
 12. If you focus on helping your customer in practice, *now*, where the customer is, you will be forgiven a lot of 'sins'!
 13. You can understand things much better, by getting *some* practical experience (removing *some* of your fears).
 14. Do *early* cycles, on willing local mature parts of your user community.
 15. When some cycles, like a purchase-order cycle, take a long time, initiate them early, and do other useful cycles while you wait.
 16. If something seems to need to wait for 'the big new system', ask if you cannot usefully do it with the 'awful old system', so as to pilot it realistically, and perhaps alleviate some 'pain' in the old system.
 17. If something seems too costly to buy, for limited initial use, see if you can negotiate some kind of 'pay as you really use' contract. Most suppliers would like to do this to get your patronage, and to avoid competitors making the same deal.
 18. If *you* can't think of some useful small cycles, then talk directly with the real 'customer' or end user. They probably have dozens of suggestions.
 19. Talk with end users in *any* case, they have insights you need.
 20. Don't be afraid to use the old system and the old 'culture' as a launching platform for the radical new system. There is a lot of merit in this, and many people overlook it.
- I have never seen an exception in 33 years of doing this with many varied cultures. Oh Ye of little faith!



Rene Descartes on Focus

- ▶ **“We should bring the whole force of our minds**
 - **to bear upon the most minute and simple details**
 - **and to dwell upon them for a long time**
 - **so that we become accustomed to perceive the truth clearly and distinctly.”**
- ▶ **Rene Descartes, Rules for the Direction of the Mind, 1628**

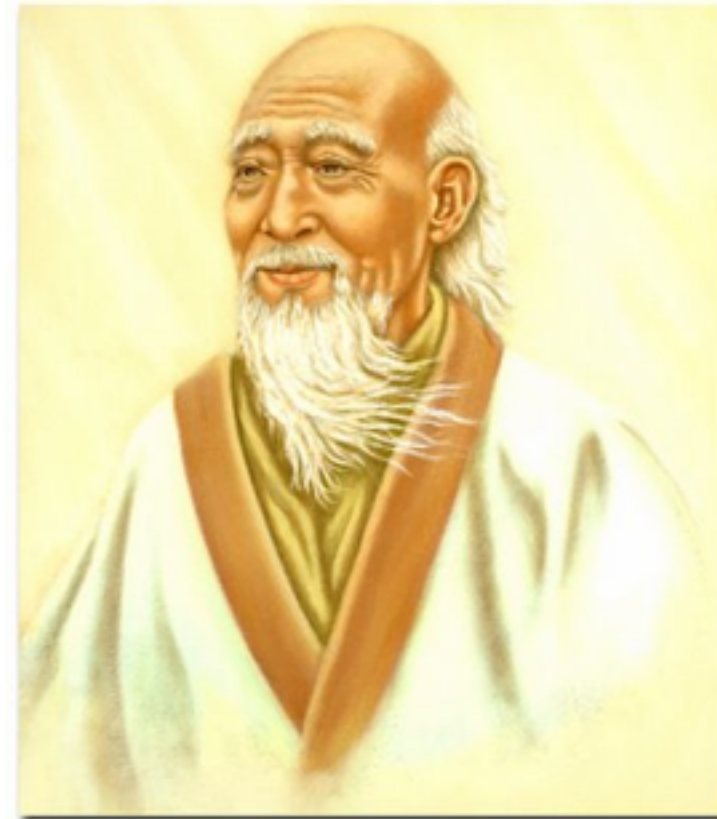




Tao Te Ching (500BC)



- ▶ **That which remains quiet, is easy to handle.**
- ▶ **That which is not yet developed is easy to manage.**
- ▶ **That which is weak is easy to control.**
- ▶ **That which is still small is easy to direct.**
- ▶ **Deal with little troubles before they become big.**
- ▶ **Attend to little problems before they get out of hand.**
 - **For the largest tree was once a sprout,**
- ▶ **the tallest tower started with the first brick,**
- ▶ **and the longest journey started with the first step.**



Principle 7.

Change **designs,**

based on

quantified experience of implementation

**Design is the
servant of the
requirement.
If it does not
work
'fire' it.**



Lean Startup: High Unknown



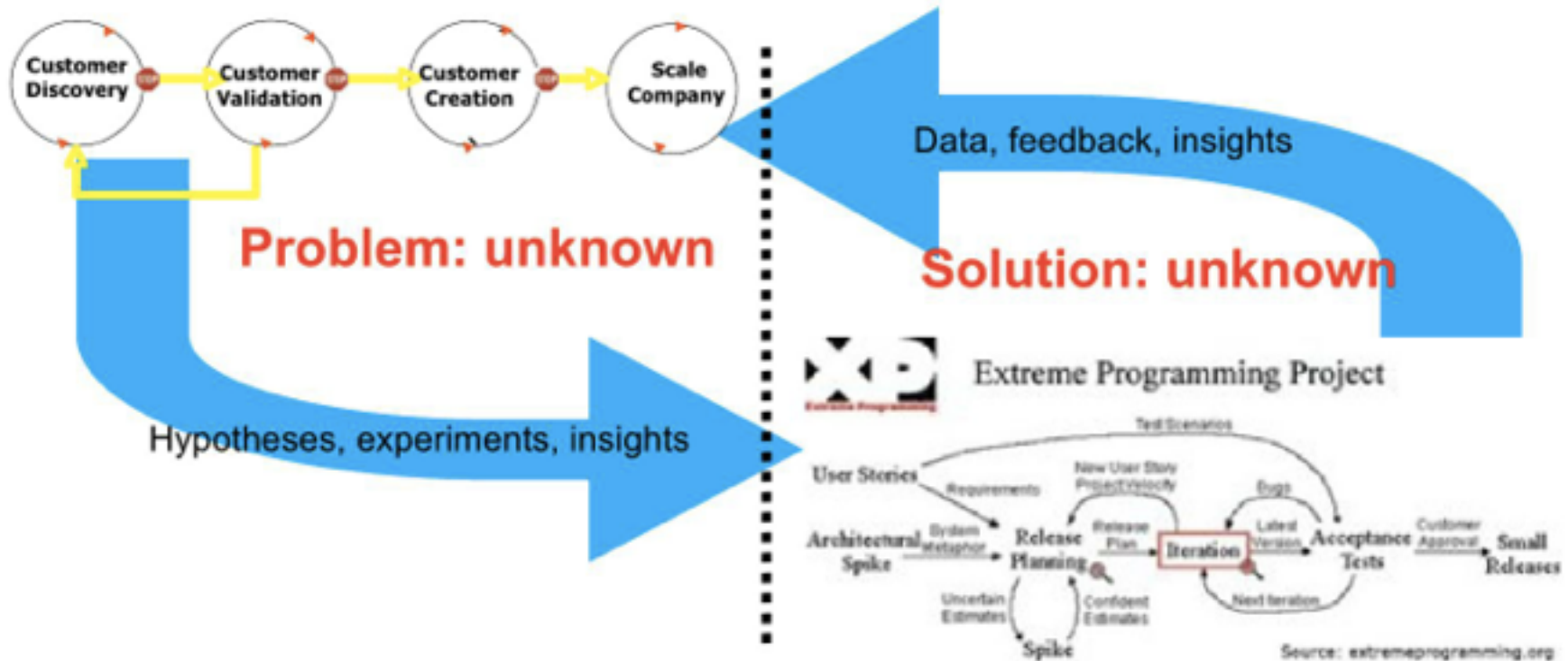
THE LEAN
STARTUP



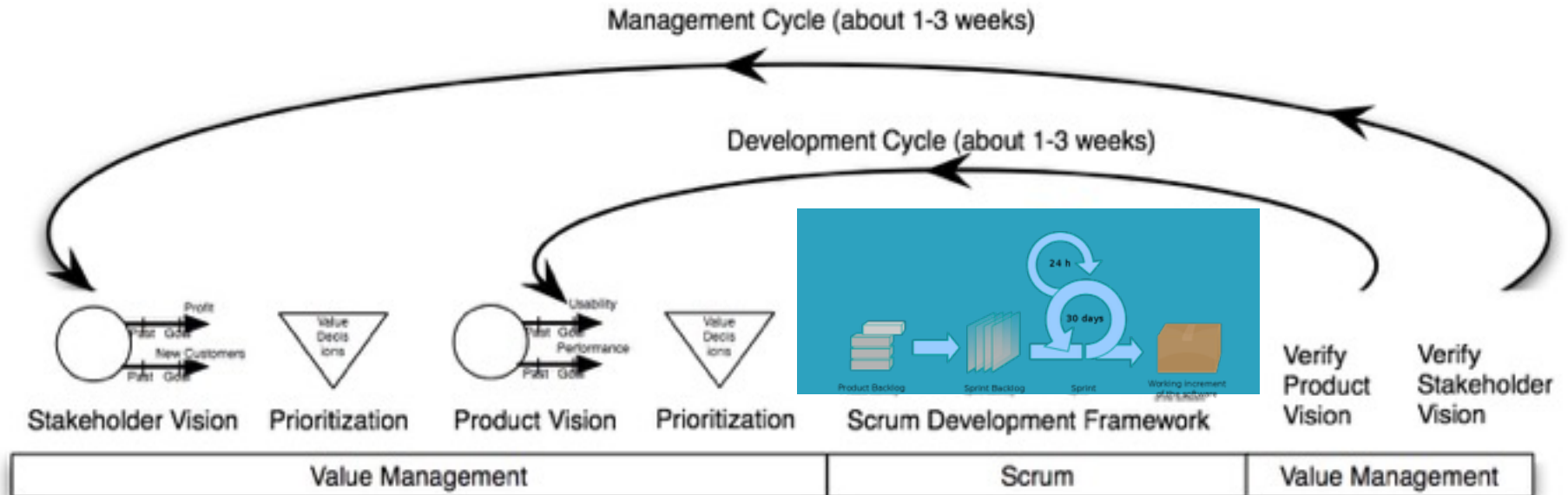
ERIC RIES

Product Development at Lean Startup
Assumes Customers and Markets are Unknown

Customer Development Engineering



Value Management (Gilb, Evo)





20 Sept, 2011 Report on Gilb Evo method (Richard Smith, Citigroup)



- ▶ <http://rsbatechnology.co.uk/blog:8>
- ▶ Back in 2004, I was employed by a large investment bank in their FX e-commerce IT department as a business analyst.
- ▶ The wider IT organisation used a complex waterfall-based project methodology that required use of an intranet application to manage and report progress.
- ▶ However, it's main failings were that it almost **totally missed the ability to track delivery of actual value improvements to a project's stakeholders, and the ability to react to changes in requirements and priority for the project's duration.**
- ▶ The toolset generated lots of charts and stats that provided **the illusion of risk control.** but actually provided very little help to the analysts, developers and testers actually doing the work at the coal face.
- ▶ The proof is in the pudding;
 - I have **used Evo** (albeit in disguise sometimes) on two large, high-risk projects in front-office investment banking businesses, and several smaller tasks.
 - On the largest critical project, the original business functions & performance objective **requirements document, which included no design, essentially remained unchanged** over the 14 months the project took to deliver,
 - but the detailed **designs** (of the GUI, business logic, performance characteristics) **changed many many times**, guided by lessons learnt and **feedback** gained by delivering a succession of early deliveries to real users.
 - In the end, the new system responsible for 10s of USD billions of notional risk, **successfully went live over one weekend for 800 users worldwide**, and was **seen as a big success by the sponsoring stakeholders.**

“I attended a 3-day course with you and Kai whilst at Citigroup in 200



Dynamic (Agile, Evo) design testing: not unlike 'Lean Startup'



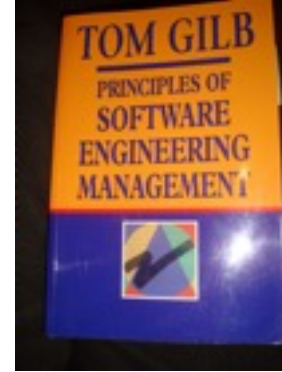
Richard Smith

- ▶ "... but **the detailed designs**
 - (of the GUI, business logic, performance characteristics)
- ▶ **changed many many times,**
- ▶ guided by lessons learnt
- ▶ and **feedback** gained by
- ▶ delivering a succession of early deliveries
- ▶ to real users"

"I attended a 3-day course with you and Kai whilst at Citigroup in 2006",
Richard Smith

Quinnan: IBM FSD Cleanroom

Dynamic Design to Cost



Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

'Cost management. . . yields valid cost plans linked to technical performance. Our practice carries cost management farther by introducing design-to-cost guidance. Design, development, and managerial practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method [illustrated in this book by Figure 7.10] consists of developing a design, estimating its cost, and ensuring that the design is cost-effective.' (p. 473)

-
He goes on to describe a design iteration process trying to meet cost targets by either 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.' (p. 474)

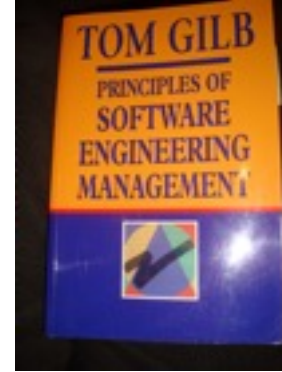
It is clear from this that they avoid the big bang cost estimation approach. Not only do they iterate in seeking the appropriate balance between cost and design for a single increment, but they iterate through a series of increments, thus reducing the complexity of the task, and increasing the probability 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.' (p. 474)

Source: Robert E. Quinnan, 'Software Engineering Management Practices', IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 466-77

This text is cut from Gilb: The Principles of Software Engineering Management, 1988

Quinnan: IBM FSD Cleanroom *Dynamic Design to Cost*



Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

'Cost management is carried farther by introducing an integrated way to estimate cost, as illustrated in this diagram. The method is cost-effective.' (p. 474)

He goes on to say that 'sacrificing planned development of each increment for the sake of cost is not a good idea.'

'Design is an iterative process.'

It is not only the appropriate balance between cost and design for a single increment, 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.' (p. 474)

Source: Robert E. Quinnan, 'Software Engineering Management Practices', IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 466-77

This text is cut from Gilb: The Principles of Software Engineering Management, 1988

**of developing a design,
estimating its cost, and
ensuring that the design
is cost-effective**

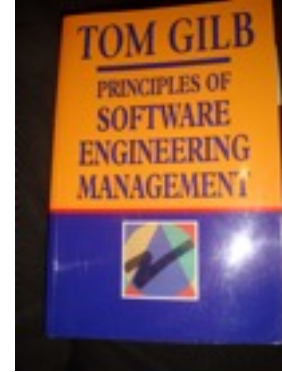
carries cost management practices are applied in an incremental manner. The method ensures that the design is cost-effective.

either redesign or by a single increment, the process is iterative.

' (p. 474)

only do they iterate in seeking the appropriate balance between cost and design for a single increment, 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.

Quinnan: IBM FSD Cleanroom *Dynamic Design to Cost*



Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

'Cost management. . . yields valid cost plans linked to technical performance. Our practice carries cost management farther by introducing design-to-cost guidance. Design, development, and managerial practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method [illustrated in this book by Figure 7.10] consists of developing a design, estimating its cost, and ensuring that the design is cost-effective.' (p. 473)

He goes on to describe a design iteration process trying to meet cost targets by either 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 iteration

It is only when the appropriate balance has been struck, thus reducing the cost of each increment development

'When the development cost has been computed.' (p. 474)

Source: Robert E. Quinn, 'The Cleanroom Approach to Software Development', IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 466-77

**iteration process
trying to meet cost
targets by either
redesign or by
sacrificing 'planned
capability'**

' (p. 474)

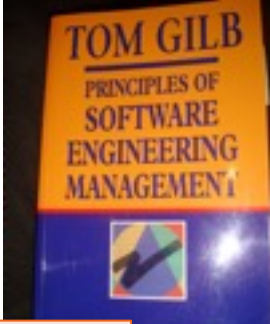
only do they iterate in seeking through a series of increments, experience, won as each

remaining increments is

. 19, No. 4, 1980, pp. 466-77

Quinnan: IBM FSD Cleanroom

Dynamic Design to Cost

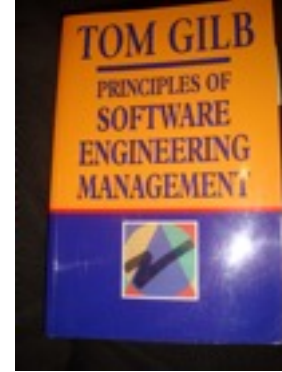


**Design is an
iterative process**

Quinnan: IBM FSD Cleanroom

Dynamic Design to Cost

Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.



~~'Cost management yields valid cost plans linked to technical performance. Our practice carries cost management~~

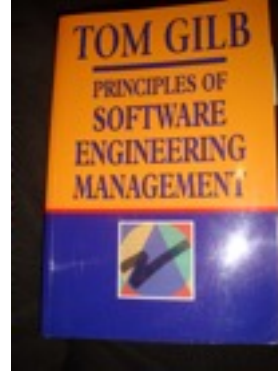
**but they iterate through a series of
increments,
thus *reducing the complexity of the
task,*
and *increasing the probability of
learning from experience***

Quinnan: IBM FSD Cleanroom *Dynamic Design to Cost*

Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

~~'Cost management yields valid cost plans linked to technical performance. Our practice carries cost management~~

**an estimate to
complete the remaining
increments is
computed.**



Principle 8. Change **requirements**, based on quantified experience, new inputs: intelligent tradeoff.

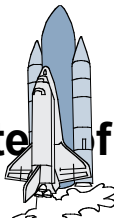
Reduce the level or delivery time, of lower-priority requirements, in order to deliver high priority requirements on time, within budget, or at Goal levels.



REAL EXAMPLE: Cleanroom Method, developed by IBM's Harlan Mills (1970-80) EARLY AGILE !!!



- ▶ **“Software Engineering began to emerge in FSD” (IBM Federal Systems Division, from 1996 a part of Lockheed Martin Marietta) “some ten years ago [Ed. about 1970] in a continuing evolution that is still underway:**
- ▶ **Ten years ago general management expected the worst from software projects – cost overruns, late deliveries, unreliable and incomplete software**
- ▶ **Today [Ed. 1980!], management has learned to expect on-time, within budget, deliveries of high-quality software.** A Navy helicopter ship system, called LAMPS, provides a recent example. LAMPS software was a four-year project of over 200 person-years of effort, developing over three million, and integrating over seven million words of program and data for eight different processors distributed between a helicopter and a ship in 45 incremental deliveries [Ed. Note 2%!]. Every one of those deliveries was on time and under budget
- ▶ **A more extended example can be found in the NASA space program,**
- ▶ **- Where in the past ten years, FSD has managed some 7,000 person-years of software development, developing and integrating over a hundred million bytes of program and data for ground and space processors in over a dozen projects.**
- ▶ **- There were few late or overrun deliveries in that decade, and none at all in the past four years.”**



In the Cleanroom Method, developed by IBM's Harlan Mills (1980) they report note: real Agile large scale from 1970-80!



- ▶ “Software Engineering began to emerge in FSD” (IBM Federal Systems Division, from 1996 a part of Lockheed Martin Marietta) “some ten years ago [Ed. about 1970] in a

in 45 incremental deliveries

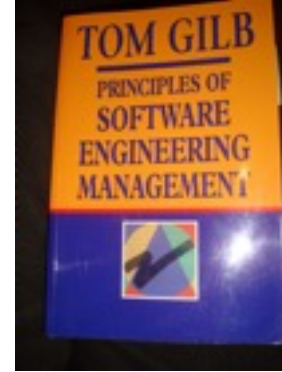
- ▶ Today [Ed. 1980!], management has learned to expect on-time, within budget, deliveries of high-quality software. A Navy helicopter ship system, called LAMPS, provides a recent example. LAMPS software was a four-year project of over 200 person-years of effort, developing over three million, and integrating over seven million words of program and data for eight different processors distributed between a helicopter and a ship in 45 incremental deliveries on time and under

- ▶ A more recent example of software program and data for eight different processors distributed between a helicopter and a ship in 45 incremental deliveries on time and under
- ▶ - When the software program and data for eight different processors distributed between a helicopter and a ship in 45 incremental deliveries on time and under
- ▶ - There were few late or overrun deliveries in that decade, and none at all in the past four years

were few late or overrun deliveries in that decade, and none at all in the past four years

Quinnan: IBM FSD Cleanroom

Dynamic Design to Cost



Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

'Cost management. . . yields valid cost plans linked to technical performance. Our practice carries cost management farther by introducing design-to-cost guidance. Design, development, and managerial practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method [illustrated in this book by Figure 7.10] consists of developing a design, estimating its cost, and ensuring that the design is cost-effective.' (p. 473)

He goes on to describe a design iteration process trying to meet cost targets by either 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.' (p. 474)

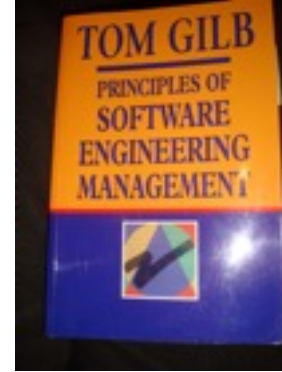
It is clear from this that they avoid the big bang cost estimation approach. Not only do they iterate in seeking the appropriate balance between cost and design for a single increment, but they iterate through a series of increments, thus reducing the complexity of the task, and increasing the probability 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.' (p. 474)

Source: Robert E. Quinnan, 'Software Engineering Management Practices', IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 466~77
This text is cut from Gilb: The Principles of Software Engineering Management, 1988

Quinnan: IBM FSD Cleanroom

Dynamic Design to Cost



Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

'Cost management is carried farther by introducing an integrated way to estimate cost, as illustrated in this diagram. The method is cost-effective.' (p. 474)

He goes on to say that 'sacrificing planned development of each increment for the sake of cost is not a good idea.'

'Design is an iterative process.'

It is not only the appropriate balance between cost and design for a single increment, 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.' (p. 474)

Source: Robert E. Quinnan, 'Software Engineering Management Practices', IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 466-77

This text is cut from Gilb: The Principles of Software Engineering Management, 1988

**of developing a design,
estimating its cost, and
ensuring that the design
is cost-effective**

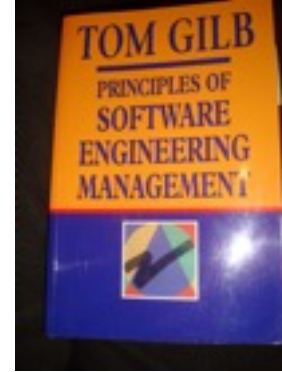
carries cost management practices are applied in an incremental manner. The method ensures that the design is cost-effective.

either redesign or by a single increment, the process is iterative.

' (p. 474)

only do they iterate in seeking the appropriate balance between cost and design for a single increment, 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.

Quinnan: IBM FSD Cleanroom *Dynamic Design to Cost*



Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

'Cost management. . . yields valid cost plans linked to technical performance. Our practice carries cost management farther by introducing design-to-cost guidance. Design, development, and managerial practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method [illustrated in this book by Figure 7.10] consists of developing a design, estimating its cost, and ensuring that the design is cost-effective.' (p. 473)

He goes on to describe a design iteration process trying to meet cost targets by either 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 iteration

It is only when the appropriate balance has been struck, thus reducing the cost of each increment development

'When the development cost has been computed.' (p. 474)

Source: Robert E. Quinn
This text is cut from

**iteration process
trying to meet cost
targets by either
redesign or by
sacrificing 'planned
capability'**

' (p. 474)

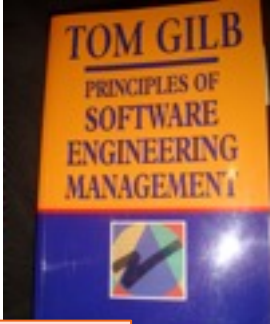
only do they iterate in seeking
through a series of increments,
experience, won as each

remaining increments is

. 19, No. 4, 1980, pp. 466-77

Quinnan: IBM FSD Cleanroom

Dynamic Design to Cost

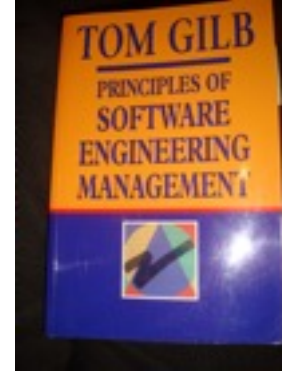


**Design is an
iterative process**

Quinnan: IBM FSD Cleanroom

Dynamic Design to Cost

Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.



~~Cost management yields valid cost plans linked to technical performance. Our practice carries cost management~~

**but they iterate through a series of
increments,
thus *reducing the complexity of the
task,*
and *increasing the probability of
learning from experience***

Quinnan: IBM FSD Cleanroom *Dynamic Design to Cost*

Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

'Cost management. . . yields valid cost plans linked to technical performance. Our practice carries cost management farther by introducing design-to-cost guidance. Design, development, and managerial practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method [illustrated in this book by Figure 7.10] consists of developing a design, estimating its cost, and ensuring that the design is co

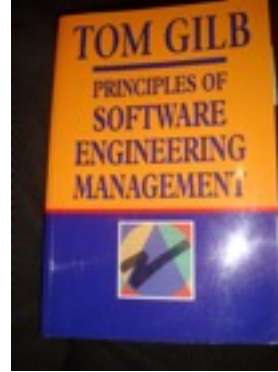
**an estimate to
complete the
remaining increments
is computed.**

'sacr
'dev

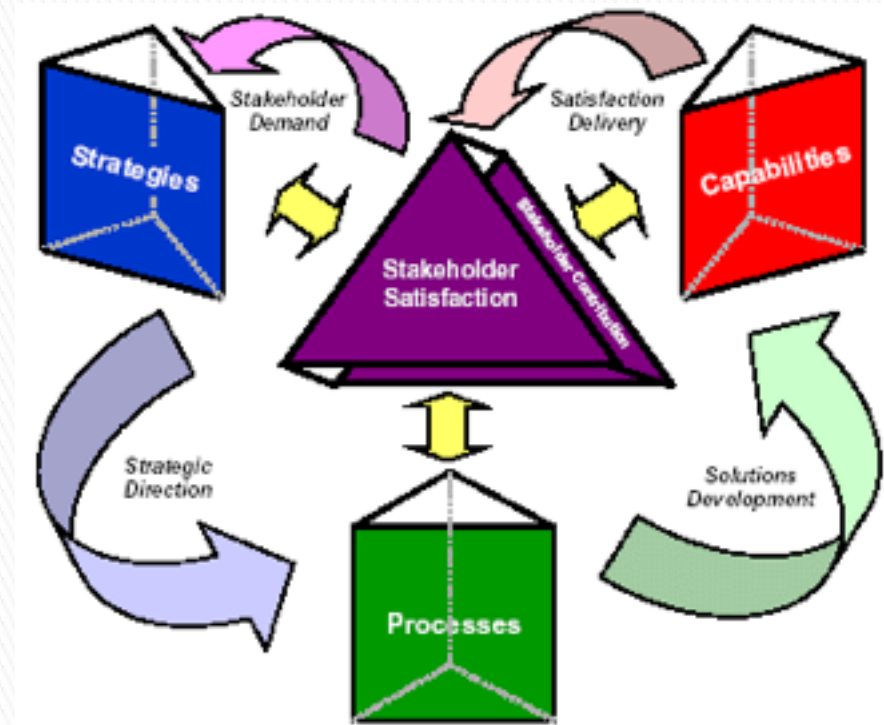
'Des

the a
thus
incre

'When
com
Sour
This



Principle 9. Involve the stakeholders, every week, in setting quantified goals



It is much easier to determine requirements with a little hindsight!

The eternal cycle of stakeholder priorities

Concurrent Quantified 'Empowered Creativity' *
The Software Engineers can use ANY design that they
believe delivers the planned value.
And keep what really works



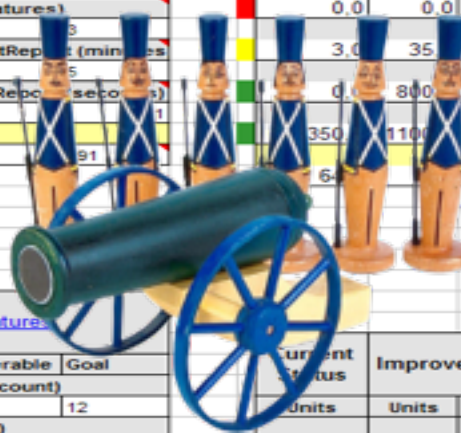
* Empowered Creativity: Term coined by Trond Johansen, Confirmit, 2003

EVO Plan Confrimit 8.5 in Evo Step Impact Measurement

4 product areas were attacked in all: **25 Qualities** concurrently, one quarter of a year. Total development staff = 13

Impact Estimation Table: Reportal codename "Hyggen"

Current Status			Improvements			Reportal - E-SAT features			Current Status			Improvements			Survey Engine .NET		
Units	Units	%	Past	Tolerable	Goal	Units	Units	%	Past	Tolerable	Goal	Units	Units	%	Past	Tolerable	Goal
75.0	25.0	62.5	50	75	90	83.0	48.0	80.0	40	85	95	0.0	67.0	100.0	67	85	0
14.0	14.0	100.0	0	11	14	4.0	59.0	100.0	63	8	4	10.0	397.0	100.0	407	100	10
15.0	15.0	107.1	0	11	14	94.0	2290.0	103.9	2384	500	180	10.0	10.0	13.3	0	100	100
5.0	75.0	96.2	80	5	2	774.0	507.0	51.7	1281	600	300	5.0	3.0	60.0	2	5	7
5.0	45.0	95.7	50	5	1	0.0	0.0	0.0	0	7	?	0.0	0.0	0.0	0	?	?
3.0	2.0	66.7	1	3	4	3.0	35	97.2	38	3	2	0.0	800	100.0	800	0	0
1.0	22.0	95.7	7	1	0	0.0	0.0	0.0	150	500	1000	0.0	0.0	0.0	0	0	0
4.0	5.0	100.0	8	5	3	350	1100	146.7	0	0	0	0.0	0.0	0.0	0	0	0
1.0	12.0	150.0	13	13	5	64	64	64	0	0	0	0.0	0.0	0.0	0	0	0
1.0	14.0	100.0	15	15	1	0	0	0	0	0	0	0.0	0.0	0.0	0	0	0
203.0			0	91		0	0	0	0	0	0	0.0	0.0	0.0	0	0	0
Current Status			Improvements			Reportal - MR Features			Current Status			Improvements			XML Web Services		
Units	Units	%	Past	Tolerable	Goal	Units	Units	%	Past	Tolerable	Goal	Units	Units	%	Past	Tolerable	Goal
1.0	1.0	50.0	14	13	12	7.0	9.0	81.8	16	10	5	17.0	8.0	53.3	25	15	10
20.0	45.0	112.5	65	35	25	943.0	-186.0	#####	170	60	30	5.0	10.0	95.2	15	7.5	4.5
4.4	4.4	36.7	0	4	12	2.0			0			0			0		
101.0			0		86												



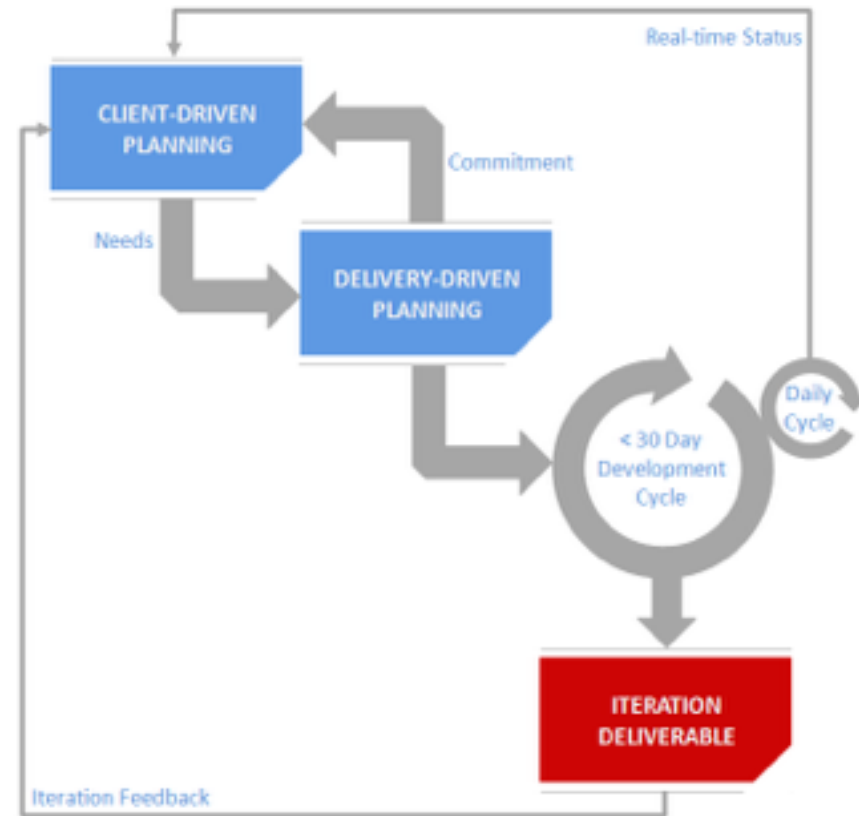
Evo's impact on Conformat product qualities 1st Qtr

- Only 5 highlights of the 25 impacts are listed here

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 Conformat 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



10. Involve the stakeholders, every week, in actually using increments



Quantified Value Delivery Project Management in a Nutshell

Quantified Value Requirements, Design, Design Value/cost estimation,
Measurement of Value Delivery, Incremental Project Progress to Date

	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

Estimates

Testing

Weekly

Priority

Next
week
Warning
metrics
based

Cumulative
weekly
progress
metric

Constraint

Target

EVO Plan Confrimit 8.5 in Evo Step Impact Measurement

4 product areas were attacked in all: **25 Qualities** concurrently, one quarter of a year. Total development staff = 13

Impact Estimation Table: Reportal codename "Hyggen"

Current Status			Improvements			Reportal - E-SAT features			Current Status			Improvements			Survey Engine .NET		
Units	Units	%	Past	Tolerable	Goal	Units	Units	%	Past	Tolerable	Goal	Units	Units	%	Past	Tolerable	Goal
75.0	25.0	62.5	50	75	90	83.0	48.0	80.0	40	85	95	0.0	67.0	100.0	67	85	0
14.0	14.0	100.0	0	11	14	4.0	59.0	100.0	63	8	4	10.0	397.0	100.0	407	100	10
15.0	15.0	107.1	0	11	14	94.0	2290.0	103.9	2384	500	180	10.0	10.0	13.3	0	100	100
5.0	75.0	96.2	80	5	2	774.0	507.0	51.7	1281	600	300	5.0	3.0	60.0	2	5	7
5.0	45.0	95.7	50	5	1	0.0	0.0	0.0	0	7	?	0.0	0.0	0.0	0	?	?
3.0	2.0	66.7	1	3	4	3.0	35	97.2	38	3	2	0.0	800	100.0	800	0	0
1.0	22.0	95.7	7	1	0	0.0	0.0	0.0	150	500	1000	0.0	0.0	0.0	0	0	0
4.0	5.0	100.0	8	5	3	350	1100	146.7	0	0	0	0.0	0.0	0.0	0	0	0
1.0	12.0	150.0	13	13	5	64	64	64	0	0	0	0.0	0.0	0.0	0	0	0
1.0	14.0	100.0	15	15	1	0	0	0	0	0	0	0.0	0.0	0.0	0	0	0
203.0			0	91		0	0	0	0	0	0	0.0	0.0	0.0	0	0	0
Current Status			Improvements			Reportal - MR Features			Current Status			Improvements			XML Web Services		
Units	Units	%	Past	Tolerable	Goal	Units	Units	%	Past	Tolerable	Goal	Units	Units	%	Past	Tolerable	Goal
1.0	1.0	50.0	14	13	12	7.0	9.0	81.8	16	10	5	17.0	8.0	53.3	25	15	10
20.0	45.0	112.5	65	35	25	943.0	-186.0	#####	170	60	30	5.0	10.0	95.2	15	7.5	4.5
4.4	4.4	36.7	0	4	12	2.0			0			0			0		
101.0			0		86												



Code quality – "green" week, 2005

"Refactoring by Proactive Design Engineering!"

- ▶ In these "green" weeks, some of the deliverables will be less visible for the end users, but more visible for our QA department.
- ▶ We manage code quality through an Impact Estimation table

Current Status		Improvement		Goals			Step 6 (week 14)		Step 7 (week 15)
	Units			Past	Tolerable	Goal	Estimated Impact	Actual Impact	Estimated Impact A
	100,0	100,0	0	80	100				100
Speed									
	100,0	100,0	0	80	100		100	100	
Maintainability.Doc.Code									
	100,0	100,0	0	80	100		100	100	
InterviewerConsole									
NUnitTests									
	0,0	0,0	0	90	100				
PeerTests									
	100,0	100,0	0	90	100				100
FxCop									
	0,0	10,0	10	0	0				
TestDirectorTests									
	100,0	100,0	0	90	100				100
Robustness.Correctness									
	2,0	2,0	0	1	2		2	2	
Robustness.BoundaryConditions									
	0,0	0,0	0	8					
Speed									
	0,0	0,0	0	8					
ResourceUsage.CPU									
	100,0	0,0	100	8					
Maintainability.Doc.Code									
	100,0	100,0	0	8					
SynchronizationStatus									
NUnitTests									

POT-SHOTS — Brilliant Thoughts in 17 words or less



© ASHLEIGH BRILLIANT 2005

© Ashleigh Brilliant

www.ashleighbrilliant.com

Speed

Maintainability

Nunit Tests

PeerTests

TestDirectorTests

Robustness.Correctness

Robustness.Boundary
Conditions

ResourceUsage.CPU

Maintainability.DocCode

Monday, 9 June 14

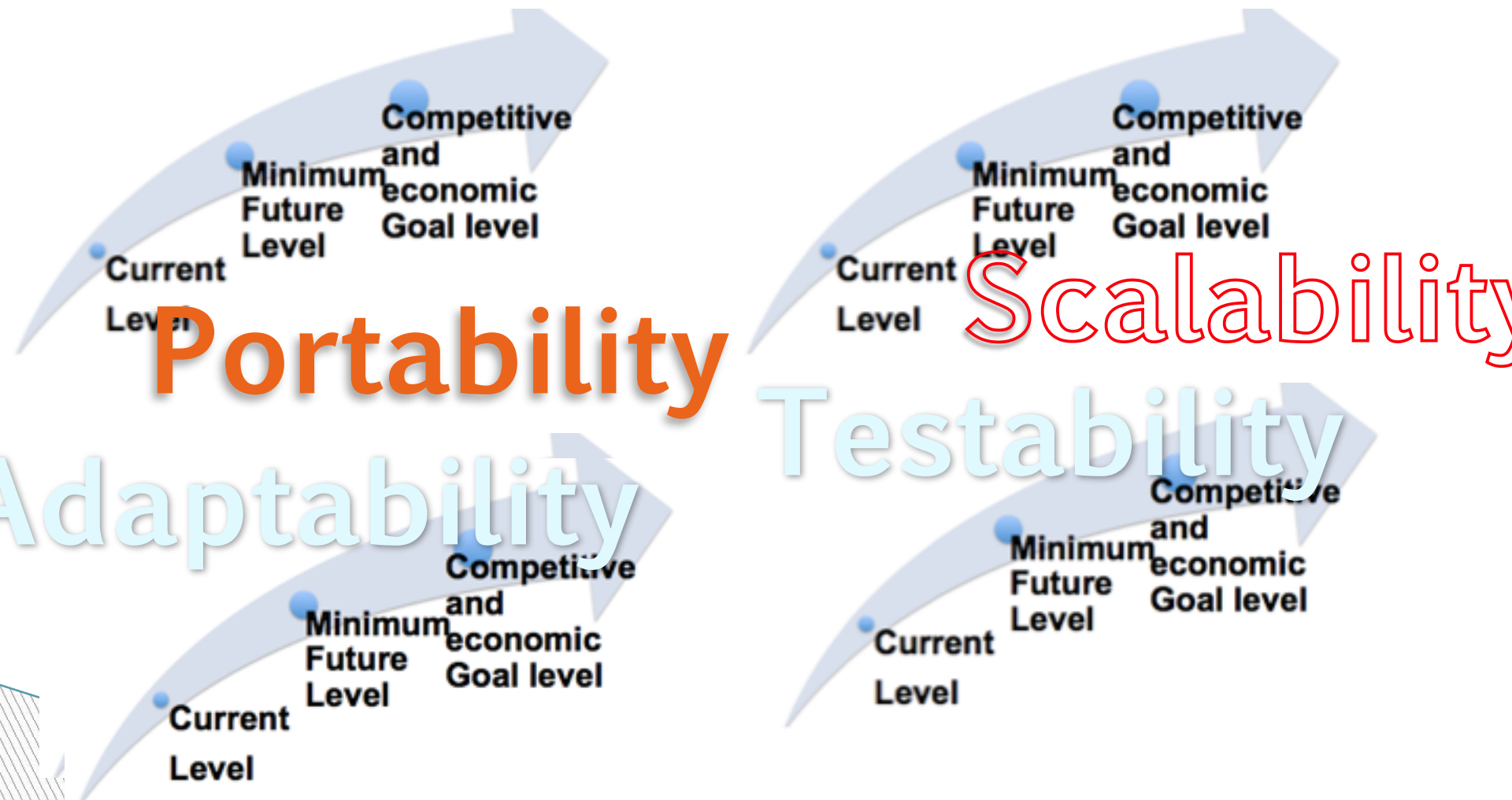
62

Raising the Levels of Maintainability like 'Mean Time To Fix a Bug'



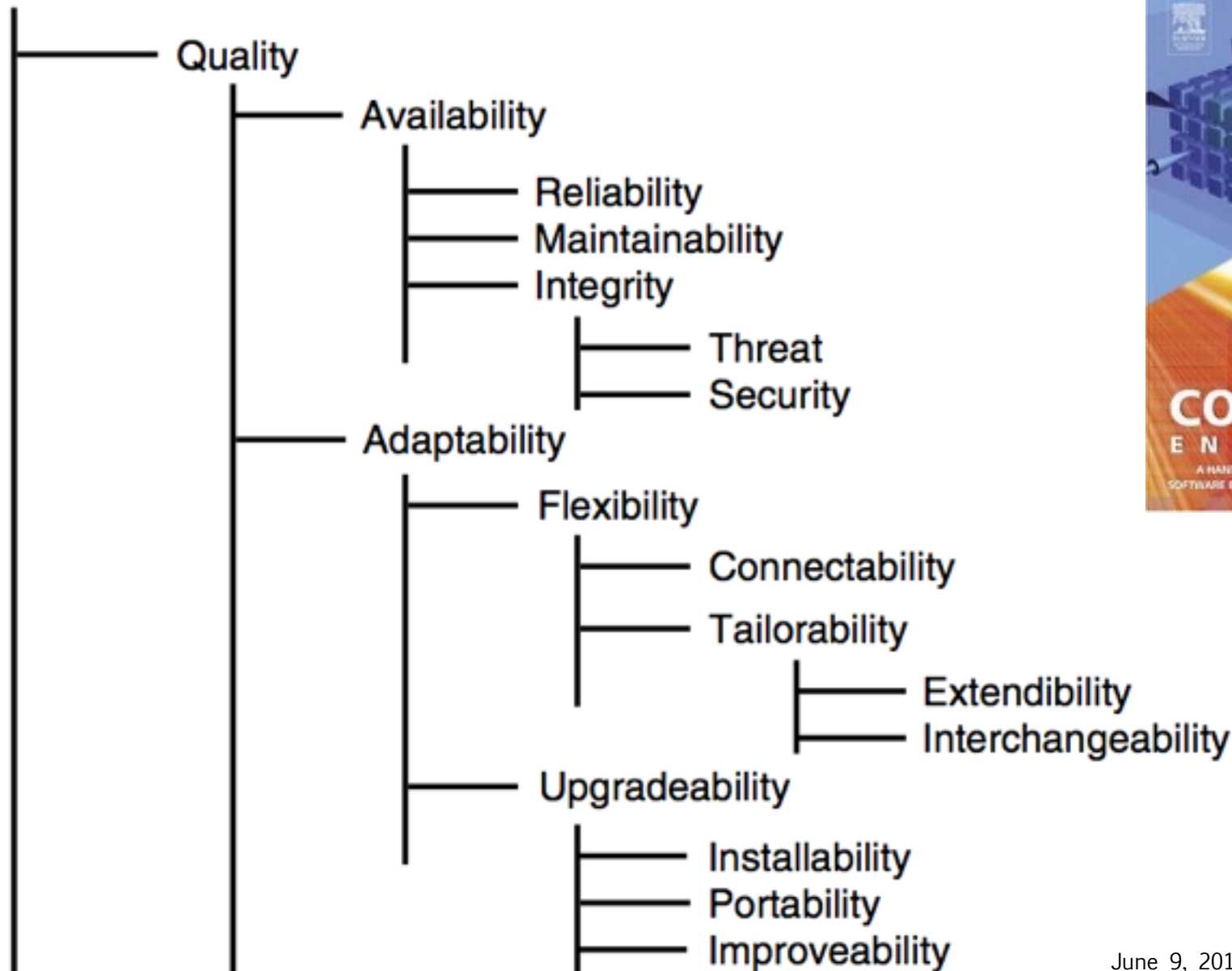
Raising the Levels of Maintainability

Multiple Attributes of Technical Debt



Broader 'Maintainability' Concepts

ALL *quantified*, with a defined Scale of measure in CE-5 Performance



1. The Conscious Design Principle:

- “Maintainability must be *consciously* designed into a system:
 - failure to **design** to a set of levels of maintainability
 - means the **resulting maintainability** is both *bad* and *random*. ”
- © Tom Gilb (2008, INCOSE Paper)
 - http://www.gilb.com/tiki-download_file.php?fileId=138



The 'Maintainability' Generic Breakdown into Sub-problems

1. Problem Recognition Time.

How can we reduce the time from bug actually occurs until it is recognized and reported?

2. Administrative Delay Time:

How can we reduce the time from bug reported, until someone begins action on it?

3. Tool Collection Time.

How can we reduce the time delay to collect correct, complete and updated information to analyze the bug: source code, changes, database access, reports, similar reports, test cases, test outputs.

4. Problem Analysis Time.

Etc. for all the following phases defined, and implied, in the Scale scope above.

5. Correction Hypothesis Time

6. Quality Control Time

7. Change Time

8. Local Test Time

9. Field Pilot Test Time

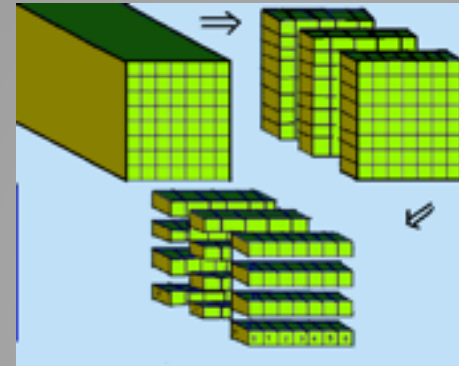
10. Change Distribution Time

11. Customer Installation Time

12. Customer Damage Analysis Time

13. Customer Level Recovery Time

14. Customer QC of Recovery Time



Source: Competitive Engineering Ch 5
& Ireson (ed.) Reliability Handbook, 1966

An Example of Specifying 1 Attribute

Restore Speed:

Type: Software Quality Requirement. Version: 25 October 2007.

Part of: Rock Solid Robustness

Ambition: Should an error occur (or the user otherwise desire to do so), the system 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.

Initiation: 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.

Let's Vote

1. How many of you would **prefer** to keep doing conventional 'softcrafter' refactoring; even if the results were not measurable

2. How many of you think you **ought** to try to engineer measurable software maintainability results into your systems

- Even if your boss is not smart enough to ask you, or support you doing it?

Further Reading: AgileRecord.com

Gillb's Mythodology Column

The Green Week: Reducing Technical Debt by Engineering

by Tom & Kai Gillb

Our client Confront.com has used our Evo Agile Method [2] successfully since 2003 [1]. They have adopted it, from the beginning, to their environment, and continued to innovate and learn. Their business success has been attributed to their remarkable product quality improvement, and that improvement specifically to the Evo Agile method, by them, on their website, and share offerings prospectus. Evo differs from other agile methods, in that it focuses on multiple, quantified, software-and-system qualities.

This column will focus on an innovation, the Green Week, that Confront, led by their method champion Trond Johansen, made and reported in 2005; two years after adopting Evo.

When we started in 2003, Confront had an 8 year old web-based system; a 'legacy' product that had grown, as most do, to meet rapidly emerging market demands. By 2005 there were the usual difficulties in enhancing the product, a web-based opinion survey tool, serving markets worldwide, to meet new opportunities, quickly and safely.

We recommended in 2003 that they spend 4 days a week on value delivery cycles to their customer base, and one day a week 'refactoring'. Their development team at the time was 13 plus 3 testers.

The 4-day value delivery cycle aimed at something like 25 distinct quality improvements (for example Usability, Intuitiveness) or performance capacity improvements. The stakeholders aimed at were users and Confront's future market. The refactoring was aimed at their development team, as stakeholders. The team that did the development initially, also did the maintenance of the system for years, until today.

Let me be explicit, the people who had to 'buffer' bug fixing and long term enhancement were actually in full control of the architecture and design of the entire system. Maintenance was not farmed out to people who just had to suffer it. Most of the staff were not merely programmers, they had formal education in real engineering.

Well, the one day of refactoring was not a great success, while the 4 days of value delivery cycles, to quantified quality and performance requirements was a big success. To my knowledge there is nothing even near-as-good of quantified results, reported for any other Agile Effort! If you know of one, AgileRecord (.com) would like to

hear from you! One possible reason for lack of success was that the refactoring was one-day-a-week, and I suspect it was a Friday, where Norwegians want to sneak off early for a Cabin Weekend ('working off site'). But I really don't know.

They asked themselves, 'why should our customers get all the quality improvements?' What not, us hard working developers, get some systematic quality improvements too?

So they decided to spend one week a month, using Evo [2] 'engineering' 'ease of maintenance' and 'testability' into their organization and their product. In other words: 3 weeks being customer oriented, and 1 week a month being internally oriented. Of course, improvements in maintenance capability also improve their ability to respond to customers!

User Week 1	User Week 2	User Week 3	Developer Week 4
Select a Goal	Select a Goal	Select a Goal	Select a Goal
Brainstorm Designs	Brainstorm Designs	Brainstorm Designs	Brainstorm Designs
Estimate Design Impact/Cost	Estimate Design Impact/Cost	Estimate Design Impact/Cost	Estimate Design Impact/Cost
Pick best design	Pick best design	Pick best design	Pick best design
Implement design	Implement design	Implement design	Implement design
Test design	Test design	Test design	Test design
Update Progress to Goal	Update Progress to Goal	Update Progress to Goal	Update Progress to Goal

Figure 1: The weekly development cycles, with the Green Week.

The key idea here is that we start by quantifying as requirements, all Confront system (the software product, the service product, the technical organization) attributes, related to ease of maintaining the system, in the widest sense of 'maintaining' [3].

Here are the requirements they quantified as requirements initially: Speed, Maintainability, Nunit Tests, Peer Tests, Test Director Tests, Robustness, Correctness, Robustness, Boundary Conditions, Resource Usage CPU, Maintainability DocCode, Synchronization Status.

Page 26  Agile Record - www.agilerecord.com

My 10 Agile Values?

- **Simplicity**
 - **1. Focus on real stakeholder values**
- **Communication**
 - **2. Communicate stakeholder values quantitatively**
 - **3. Estimate expected results and costs for weekly steps**
- **Feedback**
 - **4. Generate results, weekly, for stakeholders, in their environment**
 - **5. Measure all critical aspects of the improved results cycle.**
 - **6. Analyze deviation from your initial estimates**
- **Courage**
 - **7. Change plans to reflect weekly learning**
 - **8. Immediately implement valued stakeholder needs, next week**
 - **Don't wait, don't study (analysis paralysis), don't make excuses.**
 - **Just Do It!**
 - **9. Tell stakeholders exactly what you will deliver next week**
 - **10. Use any design, strategy, method, process that works quantitatively well - to get your results**
 - **Be a systems engineer, not a just programmer (a 'Softcrafter').**
 - **Do not be limited by your craft background, in serving your paymasters**



"Values for Value"

http://www.gilb.com/tiki-download_file.php?fileId=448

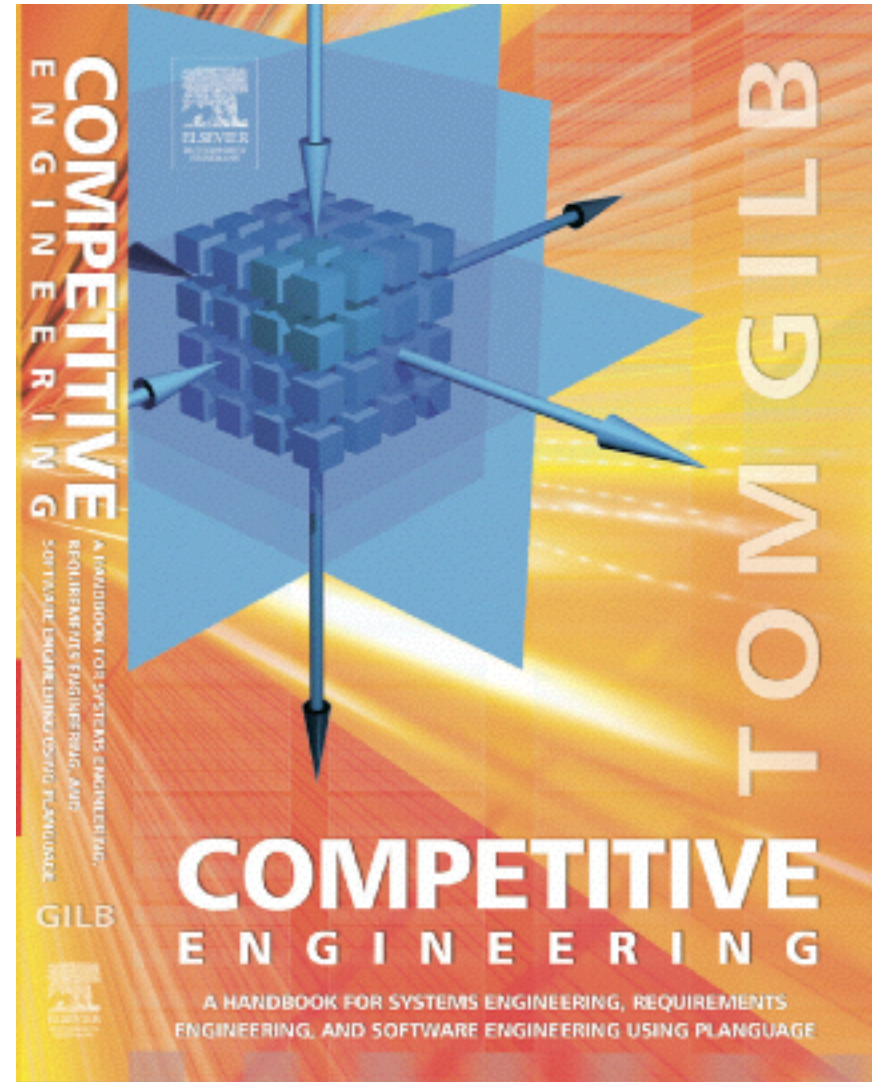
Agile Record 2010, www.agilerecord.com, October 2010, Issue 4

- Ecstatic Stakeholder!



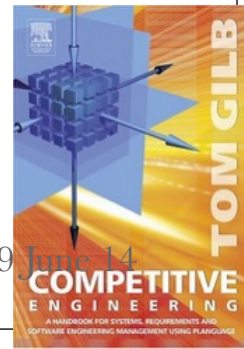
That's All Folks!

- Discussion?
 - I am here all Conference and incl My Friday 'Evo' Seminar
 - And love to talk with you!
- Remarks? Questions?
 - Email me if you like
- **For free digital copy of this book, and 4 of my Agile papers, and the Evo book**
- Email me subject **"Book"**
- **Tom@Gilb.com**
- If you want to agree a meeting, email me
- or text +47 92066705
-
- This talk is NOW at Gilb.com/Downloads (Slides)



Agile Credibility

- **Agile ‘Grandfather’ (Tom)**
 - **Practicing ‘Agile’ IT Projects since 1960**
 - **Preaching Agile since 1970’s (CW UK)**
 - **Acknowledged Pioneer by Agile Gurus and Research**
 - Beck, Sutherland, Highsmith, Cohn, Larman etc.
 - Ask me for details on this! I am too shy to show it here!
- **Agile Practice**
 - **IT: for decades (Kai and Tom)**
 - **Organisations: for Decades (Citigroup, Intel, HP, Boeing)**
- **Books:**
 - **Principles of Software Engineering Management (1988)**
the book Beck and others refer to
 - **Competitive Engineering (2005)**
 - **Evo: (Kai, evolving, 55 iterations)**





am not that shy! most influential!)



Agile References:

"Tom Gilb invented Evo, arguably the first Agile process. He and his son Kai have been working with me in Norway to align what they are doing with Scrum."

Kai has some excellent case studies where he has acted as Product Owner. He has done some of the most innovative things I have seen in the Scrum community."

Jeff Sutherland, co-inventor of Scrum, 5Feb 2010 in Scrum Alliance Email.

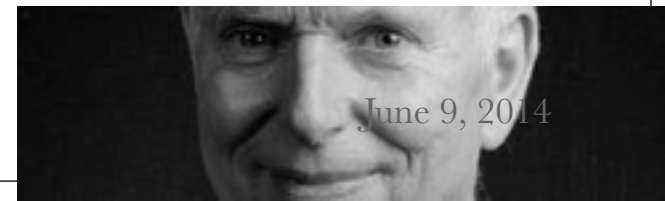
"Tom Gilb's Planguage referenced and praised at #scrumgathering by Jeff Sutherland. I highly agree" Mike Cohn, Tweet, Oct 19 2009

"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." Mike Cohn <http://blog.mountangoatsoftware.com/?p=77>

Comment of Kent Beck on Tom Gilb's book , "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)

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" (Cutter It Journal: The Journal of Information Technology Management, July 2004page 4, July 2004).



TWELVE TOUGH QUESTIONS

1. Why isn't the improvement quantified?
2. What is degree of the risk or uncertainty and why?
3. Are you sure? If not, why not?
4. Where did you get that from? How can I check it out?
5. How does your idea affect my goals, measurably?
6. Did we forget anything critical to survival?
7. How do you know it works that way? Did it before?
8. Have we got a complete solution? Are all objectives satisfied?
9. Are we planning to do the 'profitable things' first?
10. Who is responsible for failure or success?
11. How can we be sure the plan is working, during the project, early?
12. Is it 'no cure, no pay' in a contract? Why not?