‘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.slideshare.net/tomgilb1/agility-is-the-tool-gilb-vilnius-9-dec-2013

Agile Principles : Agile Record Paper as Published

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

- Learn
- Stakeholders
- Values
- Solutions
- Deliver
- Develop
- Decompose
- Measure
“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

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

Source: David Rico


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.)*

Value Driven Scrum

System Owner
• Stakeholders Values
• Business Values
• System Functions

Product Owner
• Build
• Test
• Maintain
• Detailed Technical Design
Value Decision Tables

<table>
<thead>
<tr>
<th>Business Goals</th>
<th>Training Costs</th>
<th>User Productivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profit</td>
<td>-10%</td>
<td>40%</td>
</tr>
<tr>
<td>Market Share</td>
<td>50%</td>
<td>10%</td>
</tr>
<tr>
<td>Resources</td>
<td>20%</td>
<td>10%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stakeholder Val.</th>
<th>Intuitiveness</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Costs</td>
<td>-10%</td>
<td>50%</td>
</tr>
<tr>
<td>User Productivity</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Resources</td>
<td>2%</td>
<td>5%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product Values</th>
<th>GUI Style Rex</th>
<th>Code Optimize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intuitiveness</td>
<td>-10%</td>
<td>40%</td>
</tr>
<tr>
<td>Performance</td>
<td>50%</td>
<td>80%</td>
</tr>
<tr>
<td>Resources</td>
<td>1%</td>
<td>2%</td>
</tr>
</tbody>
</table>

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

Scrum Develops
We measure improvements
Learn and Repeat

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

Copyright: Kai@Gilb.com

June 9, 2014
Jeffsutherland Twitter: Very cool product backlog management by Tom and Kai Gilb http://ad.vu/2h4d   Sat 28 March 2009
Copyright: Kai@Gilb.com
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

<table>
<thead>
<tr>
<th></th>
<th>Value to Create</th>
<th>Value to Preserve</th>
<th>Value to Sacrifice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employees</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suppliers and</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Professional Advisers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Investors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trades Unions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Government</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Media</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Community</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other Stakeholder</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Groups</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

**Value Decision Tables**

<table>
<thead>
<tr>
<th>Business Goals</th>
<th>Training Costs</th>
<th>User Productivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profit</td>
<td>-10%</td>
<td>40%</td>
</tr>
<tr>
<td>Market Share</td>
<td>30%</td>
<td>10%</td>
</tr>
<tr>
<td>Resources</td>
<td>20%</td>
<td>10%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stakeholder Val.</th>
<th>Intelligence</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Costs</td>
<td>-10%</td>
<td>50%</td>
</tr>
<tr>
<td>User Productivity</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Resources</td>
<td>2%</td>
<td>5%</td>
</tr>
</tbody>
</table>

**Product Values**

<table>
<thead>
<tr>
<th>Product Values</th>
<th>GUI Style</th>
<th>Code Optimise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usability</td>
<td>10%</td>
<td>40%</td>
</tr>
<tr>
<td>Performance</td>
<td>50%</td>
<td>80%</td>
</tr>
<tr>
<td>Resources</td>
<td>1%</td>
<td>3%</td>
</tr>
</tbody>
</table>

Scrum Develops - We measure improvements - Learn and Repeat

**End Results / Financial**

- Shareholder Value
- Revenue
- Productivity
- Customer Value
- Revenue Sources
- Cost Structure
- Use of Assets
- Customer Relationship
- Image
- Product Properties
- Operational Processes
- Work Climate
- Competences
- Technology

Learn & Growth

(Jeff Sutherland, Twitter: Very cool product backlog management by Tom and Kai Gilb http://twi.ve/2h4d Set 8 March 2012)

---

© Gilb.com  Agility is the Tool  June 9, 2014
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 Corporation's 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 total quality are without parallel.

This lack of clarity cost them $100,000,000.
## Business Objectives

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

Numbers are intentionally changed from real ones. 

More like this! (Real Example).

<- Business Objectives Quantified
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  

**Record [NARDAC]** 0 ?  

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

**Goal [This Year, PERSINCOM]** 0 “Go for the Record”  

Group SWAG
Principle 2.

Make sure those results are business results, not JUST technical

Align your project with your financial sponsor’s interests!
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:**

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

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

"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
Simple Product Owner (Ambler)

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

**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
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.
The 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.
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 (Confirmit Case, Norway)
Quantified Value Requirements, Design, Design Value/cost estimation, Measurement of Value Delivery, Incremental Project Progress to Date

<table>
<thead>
<tr>
<th>Cumulative weekly progress metric</th>
<th>Constraint Target</th>
<th>Weekly Testing</th>
<th>Estimates</th>
<th>Priority</th>
<th>Next week</th>
<th>Warning metrics based</th>
</tr>
</thead>
<tbody>
<tr>
<td>101.0</td>
<td>91.8</td>
<td>4.00</td>
<td>3.64</td>
<td>4.00</td>
<td>3.64</td>
<td></td>
</tr>
<tr>
<td>Development resources</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
REAL EXAMPLE: Strategy Impact Estimation:
for a $100,000,000 Organizational Improvement Investment

<table>
<thead>
<tr>
<th>Objectives</th>
<th>Technical Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"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.
## Impact Estimation: Value Decision Table

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Service</td>
<td>50%</td>
<td>10%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>60%</td>
<td>185%</td>
</tr>
<tr>
<td>Availability</td>
<td>50%</td>
<td>5%</td>
<td>5-10%</td>
<td>0</td>
<td>0</td>
<td>200%</td>
<td>265%</td>
</tr>
<tr>
<td>Usability</td>
<td>50%</td>
<td>5-10%</td>
<td>5-10%</td>
<td>50%</td>
<td>0</td>
<td>10%</td>
<td>130%</td>
</tr>
<tr>
<td>Responsiveness</td>
<td>50%</td>
<td>10%</td>
<td>90%</td>
<td>25%</td>
<td>5%</td>
<td>50%</td>
<td>180%</td>
</tr>
<tr>
<td>Productivity</td>
<td>45%</td>
<td>60%</td>
<td>10%</td>
<td>35%</td>
<td>100%</td>
<td>53%</td>
<td>303%</td>
</tr>
<tr>
<td>Morale</td>
<td>50%</td>
<td>10%</td>
<td>75%</td>
<td>45%</td>
<td>15%</td>
<td>61%</td>
<td>251%</td>
</tr>
<tr>
<td>Data Integrity</td>
<td>42%</td>
<td>10%</td>
<td>25%</td>
<td>5%</td>
<td>70%</td>
<td>25%</td>
<td>177%</td>
</tr>
<tr>
<td>Technology Adaptability</td>
<td>5%</td>
<td>30%</td>
<td>5%</td>
<td>60%</td>
<td>0</td>
<td>60%</td>
<td>160%</td>
</tr>
<tr>
<td>Requirement Adaptability</td>
<td>80%</td>
<td>20%</td>
<td>60%</td>
<td>75%</td>
<td>20%</td>
<td>5%</td>
<td>260%</td>
</tr>
<tr>
<td>Resource Adaptability</td>
<td>10%</td>
<td>80%</td>
<td>5%</td>
<td>50%</td>
<td>50%</td>
<td>75%</td>
<td>270%</td>
</tr>
<tr>
<td>Cost Reduction</td>
<td>50%</td>
<td>40%</td>
<td>10%</td>
<td>40%</td>
<td>50%</td>
<td>50%</td>
<td>240%</td>
</tr>
<tr>
<td>SUM IMPACT FOR EACH SOLUTION</td>
<td>482%</td>
<td>280%</td>
<td>305%</td>
<td>390%</td>
<td>315%</td>
<td>649%</td>
<td></td>
</tr>
<tr>
<td>Money % of total budget</td>
<td>15%</td>
<td>4%</td>
<td>3%</td>
<td>4%</td>
<td>6%</td>
<td>4%</td>
<td></td>
</tr>
<tr>
<td>Time % total work months/year</td>
<td>15%</td>
<td>15%</td>
<td>20%</td>
<td>10%</td>
<td>20%</td>
<td>18%</td>
<td></td>
</tr>
<tr>
<td>SUM RESOURCES/BENEFIT/RESOURCES RATIO</td>
<td>16:1</td>
<td>14:7</td>
<td>13:3</td>
<td>27:9</td>
<td>12:1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

111111 Unity Method slides
Decomposition Principles
A Teachable Discipline

How to decompose systems into small evolutionary steps:
some principles to apply:

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 can make no 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 they really need it, you will be forgiven a lot of ‘sins’!
13. You can understand things much better, by getting some practical experience (and 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.

From Lao Tzu in Bahn, 1980 (also quoted in Gilb, Principles of Software Engineering Management page 96), Penguin book
Design is the servant of the requirement. If it does not work ‘fire’ it.

Principle 7.
Change designs, based on quantified experience of implementation.
Lean Startup: High Unknowns

Product Development at Lean Startup
Assumes Customers and Markets are Unknown

Customer Development Engineering

Problem: unknown
Solution: unknown

Hypotheses, experiments, insights

Extreme Programming Project

http://www.slideshare.net/venturehacks/the-lean-startup-2
Value Management (Gilb, Evo)
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 2006"
Dynamic (Agile, Evo) design testing: not unlike ‘Lean Startup’

- “... 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 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)


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

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


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

iteration process trying to meet cost targets by either redesign or by sacrificing 'planned capability'
Quinnan: IBM FSD Cleanroom

Dynamic Design to Cost

Design is an iterative process
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.

When the development and test of an increment are complete, an estimate to complete the remaining increments is computed.


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

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.
“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%!]s. 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 reported:

- “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%]s. 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 45 incremental deliveries
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)

This text is cut from Gilb: The Principles of Software Engineering Management, 1988
Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

"Cost management...carries cost management farther by introducing design-to-cost guidance. The method carries cost management into an integrated way to ensure that design, development, and managerial practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method 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)


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

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.' (p. 474)

'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)


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

'\textit{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.

"When the development and test of an increment are complete, an estimate to complete the remaining increments is computed." (p. 474)


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

<table>
<thead>
<tr>
<th>Current Status</th>
<th>Improvements</th>
<th>Reports - E-SAT features</th>
<th>Current Status</th>
<th>Improvements</th>
<th>Survey Engine .NET</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Past</td>
<td>Tolerable</td>
<td>Goal</td>
<td>Past</td>
</tr>
<tr>
<td>Units</td>
<td>Units</td>
<td>%</td>
<td></td>
<td></td>
<td>Units</td>
</tr>
<tr>
<td>75.0</td>
<td>25.0</td>
<td>62.5</td>
<td>0</td>
<td>0</td>
<td>83.0</td>
</tr>
<tr>
<td>14.0</td>
<td>14.0</td>
<td>100.0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>15.0</td>
<td>15.0</td>
<td>107.1</td>
<td>0</td>
<td>0</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>94.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>774.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current Status</th>
<th>Improvements</th>
<th>Report - MR Features</th>
<th>Current Status</th>
<th>Improvements</th>
<th>XML Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Past</td>
<td>Tolerable</td>
<td>Goal</td>
<td>Current Status</td>
</tr>
<tr>
<td>Units</td>
<td>Units</td>
<td>%</td>
<td></td>
<td></td>
<td>Units</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>50.0</td>
<td>14</td>
<td>12</td>
<td>7.0</td>
</tr>
<tr>
<td>20.0</td>
<td>45.0</td>
<td>112.5</td>
<td>35</td>
<td>25</td>
<td>17.0</td>
</tr>
<tr>
<td>4.4</td>
<td>4.4</td>
<td>36.7</td>
<td>0</td>
<td>4</td>
<td>943.0</td>
</tr>
<tr>
<td>101.0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>5.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.0</td>
</tr>
</tbody>
</table>
Only 5 highlights of the 25 impacts are listed here.

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

Release 8.5
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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>BX</th>
<th>BY</th>
<th>BZ</th>
<th>CA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Current Status | Improvements | Goals | Step9
<table>
<thead>
<tr>
<th>Units</th>
<th>Units</th>
<th>%</th>
<th>Past</th>
<th>Tolerable</th>
<th>Goal</th>
<th>Recoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1,00</td>
<td>1,0</td>
<td>50,0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5,00</td>
<td>5,0</td>
<td>100,0</td>
<td>0</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>10,00</td>
<td>10,0</td>
<td>200,0</td>
<td>0</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>0,00</td>
<td>0,0</td>
<td>0,0</td>
<td>0</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>0,00</td>
<td>0,0</td>
<td>0,0</td>
<td>0</td>
<td>60</td>
<td>80</td>
</tr>
<tr>
<td>7</td>
<td>20,00</td>
<td>45,0</td>
<td>112,5</td>
<td>65</td>
<td>35</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>35,00</td>
<td>35,0</td>
<td>70,0</td>
<td>40</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>101,0</td>
<td>91,8</td>
<td>201,8</td>
<td>110</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

Cumulative weekly progress metric

Priority Next week

Warning metrics based

Constraint Target

Estimates Testing Weekly
EVO Plan ConﬁrmIt 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

<table>
<thead>
<tr>
<th>Current Status</th>
<th>Improvements</th>
<th>Reportal - E-SAT features</th>
<th>Current Status</th>
<th>Improvements</th>
<th>Survey Engine .NET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units</td>
<td>Units</td>
<td>%</td>
<td>Past</td>
<td>Tolerable</td>
<td>Goal</td>
</tr>
<tr>
<td>75.0</td>
<td>25.0</td>
<td>62.5%</td>
<td>83.0</td>
<td>48.0</td>
<td>80.0</td>
</tr>
<tr>
<td>14.0</td>
<td>14.0</td>
<td>100.0%</td>
<td>0.0</td>
<td>67.0</td>
<td>100.0</td>
</tr>
<tr>
<td>15.0</td>
<td>15.0</td>
<td>107.1%</td>
<td>4.0</td>
<td>59.0</td>
<td>100.0</td>
</tr>
<tr>
<td>5.0</td>
<td>5.0</td>
<td>96.2%</td>
<td>10.0</td>
<td>397.0</td>
<td>100.0</td>
</tr>
<tr>
<td>3.0</td>
<td>2.0</td>
<td>66.7%</td>
<td>94.0</td>
<td>22990.0</td>
<td>103.9</td>
</tr>
<tr>
<td>1.0</td>
<td>2.0</td>
<td>95.7%</td>
<td>0.0</td>
<td>10.0</td>
<td>13.3%</td>
</tr>
<tr>
<td>4.0</td>
<td>5.0</td>
<td>100.0%</td>
<td>7.0</td>
<td>507.0</td>
<td>51.7</td>
</tr>
<tr>
<td>1.0</td>
<td>12.0</td>
<td>150.0%</td>
<td>5.0</td>
<td>60.0</td>
<td>30.0</td>
</tr>
<tr>
<td>1.0</td>
<td>14.0</td>
<td>100.0%</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Development resources</td>
<td>203.0</td>
<td>91</td>
<td>Development resources</td>
<td>35.0</td>
<td>111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current Status</th>
<th>Improvements</th>
<th>Reportal - MR Features</th>
<th>Current Status</th>
<th>Improvements</th>
<th>XML Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units</td>
<td>Units</td>
<td>%</td>
<td>Past</td>
<td>Tolerable</td>
<td>Goal</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>50.0%</td>
<td>7.0</td>
<td>9.0</td>
<td>81.8</td>
</tr>
<tr>
<td>20.0</td>
<td>45.0</td>
<td>112.5%</td>
<td>17.0</td>
<td>8.0</td>
<td>53.3%</td>
</tr>
<tr>
<td>4.4</td>
<td>4.4</td>
<td>36.7%</td>
<td>0.0</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>Development resources</td>
<td>101.0</td>
<td>86</td>
<td>Development resources</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

© Tom @ Gilb.com June 9, 2014
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.

<table>
<thead>
<tr>
<th>Current Status</th>
<th>Improvement</th>
<th>Goals</th>
<th>Step 6 (week 14)</th>
<th>Step 7 (week 15)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units</td>
<td>100.0</td>
<td>Past</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Maintainability</td>
<td>100.0</td>
<td>Tolerable</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Nunit Tests</td>
<td>100.0</td>
<td>Goal</td>
<td>Estimated Impact</td>
<td></td>
</tr>
<tr>
<td>Maintainability.Doc.Code</td>
<td>100.0</td>
<td>Estimated Impact</td>
<td>Actual Impact</td>
<td></td>
</tr>
<tr>
<td>InterviewerConsole</td>
<td>100.0</td>
<td>Estimated Impact</td>
<td>Actual Impact</td>
<td></td>
</tr>
<tr>
<td>NUnitTests</td>
<td>100.0</td>
<td>0</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>PeerTests</td>
<td>100.0</td>
<td>0</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>Robustness.Correctness</td>
<td>100.0</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Robustness.BoundaryConditions</td>
<td>100.0</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ResourceUsage.CPU</td>
<td>100.0</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Maintainability.Doc.Code</td>
<td>100.0</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>SynchronizationStatus</td>
<td>100.0</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Speed**

**Maintainability**

**Nunit Tests**

**PeerTests**

**TestDirectorTests**

**Robustness.Correctness**

**Robustness.BoundaryConditions**

**ResourceUsage.CPU**

**Maintainability.DocCode**
Raising the Levels of Maintainability like ‘Mean Time To Fix a Bug’

Current Level → Minimum Future Level → Competitive and economic Goal level
Raising the Levels of Maintainability
Multiple Attributes of Technical Debt

- Portability
- Scalability
- Adaptability
- Testability
Broader ‘Maintainability’ Concepts
ALL quantified, with a defined Scale of measure in CE-5

Performance
  - Quality
    - Availability
      - Reliability
      - Maintainability
      - Integrity
        - Threat
        - Security
    - Adaptability
      - Flexibility
        - Connectability
        - Tailorability
          - Extendibility
          - Interchangeability
        - Upgradeability
          - Installability
          - Portability
          - Improveability
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)
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
**Restore Speed:**


*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?
The Green Week: Reducing Technical Debt by Engineering

by Tom & Kai Gilb

Our client Conformit.com has used our Evo Agile Method (1) successfully since 2009 (2). They have adapted 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-offering's prospects. 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 Conformit, led by their method champion Trond Johannsen, made and reported in 2005; two years after adopting Evo.

When we started in 2003, Conformit 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 arrived at something like 26 distinct quality improvements (for example, usability improvements) or performance capacity improvements. The changes arrived at were users and Conformit'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 'suffer' 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 remotely close 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, i.e. how 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' in 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.

<table>
<thead>
<tr>
<th>User Week 1</th>
<th>User Week 2</th>
<th>User Week 3</th>
<th>Developer Week 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select a Goal</td>
<td>Select a Goal</td>
<td>Select a Goal</td>
<td>Select a Goal</td>
</tr>
<tr>
<td>Brainstorm Designs</td>
<td>Brainstorm Designs</td>
<td>Brainstorm Designs</td>
<td>Brainstorm Designs</td>
</tr>
<tr>
<td>Estimate Design</td>
<td>Estimate Design</td>
<td>Estimate Design</td>
<td>Estimate Design</td>
</tr>
<tr>
<td>Implement</td>
<td>Implement</td>
<td>Implement</td>
<td>Implement</td>
</tr>
<tr>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td>Update Progress to Goal</td>
<td>Update Progress to Goal</td>
<td>Update Progress to Goal</td>
<td>Update Progress to Goal</td>
</tr>
</tbody>
</table>

Figure 1: The weekly development cycle, with the Green Weeks.

The key idea here is that we start by quantifying as requirements, all Conformit 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, Run Time, Peer Tests, Peer Director Tests, Robustness, Correctness, Robustness, Boundary Conditions, Resource Usage CPU, Maintainability Doc/Code, Synchronization Status.
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”

● 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](mailto: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:
    the book Beck and others refer to
  - Competitive Engineering (2005)
  - Evo: (Kai, evolving, 55 iterations)
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.mountaingoatsoftware.com/?p=77


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 2004 page 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?

There is a detailed paper on these questions at Gilb.com