WHAT'S WRONG WITH AGILE METHODS?

SOME PRINCIPLES AND VALUES TO ENCOURAGE QUANTIFICATION

Tom Gilb   Independent Consultant   Tom@Gilb.com

Iver Holtersvei 2, NO-1410 Kolbotn, Norway.   +47 66801697

Lindsey Brodie   Middlesex University   L.Brodie@mdx.ac.uk

WHAT'S WRONG WITH AGILE METHODS?

SOME PRINCIPLES AND VALUES TO ENCOURAGE QUANTIFICATION

ABSTRACT

Current agile methods could benefit from using a more quantified approach across the entire implementation process (that is, throughout development, production and delivery). The main benefits of adopting such an approach include improved communication of the requirements and, better support for feedback and progress tracking.

This chapter first discusses the benefits of quantification, then outlines a proposed approach (Planguage) and, finally describes an example of its successful use (a case study of the 'Confirmit' product within a Norwegian organization, 'FIRM').

INTRODUCTION

Agile Software Methods (Agile Alliance 2006) have insufficient focus on quantified performance levels (that is, metrics stating the required qualities, resource savings and workload capacities) of the software being developed. Specifically, there is often no quantification of the main reasons why a project was funded (that is, metrics stating the required business benefits, such as business advancement, better quality of service and financial savings). This means projects cannot directly control the delivery of benefits to users and stakeholders. In turn, a consequence of this is that projects cannot really control the corresponding costs of getting the main benefits. In other words, if you don't *estimate* quantified requirements, then you won't be able to get a realistic *budget* for achieving them. See Figure 1 for a scientist's (Lord Kelvin's) opinion on the need for numerical data!

"In physical science the first essential step in the direction of learning any subject is to find principles of numerical reckoning and practicable methods for measuring some quality connected with it. I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the state of Science, whatever the matter may be."

Lord Kelvin, 1893

Figure 1. A statement made by Lord Kelvin on the importance of measurement. From http://zapatopi.net/kelvin/quotes.html [Last Accessed: April 2006]

Further, quantification must be utilized throughout the duration of an agile project, not just to state requirements but, to drive design, assess feedback and, track progress. To spell this last point out, *quantification of the requirements* (what do we want to control?) is only a first step in getting control. The next steps, based on this quantification, are *design estimation* (how good do we think our solutions are?) and *measurement of the delivered results* (how good were the solutions in practice?). The key issue here is the active use of quantified data (requirements, design estimates and feedback) to drive the project design and planning.

One radical conclusion to draw, from this lack of quantification, is that current conventional agile methods are not really suitable for development of industrial products. The rationale for this being that industry is not simply interested in delivered 'functionality' alone; they probably already have necessary business functions at some level. Projects must produce competitive products, which means projects must deliver specific performance levels (including qualities and savings). To address this situation, it is essential that the *explicit* notion of quantification be added to agile concepts.

See Figure 2 for a list of the benefits to agile development of using quantification.

Benefits of the Use of Quantification in Agile Development

• Simplify requirements (if the top few requirements are quantified, there is less need for copious documentation as the developers are focused on a clearer, simpler 'message');

• Communicate quality goals much better to all parties (that is, users, customers, project

management, developers, testers, and lawyers);
• Contract for results. Pay for results only (not effort expended). Reward teams for results achieved. This is possible as success is now measurable;
• Motivate technical people to focus on real business results;
• Evaluate solutions/designs/architectures against the quantified quality requirements;
• Measure evolutionary project progress towards quality goals and get early & continuous improved estimates for time to completion;
• Collect numeric historical data about designs, processes, organizational structures for future use. Use the data to obtain an understanding of your process efficiency, to bid for funding for improvements and to benchmark against similar organizations!

Figure 2. What can we do better in agile development (or ‘at all’), if we quantify requirements

DEFINING QUALITY

The main focus for discussion in this chapter will be the quality characteristics, because that is where most people have problems with quantification. A long held opinion of one of the authors of this chapter (Tom Gilb) is that all qualities are capable of being expressed quantitatively (see Figure 3).

The Principle Of 'Quality Quantification’

All qualities can be expressed quantitatively, 'qualitative' does not mean unmeasurable.

Tom Gilb

Figure 3. Tom Gilb’s opinion that all qualities can be expressed numerically

A Planguage definition of ‘quality’ is given in Figure 4. Planguage is a planning language and a set of methods developed by Tom Gilb over the last three decades (Gilb 2005). This next part of the chapter will outline the Planguage approach to specifying and using quantitative requirements to drive design and determine project progress.

Definition of Quality

Quality is characterized by these traits:

• A quality describes ‘how well’ a function is done. Qualities each describe the partial effectiveness of a function (as do all other performance attributes).

• Relevant qualities are either valued to some degree by some stakeholders of the system - or they are not relevant. Stakeholders generally value more quality, especially if the increase is free, or lower cost, than the stakeholder-perceived value of the increase.

• Quality attributes can be articulated independently of the particular means (the designs and architectures) used for reaching a specific quality level, even though achievement of all quality levels depend on the particular designs used to achieve quality.

• A particular quality can potentially be a described in terms of a complex concept, consisting of multiple elementary quality concepts, for example, ‘Love is a many-splendored thing!’

• Quality is *variable* (along a definable scale of measure: as are all scalar attributes).

• Quality levels are capable of being specified *quantitatively* (as are *all* scalar attributes).

• Quality levels can be *measured* in practice.

• Quality levels can be *traded off* to some degree; with other system attributes valued more by stakeholders.

• Quality can never be perfect (no fault and no cost) in the real world. There are some valued levels of a particular quality that may be outside the state of the art at a defined future time and circumstance. When quality levels increase towards perfection, the resources needed to support those levels tend towards infinity.

(Gilb 2005)

Figure 4. Planguage definition of ‘quality’

QUANTIFYING REQUIREMENTS

Planguage enables capture of quantitative data (metrics) for performance and resource requirements. A scalar requirement, that is, either a performance or resource requirement, is specified by identifying a relevant scale of measure and stating the current and required levels on that scale. See Figure 5, which is an example of a performance requirement specification. Notice the parameters used to specify the levels on the scale (that is, Past, Goal. And Fail.

Tag:

Scale:

Meter:

Past:

Goal:

Fail:

Figure 5. Planguage parameters used to specify a performance requirement

EVALUATING DESIGNS

Impact Estimation (IE) is the Planguage method for evaluating designs. See Table 1, which shows an example of a simple IE table. The key idea of an IE table is to put the potential design ideas against the quantified requirements and estimate the impact of each design on each of the requirements. If the current level of a requirement is known (its baseline, 0%), and the target level is known (its goal or budget depending on whether a performance requirement (an objective) or a resource requirement respectively, 100%), then the percentage impact of the design in moving towards the performance/resource target can be calculated. Because the values are converted into percentages, then simple arithmetic is possible to calculate the cumulative effect of a design idea (sum of performance and sum of cost) and the performance to cost ratio (see Table 1). You can also sum across the designs (assuming the designs are capable of being implemented together and that their impacts don't cancel each other out) to see how much design you have that is addressing an individual requirement.

Table 1 also shows how you can take into account any uncertainties in your estimates. An additional feature, not shown here, is to assess the credibility of each estimate by assigning a credibility factor between 0.0 and 1.0. Each estimate can then be multiplied by its credibility factor to moderate it.

While such simple arithmetic does not represent the complete picture, it does give a convenient means of quickly identifying the most promising design ideas. Simply filling in an IE table gives a much better understanding of the strengths and weaknesses of the various designs with respect to meeting all the requirements.

Design Ideas-> Requirements: Goals and Budgets	Idea1 Impact Estimates	Idea 2 Impact Estimates	Sum for Requirement/ (Sum of Percentage Impacts)	Sum of Percentage Uncertainty Values	Safety Deviation
Reliability 300 <-> 3000 hours MTBF	1650hr ±0	840hr ±240			
	61%±0	31%±9%	92%	±9%	-108%
Usability 20 <-> 10 minutes	1min. ±4	6 min. ±9			
	10%±40%	60%±90%	70%	±130%	-130%
Sum of Performance	71%	91%			
Capital 0 <-> 1 million US$	500K ±200K	100K ±200K			
	50%±20	10%±20	60%	±40%	-10%
Maintenance 1.1M <-> 100K/year US$	0 K$/Y ±180K	1 M$/Y ±720K			
	0%± 18%	100%±72%	100%	±90%	-50%
Sum of Cost	50%	110%			
Performance to Cost Ratio	1.42 (71/50)	0.83 (91/110)			

Table 1. An example of a simple IE table (Gilb 2005)

Table 1 simply shows estimates for potential design ideas. However, you can also input the actual measurements (feedback) from implementing the design ideas. There are two benefits to this: you learn how good your estimates where for the design ideas implemented, and you learn how much progress you have made towards your target levels. You can then use all the IE table data as a basis to decide what to implement next.

EVOLUTIONARY DELIVERY

The final Planguage method we will discuss is Evolutionary Project Management (Evo). Evo demands include the following:

• that a system is developed in a series of small increments (each increment typically taking between 2% and 5% of the total project timescale to develop);

• that each increment is delivered for real use (maybe as Beta or Fieled trial) by real 'users' (any stakeholder) as early as possible (to obtain business benefits, and feedback, as soon as possible).

• that the feedback from implementing the Evo steps is used to decide on the contents of the next Evo step;

• that the highest value Evo steps are delivered earliest, to maximize the business benefit.

Note that '*delivery*' of requirements is the key consideration. Each delivery is done within an Evo step. It may, or may not, include the building or creation of the increment (Some Evo steps may simply be further roll-out of *existing* software);

Development of necessary components will occur incrementally, and will be continuing in parallel while Evo steps are being delivered to stakeholders. Most development will only start when the decision has been taken to deliver it as the next Evo step. However, there probably will be some increments that have longer lead-times for development, and so their development will need to start early in anticipation of their future use. A project manager should always aim to 'buffer' his developers in case of any development problems by having in reserve some components (readied in the 'Backroom') ready for delivery. The 'Frontroom' being the term for the interface between developers and stakeholders – for implementation of the steps.

Planguage approach to Change

It is important to note that the quantified requirements, designs and implementation plans are not 'frozen,' they must be subject to negotiated change, over time. As Beck points out, "Everything in software changes. The requirements change. The design changes. The business changes. The technology changes. The team changes. … The problem isn't change, per se, … the problem, rather, is the inability to cope with change when it comes" (Beck 2000).

Planguage's means of dealing with change is as follows:

• performance and resource requirements are quantified to allow rapid communication of any changes in levels;

• IE tables allows dynamic reprioritization of design ideas and helps track progress towards targets;

• Evo enables all types of change to be catered for 'in-flight', as soon as possible. There is regular monitoring of what the best next Evo step to take.

DESCRIPTION OF THE PLANGUAGE PROCESS

To summarize and show how the methods (for quantifying requirements, evaluating designs and evolutionary delivery) described earlier in this chapter fit together, here is a description of the Planguage process for a project:

1. Gather from all the key stakeholders the top few (5 to 20) most critical goals that the project needs to deliver. Give each goal a reference name (a tag).

2. For each goal, define a scale of measure and a 'final' goal level. For example:

Reliable:

Scale: Mean Time Before Failure,

Goal: 1 month.

3. Define approximately 4 budgets for your most limited resources (for example, time, people, money, and equipment).

4. Write up these plans for the goals and budgets (*Try to ensure this is kept to only one page*).

5. Negotiate with the key stakeholders to formally agree the goals and budgets.

6. Draw up a list of design ideas: Ensure that you decompose the design ideas down into the smallest increments that can be delivered (these are potential Evo steps). Use Impact Estimation (IE) to evaluate your design ideas contributions towards meeting the requirements. Look for small increments with large business value. Note any dependencies, and draw up an initial rough

Evo plan, which sequences the Evo steps. In practice, decisions about what to deliver for the next Evo step will be made in the light of feedback (that is when the results from the deliveries of the previous Evo steps are known). Plan to deliver some value (that is, progress towards the required goals) in *weekly* (or shorter) increments (Evo steps). Aim to deliver highest possible value as soon as possible.

7. Deliver the project in Evo steps.

• Report to project sponsors after each Evo step (weekly, or shorter) with your best available estimates or measures, for each performance goal and each resource budget. O*n a single page,* summarize the *progress to date* towards achieving the goals and the costs incurred.

• Discuss with your project sponsors and stakeholders what design ideas you should deliver in the next Evo step. This should be done in the light of what has been achieved to date and what is left to do. Maximizing the business benefit should be the main aim.

8. When all goals are reached: 'Claim success and move on.' Free remaining resources for more profitable ventures

Ten Planguage Values for an Agile Project

Simplicity

1. Focus on real stakeholder values.

Communication

2. Communicate stakeholder values quantitatively.

3. Estimate expected results and costs for weekly steps.

Feedback

4. Give deployment, of your understanding, weekly, to stakeholders, in their environment.

5. Measure all critical aspects of the deployment.

6. Analyze deviation from 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') and, 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. Do not be limited by your craft background, in serving your paymasters

Figure 6. Planguage's ten values for an agile project based around Beck's Four Values for XP (Beck 2000 Page 29)

Planguage Project Management Policy

- The project manager, and the project, will be judged exclusively on the relationship of progress towards achieving the goals versus the amounts of the budgets used.
- The project team will do anything legal and ethical to deliver the goal levels within the budgets.
- The team will be paid and rewarded for benefits delivered in relation to cost.
- The team will find their own work process and their own design.
- As experience dictates, the team will be free to suggest to the project sponsors (stakeholders) adjustments to 'more realistic levels' of the goals and budgets.

Figure 7. Planguage policy for project management

CASE STUDY OF THE 'Confirmit' PRODUCT

Tom Gilb and his son, Kai taught the Planguage methods to the FIRM (Future Information Research Management, in Norway) organization. Subsequently, FIRM used these methods in the development of their Confirmit product. The results were impressive, so much so that they decided to write up about their experiences (Johansen 2004, Johansen and Gilb 2005). In this section, some of the details from this Confirmit product development project are presented.

Use of Planguage Methods

First, the quantified requirements were specified, including the target levels. Next, a list of design ideas (solutions) was drawn up (see Figure 8 for an example of an initial design idea specification).

Recoding:

Type: Design Idea [Confirmit 8.5].

Description: Make it possible to recode a marketing variable, on the fly, from Reportal.

Estimated effort: 4 team days.

Figure 8. A brief specification of the design idea, ‘Recoding’

The impacts of the design ideas on the requirements were then estimated. The most promising design ideas were included in an Evo plan, which was presented using an Impact Estimation (IE) table (see Tables 2 and 3, which show the part of the IE table applying to Evo Step 9. Note these tables also include the actual results after implementation of step 9). The design ideas were evaluated with respect to ‘value for clients’ versus ‘cost of implementation’. The ones with the highest value-to-cost ratio were chosen for implementation in the early Evo steps. Note that value can sometimes be defined by *risk removal* (that is, implementing a technically challenging solution early can be considered high value if implementation means that the risk is likely to be subsequently better understood). The aim was to deliver improvements to real external stakeholders (customers, users), or at least to internal stakeholders (for example, delivering to internal support people, who use the system daily and so can act as 'clients').

	EVO STEP 9: DESIGN IDEA: 'Recoding'			
	Estimated Scale Level	Estimated % Impact	Actual Scale Level	Actual % Impact
REQUIREMENTS				
Objectives				
Usability.Productivity 65 <-> 25 minutes Past: 65 minutes. Tolerable: 35 minutes. Goal: 25 minutes.	65 – 20 = 45 minutes	50%	65 - 38 = 27 minutes	95%
Resources				
Development Cost 0 <-> 110 days	4 days	3.64%	4 days	3.64%

Table 2. A simplified version of part of the IE table shown in Table 3. It only shows the objective, 'Productivity' and the resource, 'Development Cost' for Evo Step 9, 'Recoding' of the Marketing Research (MR) project. The aim in this table is to show some extra data, and some detail of the IE calculations. Notice the separation of the requirement definitions for the objectives and the resources. The Planguage keyed icon '<->' means 'from baseline to target level'. On implementation, Evo Step 9 alone moved the Productivity level to 27 minutes, or 95% of the way to the target level

The IE table was used as a tool for controlling the qualities: estimated figures and actual measurements were input into it. Each next Evo step was then decided, based on the results achieved *after* implementation and delivery of the subsequent step.

Note, the results were *not* actually measured with statistical accuracy by doing a scientifically correct large-scale survey (although FIRM are currently considering doing this). The impacts described for Confirmit 8.0 (the 'Past' levels) are based on internal usability tests, productivity tests, performance tests carried out at Microsoft Windows ISV laboratory in Redmond USA, and from direct customer feedback.

Current Status	Improvements		Goals			Step 9			
						Design = 'Recoding'			
						Estimated impact		Actual impact	
Units	Units	%	Past	Tolerable	Goal	Units	%	Units	%
			Usability.Replaceability (feature count)						
1.00	1.0	50.0	2	1	0				
			Usability.Speed.New Features Impact (%)						
5.00	5.0	100.0	0	15	5				
10.00	10.0	200.0	0	15	5				
0.00	0.0	0.0	0	30	10				
			Usability.Intuitiveness (%)						
0.00	0.0	0.0	0	60	80				
			Usability.Productivity (minutes)						
20.00	**45.0**	**112.5**	**65**	**35**	**25**	**20.00**	**50.00**	**38.00**	**95.00**
			Development resources						
	101.0	**91.8**	**0**		**110**	**4.00**	**3.64**	**4.00**	**3.64**

Table 3. Details of the real IE table, which was simplified in Table 2. The two requirements expanded in Table 1 are highlighted in bold. The 112.5 % improvement result represents a 20 minutes level achieved after the initial 4 day stint (which landed at 27 minutes, 95%) . A few extra hours were used to move from 27 to 20 minutes, rather than use the next weekly cycle.

The Results Achieved

Due to the adoption of Evo methods there were focused improvements in the product quality levels. See Table 4, which gives some highlights of the 25 final quality levels achieved for Confirmit 8.5. See also Table 5, which gives an overview of the improvements by function (that is, product component) for Confirmit 9.0. No negative impacts are hidden. The targets were largely all achieved on time.

DESCRIPTION OF REQUIREMENT / WORK TASK	**PAST**	**CURRENT 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 (MR) report	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 Confirmit Web Services without any user documentation or any other aid	15 min	5 min
Workload Capacity.Runtime.Concurrency: Maximum number of simultaneous respondents executing a survey with a click rate of 20 seconds and a response time < 500 milliseconds, given a defined [Survey-Complexity] and a defined [Server Configuration, Typical].	250 users	6000 users

Table 4. Improvements to product quality levels in Confirmit 8.5

FUNCTION	PRODUCT QUALITY	DEFINITION (quantification)	CUSTOMER VALUE
Authoring	Intuitiveness	Probability that an inexperienced user can intuitively figure out how to set up a defined Simple Survey correctly.	Probability increased by 175% (30% to 80%)
Authoring	Productivity	Time in minutes for a defined advanced user, with full knowledge of Confirmit 9.0 functionality, to set up a defined advanced survey correctly.	Time reduced by 38%
Reportal	Performance	Number of responses a database can contain if the generation of a defined table should be run in 5 seconds.	Number of responses increased by 1400%
Survey Engine	Productivity	Time in minutes to test a defined survey and identify 4 inserted script errors, starting from when the questionnaire is finished to the time testing is complete and ready for production. (Defined Survey: Complex Survey, 60 questions, comprehensive JScripting.)	Time reduced by 83% and error tracking increased by 25%
Panel Management	Performance	Maximum number of panelists that the system can support without exceeding a defined time for the defined task, with all components of the panel system performing acceptably.	Number of panelists increased by 1500%
Panel Management	Scalability	Ability to accomplish a bulk-update of X panelists within a timeframe of Z seconds.	Number of panelists increased by 700%
Panel Management	Intuitiveness	Probability that a defined inexperienced user can intuitively figure out how to do a defined set of tasks correctly.	Probability increased by 130%

Table 5. Some detailed results by function (product component) for Confirmit 9.0

The customers responded very favorably (see Figure 9).

"I just wanted to let you know how appreciative we are of the new 'entire report' export functionality you recently incorporated into the Reportal. It produces a fantastic looking report, and the table of contents is a wonderful feature. It is also a HUGE time saver."

Figure 9. An example of pilot customer (Microsoft) feedback

On the *second* release (Confirmit 9.0) using Planguage, and specifically the Evo method, the Vice President (VP) of Marketing proudly named the Evo development method on the FIRM website (see Figure 10. A line executive bragging about a development method is somewhat exceptional!).

"FIRM, through evolutionary development, is able to substantially increase customer value by focusing on key product qualities important for clients and by continuously asking for their feedback throughout the development period. Confirmit is used by the leading market research agencies worldwide and Global 1000 companies, and together, we have defined the future of online surveying and reporting, represented with the Confirmit 9.0."

Figure 10. Comments by FIRM's VP of Marketing, Kjell Øksendal

Details of the quantified improvements were also given to their customers (see Figure 11, which is an extract from the product release for Confirmit 9.0 published on the organization's website).

News release

2004-11-29: Press Release from FIRM

New version of Confirmit increases user productivity up to 80 percent

NOVEMBER 29th, 2004: FIRM, the world's leading provider of online survey & reporting software, today announced the release of a new version of Confirmit delivering substantial value to customers including increased user productivity of up to 80 percent.

FIRM is using Evolutionary (EVO) development to ensure the highest focus on customer value through early and continuous feedback from stakeholders. A key component of EVO is measuring the effect new and improved product qualities have on customer value. Increased customer value in Confirmit 9.0 includes:

* Up to 175 percent more intuitive user interface*
* Up to 80 percent increased user productivity in questionnaire design and testing*
* Up to 1500 percent increased performance in Reportal and Panel Management*

Figure 11. Confirmit 9.0 release announcement from the FIRM website http://www.firmglobal.com. It gives detail about the method and the quantified product results

Impact on the developers

Use of Evo has resulted in increased *motivation* and *enthusiasm* amongst the FIRM developers, because it has opened up '*empowered creativity' (Trond Johansen, FIRM Project Director)*. The developers can now determine their own design ideas, and are not subject to being dictated the design ideas by marketing and/or customers, who often tend to be amateur technical designers. Daily, and more often, product builds, called Continuous Integration (CI, using Cruise Control), were introduced. Evo combined with CI, is seen as a vehicle for innovation and inspiration. Every week, the developers get their work out onto the test servers, and receive feedback.

By May 2005, FIRM had adopted the approach of using a ‘Green Week’ once monthly. In a Green Week, the internal stakeholders are given precedence over the client stakeholders and can choose what product improvements they would like to see implemented. The FIRM developers chose to focus on the evolutionary improvement of about 12 internal stakeholder qualities (such as testability and maintainability).

Initial difficulties in implementing Planguage

Even though Planguage was embraced, there were parts of Planguage that were initially difficult to understand and execute at first. These included:

- Defining good requirements (‘Scales’ of measure) sometimes proved hard; (they only had one day training initially, but after the first release saw the value in a weeks training!)
- It was hard to find ‘Meters’ (that is, ways of measuring numeric qualities, to test the current developing quality levels), which were practical to use, and at the same time measured real product qualities;

• Sometimes it took more than a week to deliver something of value to the client; (this was mainly a test synchronization problem they quickly overcame).

• Testing was sometimes ‘postponed’ in order to start the next step. Some of these test postponements were then not in fact done in later testing.

Lessons learned with respect to Planguage, especially the Evo method

Some of the lessons learnt about the use of Planguage, and especially the Evo method, included:

• Planguage places a focus on the measurable product qualities. Defining these clearly and testably requires training and maturity. It is important to *believe* that everything can be measured and to seek guidance if it seems impossible;

• Evo demands dynamic re-prioritization of the next development steps using the ratio of delivering value for clients versus the cost of implementation. Data to achieve this is supplied by the weekly feedback. The greatest surprise was the power of focusing on these ratios. What seemed important at the start of the project may be replaced by other solutions based on gained knowledge from previous steps;

• an *open architecture* is a pre-requisite for Evo;

• management support for changing the software development process is another pre-requisite, but this is true of any software process improvement;

• The concept of daily builds, CI, was valuable with respect to delivering a new version of the software every week;

• It is important to control expectations. ‘Be humble in your promises, but overwhelming in your delivery’ is a good maxim to adopt;

• There needed to be increased focus on feedback from clients. The customers willing to dedicate time to providing feedback need identifying. Internal stakeholders (like sales and help desk staff) can give valuable feedback, but some interaction with the actual customers is necessary;

• Demonstrate new functionality automatically, with screen recording software or early test plans. This makes it easier for internal and external stakeholders to do early testing;

• Tighter integration between Evo and the test process is necessary.

Conclusions of the Case Study

The positive impacts achieved on the Confirmit product qualities has proved that the Evo process is better suited than the Waterfall process (used formerly) to developing the Confirmit product. Overall, the whole FIRM organization embraced Planguage, especially Evo. The first release, Confirmit 8.5 showed some of Planguage’s great potential. By the end of November 2004, with the second release (Confirmit 9.0), there was confirmation that the Evo method can, consistently and repetitively, produce the results needed for a competitive product. Releases 9.5 and 10.0 of

Confirmit continued this pattern of successful product improvements delivered to the customers (as of November 2005).

It is expected that the next versions of Confirmit will show even greater maturity in the understanding and execution of Planguage. The plan is to continue to use Planguage (Evo) in the future.

CHAPTER SUMMARY

Use of quantified requirements throughout the implementation of a project can provide many benefits as has been demonstrated by the FIRM organization's use of Planguage (including Evo). The key messages of this chapter can be summarized in twelve Planguage principles (see Figure 12). By adopting such principles, agile methods would be much better suited for use in the development of industrial products.

Twelve Planguage Principles

1. Control projects by a small set of quantified critical results (that is, not stories, functions, features, use cases, objects, etc.). Aim for them to be stated on one page!

2. Make sure those results are *business* results, not technical.

3. Align your project with your financial sponsor's interests!

4. Identify a set of designs. Ensure you decompose the designs into increments of the smallest possible deliverables.

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

6. Select designs with the best performance to cost ratios; do them first.

7. Decompose the workflow and/or deliveries, into weekly (or 2% of budget) time boxes.

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

9. Change designs, based on quantified experience of implementation (feedback).

10. Change requirements, based in quantified experience (new inputs).

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

12. Involve the stakeholders, every week, in *actually using* increments.

Figure 12. Twelve Gilb Planguage principles for project management/software development.

REFERENCES

Agile Alliance, 2006, URL: http://www.agilealliance.com/ [Last Accessed: April 2006].

Beck, Kent, *Extreme Programming Explained: Embrace Change*, 2000, Addison-Wesley. ISBN 0201616416.

Gilb, Tom, *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*, 2005, Elsevier Butterworth-Heinemann, ISBN 0750665076.

Johansen, Trond, FIRM: From Waterfall to Evolutionary Development (Evo) or How we rapidly created faster, more user-friendly, and more productive software products for a competitive multi-national market, *Proceedings of European Software Process Improvement (EuroSPI)*, Trondheim, Norway, November 10-12, 2004, Torgeir Dingsøyr (Ed.), Lecture Notes in Computer Science 3281, Springer 2004, ISBN 3-540-23725-9. See also Proceedings of INCOSE 2005 (Johansen and Gilb 2005) and FIRM website, http://www.confirmit.com/news/release_20041129_confirmit_9.0_mr.asp/ [Last Accessed: February 2006].

AUTHOR BIOGRAPHIES

Tom Gilb has been an independent consultant, teacher and author, since 1960. He works mainly with multinational clients; helping improve their organizations, and their systems engineering methods.

Tom's latest book is 'Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage' (2005). Other books are 'Software Inspection' (Gilb and Graham 1993) and 'Principles of Software Engineering Management' (1988). His 'Software Metrics' book (1976 Out of Print) has been cited as the initial inspiration (IBM, Radice) for what is now CMMI Level 4.

Tom's key interests include business metrics, evolutionary delivery, and further development of his planning language, 'Planguage'.

Tom@Gilb.com

http://www.Gilb.com

Lindsey Brodie is currently studying for a PhD. at Middlesex University, UK. She holds a MSc. in Information Systems Design. She edited Tom Gilb's latest book, Competitive Engineering. Previously she worked for many years for International Computers Limited (ICL), UK carrying out technical project support, product support (operating system and database support), and business process and management consultancy.

Lindsey is a member of the British Computer Society and a Chartered Engineer (MBCS CITP CEng).

Author Contact: L.Brodie@mdx.ac.uk

END OF CHAPTER

Version 14 April 2006 01:05 TG