Agile Record

The Magazine for Agile Developers and Agile Testers

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

January 2011

issue 5

Agile Aspects of Planguage for Cost-Effective Engineering

by Tom Gilb & Lindsey Brodie

Planguage (Planning Language) is a comprehensive, but not exhaustive, set of tools for planning systems engineering. It encompasses language constructs to capture system requirements, designs and delivery increments. It also includes well-defined processes for some of the systems engineering processes, principally requirements specification, quality control, and project management.

Planguage has been developed over many years in industry. The guiding principles were to support quantified requirements and the evolutionary delivery of such requirements. As such, Planguage provides a strong capability to underpin and improve existing agile practices. It achieves this through providing enhanced measurement of progress, from setting the objectives to supporting testing to evaluate deliverables. Also by supporting system delivery being achieved as a series of small, early, high-value evo steps.

This paper discusses certain agile aspects of Planguage but does not describe all its details. If the reader is encouraged to find out more about Planguage, then they should see (Gilb 2005).

Defining Agile

The term 'agile' within Planguage is considered to primarily mean 'adapting successfully to new circumstances'. Traditional dictionary definitions such as 'moving quickly and lightly' (Webster) define only one way in which to be agile, they do not cover all the possible means for being agile, in terms of adapting to circumstances. Indeed, some of the alternative and supplementary means of 'adapting successfully', may involve the opposite ideas to being quick and light. For example, ideas such as being conservative enough to make sure things will actually work successfully, rather than changing too quickly to an untested way. Simply being quick and light are not necessarily the right strategies for meeting the requirements of a project or organization, especially if there are no new circumstances.

The previous point highlights one of the dominant characteristics of Planguage. Planguage emphasizes the 'ends', rather than the 'means'. This alone can be seen as key to the agility of Planguage as every aspect of a system is subject to consideration for change (all lower priority requirements, designs, deliverables, systems engineering processes and project management processes) in order to give maximum effect to the satisfaction of the higher prioritized objectives, when responding to new information and situations.

The key principles of agile as defined by the agile community (Agile Principles 2001) include "early and continuous delivery of valuable software", "welcoming changing requirements", "delivering working software frequently", "business people and developers must work together daily", and "working software is the primary measure of progress". Planguage with its focus on 'ends' can support these principles. What Planguage demands in addition though is that progress is measured through quantified requirements and results.

In some respects, agility in Planguage can be though of as being quantified by the *efficiency* concept. Agility is the effectiveness of meeting a defined set of requirements, in relation to the cost and timescales. The lower the cost and timescales of meeting all the requirements, the more agile the method. As such, Planguage focuses on understanding the objectives as quantified, measurable requirements, and on identifying and delivering high-value evo steps to deliver early stakeholder value and obtain feedback from real deployment. So, the key question is not whether a given method is light or heavy! The only rational question is, 'What is the smartest way to satisfy the requirements?' Many in the agile community have never understood this notion, and therefore they seem to embrace lightness itself, even if that is too light for purpose.

Requirements Language Agility

Specification of Planguage Requirements

In order to aid communication amongst the stakeholders, Planguage defines a very comprehensive set of statements and expressions to specify information about a requirement. Over 90% of a typical Planguage requirement specification can be additional information filling in the background details, such as the relationships, priorities, risks, dependencies and change control. The Planguage user is at liberty to specify what is mandatory, what is optional, and what is discouraged, for any type of requirement specification according to its potential different system contexts. A specification can grow and be modified over time, as a project develops and obtains more information and insights. For example a requirement can start life as a simple name, like 'Agile'. It can then have its overall aim defined:

Ambition: to be more effective than competitors in meeting our requirements efficiently.

It can then be improved by adding initial attempts at quantification, such as:

Scale: % Product Cost to meet requirements compared to Benchmark.

Past [This Organization, New Product Development, End of Last Year]: 100%.

Goal [This Organization, New Product Development, End of This Year]: 95%.

With Past and Goal, a notion of where the system is currently and where it should be at some future time is introduced, and of course these are measurable so we can understand our progress. To increase clarity, other details might be added:

Product Cost: defined as: Product Development Cost as a percentage of real or projected system costs over product lifetime.

Authority: Corporate Policy paragraph 6.3.

Dependencies: Mandated policies such as safety, security, and ethics.

Risks: R1: Long-term effects of changes to the development process might be hidden for too long.

Issues: I1: How long a life cycle scope shall we include? In particular, does it include on-going costs when product is not sold?

Such specification provides a lightweight means of capturing and communicating the key aspects of a system. The use of Planguage templates ensures that the main specification details are considered and aids readers to find the information that they are seeking rapidly.

Reuse Aspects

Reuse contributes to agility because you do not have to take the effort to redefine things from scratch, and the reused items are more likely to be safe to use than quickly made-up definitions. Planguage provides many opportunities for reuse of specifications, for example, tag definitions and concepts.

Concepts: Planguage currently defines over 640 concepts in the Planguage Concept Glossary (Gilb 2010). These are basic systems engineering concepts, such as 'Quality', 'Requirement', 'Constraint' and 'Goal'. They are assigned a specific meaning that is consistent with the rest of Planguage. They are then referred to by a tag, preferably but not always, with a leading capital letter in order to announce that they are formally defined. These concepts are reused constantly and frequently. Many of them provide the core language for specification, such as 'Scale', 'Goal' and 'Ambition' (see the previous example specifying 'Agile').

Templates: Planguage provides templates to aid users with their specification. These templates are often adopted and modified at the corporate level by my clients and readers. The template definitions are fairly stable over time, and apply to all projects. By contrast, many corporations have no standard definitions of the most basic concepts, and they offer nothing to be systematically reused by their engineers. This tends to lead directly to ambiguity and wasted effort.

Tag Definitions: Almost any set of words or symbols can be name tagged with a unique tag. Whenever this tag is referred to, we are *reusing* the initial definition of the tag. This reuse principle applies as many levels of specification from Planguage definition, through to user-specific definitions. The symbol indicating that we are reusing a predefined specification is the use of words with leading capital letters, for example, 'Product Development Cost'.

Define Once: One of the suggested basic formal (reusable!) rules of Planguage is that planning objects such as requirements and designs should have only one specification, which is tagged and reused whenever needed.

"R3: **Unique**: Specifications shall exist as *one official 'master' version only*. Then they shall be re-used, by cross-referencing, using their identity tag. 'Duplication' (copy and paste) should be strongly discouraged." (Gilb 2005, Section 1.4).

Process Reuse: Fundamental processes such as clear technical specification, quality control of a specification, or quantifying quality ideas, are designed to be reused in several contexts, such as in requirement or design specification.

Tailoring Aspects

One thing that makes reuse more interesting and practical, is when the reused specification can be *tailored* to adapt to the local circumstances. Of course reuse is not the only benefit with such tailoring, more accurate specification can also be achieved that better reflects the real requirements and so saves effort. Planguage gives many such options. Consider for example, the use of scale qualifiers and qualifiers.

Scale Qualifiers

For a given scale, any useful number of scale qualifiers can be defined in the scale definition. These must and can be further defined in any statements that refer to the scale (such as Past,

Goal, Meter). This tailors these particular statements to particular circumstances of interest, such as the type of customer, market, type of use of product, etc. For example, Task is a scale qualifier in the scale below.

Scale: Time to learn a defined [Task].

Scale qualifiers are generic; each scale qualifier needs to be explicitly assigned a corresponding 'scale variable' (unless a default is being used) when the scale is used in other parameter statements (such as any benchmarks or targets). For example:

Goal [Task = Setup]: 10 minutes.

'Setup' is a scale qualifier defining the Scale qualifier 'Task' that was previously undefined in the original scale definition.

The purpose of scale qualifier, and their consequent definitions, is to allow a scale specification to be more generalized and flexible; this consequently makes a scale specification more reusable and agile.

Qualifiers

Qualifiers are sets of parameters that enable tailoring of specifications. They can contain any number of interesting parameters (usually from 1 to 6), and they can be as tailored as a project needs. For example:

Goal [User = Engineer, Maturity = Novice, Task = Calculation, Market = Europe, Deadline = Release 9.0]: 60%.

The format is

<parameter Name> [<set of qualifiers>] <specification that is valid when all qualifiers are 'true'>.

The qualifiers allow much more detailed specification than we would tend normally to try to do. They invite you to specify many interesting variations. Instead of just one requirement, we end up with a set of requirements for specific contexts, that is for specific categories, localities, conditions and delivery intervals. The requirement becomes a 'curve of improvement in a multi-dimensional space'. Note the system space is described by the qualifiers. This allows projects to be divided up into many smaller evolutionary delivery steps that correspond to each specification variant, or to increments of improvement levels between such required points. This in turn directly allows the project to be far more sensitive to being effective earlier in delivering specific requirements. This directly lays the basis for more-sensitive agile reactions to any deviations from the planned trajectory.

Mid-Development Agility

Agility is about obtaining useful feedback on progress and deviation, as early and frequently as possible, and making sure that the information is acted on quickly. Specifications such as the example below help deal with 'midway progress':

Usability.Intuitiveness:

Ambition: Radical improvement in the intuitiveness of the product, compared to the existing product and competitors' products.

Scale: Percentage probability that the defined [Tasks] can be successfully completed by the defined [Users] without any reference to training, handbooks and help desks.

Past [Release 8.5, Tasks =Normal Mix, Users = Beginners, February 2005]: 30%. 'The benchmark'

Fail [Release 9.0, Tasks =Normal Mix, Users = Beginners]: 50%. 'A constraint level'

Goal [Release 9.5, Tasks =Normal Mix, Users = Beginners]: 80%. 'A target level'

To give an example, in one customer case (Johansen and Gilb 2005) when the project was midway between start and product version release, the client could measure that the project had reached about 50% of the intuitiveness requirement. So they knew they had kept within the worst case constraint (Fail level), and knew that they were on track to reach their 80% target (which they in fact did). Their website could brag "Up to 175 percent more intuitive user interface" (Confirmit 2010).

Background Specification

Numerous background requirement specifications can make a contribution to the ability of project management to see problems, to sense emerging problems, and to react to problems. For example:

Risks: R1: Lack of skilled specialists can threaten deadline.

Issues: I1: The mandatory duration of the software leasing contract can seriously impact our ability to reduce costs if the volume of sales is lower than expected.

Dependencies: D1: The software outsourcer must be able to turn around the most-critical changes within a week.

Authority: The local national authority or possibly super-national authority (such as European Union) law may restrict freedom to choose sub-suppliers.

Again this is all about capturing the necessary information in a lightweight way. The use of templates helps achieve this.

The Requirement Specification Object Database

Planguage does not think in terms of specification documents or specifications, as such. However, the requirement specifications are themselves primarily reusable objects, containing all the collected information about a requirement, in a highly organized format. Requirements (and designs and other specifications) are essentially regarded as a database of project information. We can systematically extract whatever views of the requirements we need for the purpose at hand.

Each requirement has its own set of specification management information, such as:

Type:

Version:

Specification Owner:

Specification Implementer:

Test Specifications:

Last Change Date:

Stakeholders:

These parameters essentially allow you to manage change and analysis at the level of the single requirement object. They help you know exactly who to communicate with about requirement changes when you are in a hurry.

High Level Requirements Give Agility

Planguage is especially adamant that we capture the 'real requirements'. These are the requirements *really* needed by defined stakeholders. Too many 'requirements' are actually design (the 'means') assumed to be the way to satisfy the real objectives (the 'ends') and they are often completely un-stated, or poorly defined. For example, a requirement to implement a password (a design for security) is specified, instead of a specification of how much security (the real requirement) is needed.

The key is an emphasis on quantification of all the qualitative requirements (like security, adaptability and usability) (Gilb 2005, Chapter 5, How to Quantify: Scales of Measure). Once people have learned how to quantify qualitative requirements, they can be *specific* about their requirements; and do not have to stoop to the *wrong* level of articulation (design) in order to specify their needs. This dramatically promotes agility, in that we are then free to chose and re-choose any design idea that best satisfies our quality objectives. We are not locked into the initial design ideas, falsely stated as 'requirements'.

Impact Estimation

Space does not permit a full description of Impact Estimation (IE) (Gilb 2005, Chapter 9), which is one of the main Planguage methods. However, see Figure 1 for an overview of how the method operates: it places the designs in a matrix against the system objectives and demands the designer consider how well each design meets each of the objectives. Further when the chosen de-

Key:
s = seconds
m = minutes
d = days
w = week

Bank System

By End Date: dd/mm/yyyy

Designs by expected Increment with design dependencies

Requirements	1 D1: Automate Rules + Manual Testing	2 D2: Back Office Loan Decisioning	3 D3: Web Self-Service	4 D4: Automate Rules + Automate Testing
R1: Time for customer to submit request 30 min <-> 10 min	-	-	10 m 100%	-
R2: Time for Back Office to enter request 30 min <-> 10 min	-	-	0 m 150%	-
R3: Time to respond to customer request 5 days <-> 20 seconds	-	1 d 80%	20 s 100%	-
R4: No of Back Office complaints 10 per week <-> 0	5 50%	<1 90%	0 100%	(2) (80%)
R5: No of customer complaints 25 per week <-> 5	-	15 50%	5 100%	-
R6: Time to update business rules 1 month <-> 1 day	2 w 50%	-	-	1 d 100%
R7: Time to distribute business rules 2 weeks <-> 1 day	1 d 100%	-	20 s 103%	-
Cumulative Total for Performance Requirements	200%	170%	280%	50%
Design Cost (M)	0.2	0.3	1.0	0.5
Development Budget 2.5M <-> 300K	2.3	2.0	1.0	0.5
Cumulative Perf. to Devt. Cost Ratio	1000	567	280	100

Figure 1: An example of an IE table. This shows an initial proposed set of designs, ordered by increment, and their impacts on a selected set of the system quality requirements. For requirement R1, the current time taken for a customer to submit a request is 30 minutes and the goal is to reduce this time to 10 minutes. Note the cumulative performance to development cost ratio at the bottom of the table, which measures comparative cost-effectiveness of the different designs by summing the percentage increases in impacts up to 100% and dividing by the design cost. This IE table example was developed by Lindsey Brodie.

signs are implemented, the actual results can be input and any deviations in the original estimates assessed. The designer can then reconsider the system design in the light of this feedback.

Conclusions

Planguage not only supports the principles of the agile community, it goes a step beyond by providing a method that supports effective specification and focuses on measurable result delivery. Communication is at the heart of Planguage and by capturing the system quality requirements in a measurable way, unambiguous progress can be tracked throughout a project's lifetime. ■

References

Agile Principles (2001). Available from: http://agilemanifesto.org/principles.html [Accessed 20 December 2010].

Confirmit (2010). Available from: http://www.Confirmit.com [Accessed 21 December 2010].

Gilb, T. (2004) What is missing from the conventional agile and extreme methods? Slides presented as keynote at XP Days, 2004, London.

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

Gilb, T. (2010). Planguage Glossary Concepts. Available from http://www.Gilb.com.

Johansen, T. and Gilb, T. (2005) 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. Paper presented at INCOSE, July 2005, Rochester NY. See also http://www.confirmit.com/news/release_20041129_confirmit_9.0_mr.asp

> About the author

Tom Gilb

has been an independent consultant, teacher and author, since 1960. He mainly works 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).

His other books include 'Software Inspection' co-authored with Dorothy Graham (1993), and 'Principles of Software Engineering Management' (1988). His 'Software Metrics' book (1976, Out of Print) has been cited as the initial foundation of what is now CMMI Level 4.

Tom's key interests include business metrics, evolutionary delivery, and further development of his planning language, Planguage. He is a member of INCOSE and is an active member of the Norwegian chapter, NORSEC.

Email: Tom@Gilb.com

URL: http://www.Gilb.com

Lindsey Brodie

is currently carrying out research on prioritization of stakeholder value, and teaching part-time at Middlesex University. She has an MSc in Information Systems Design from Kingston Polytechnic. Her first degree was Joint Honours Physics and Chemistry from King's College, London University. Lindsey worked in industry for many years, mainly for ICL. Initially, Lindsey worked on project teams on customer sites (including the Inland Revenue, Barclays Bank, and J. Sainsbury's) providing technical support and developing customised software for operations. From there, she progressed to product support of mainframe operating systems and data management software: databases, data dictionary and 4th generation applications. Having completed her Masters, she transferred to systems development - writing feasibility studies and user requirements specifications, before working in corporate IT strategy and business process re-engineering. Lindsey has collaborated with Tom Gilb and edited his book, "Competitive Engineering". She has also co-authored a student textbook, "Successful IT Projects" with Darren Dalcher (National Centre for Project Management). She is a member of the BCS and a Chartered IT Practitioner (CITP).