

Exploring Agile

David Gelperin
ClearSpecs Enterprises
2425 Zealand Ave. N.
Golden Valley, MN 55427
01-763-591-1534

david@clearspecs.com

ABSTRACT

Bob attempts to figure out exactly what Agile development is and where it works by analyzing the Agile Manifesto and its supporting principles. He proposes changes to some troubling statements.

Categories and Subject Descriptors

D.2.9 [Management]: Software Process Models – *Agile development, Agile Manifesto.*

General Terms

Management

Keywords

Agile development, Agile Manifesto

1. INTRODUCTION

Bob was excited. The boss had said, “We need to get agile.” and Bob had been assigned to figure out first steps. He had heard the buzz and was looking forward to finding out exactly what Agile development is and where it works.

2. AGILE VALUE PROPOSITIONS

2.1 Basic Propositions

He started with the Agile “Declaration of Independence”, the Agile Manifesto [1] and its supporting principles. In the Manifesto, he found the following four value propositions with a statement that items on the left were valued more than those on the right.

... we have come to value:

1. **Customer collaboration** over contract negotiation
2. **Individuals and interactions** over processes and tools
3. **Responding to change** over following a plan
4. **Working software** over comprehensive documentation

Bob readily agreed with the first two propositions and thought that most experienced practitioners would embrace these people-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

APSO’08, May 10, 2008, Leipzig, Germany.

Copyright 2008 ACM 978-1-60558-021-0/08/05...\$5.00.

preferencing statements regardless of development methodology. However, Bob did have questions about the other propositions – especially the last.

2.2 Responding to Change

Bob understood the third proposition as a call to be aware of change and to be willing to adjust plans accordingly. He assumed that it did not say that Agilest preferred to respond to those changes directly caused by a failure to plan rather than doing the corresponding planning.

Bob thought about a situation where a customer had a deep understanding of the needed system behaviors and their interactions. The Agile bias towards “early and continual delivery” seemed to imply that customer knowledge would be elicited incrementally. This was likely to result in unnecessary refactoring as the developer’s understanding deepened. Bob found this preference for “voluntary ignorance” unappealing until he realized that he may have found a situation outside the scope of Agile development. He would have to find out.

Bob wondered if Agilists would embrace change to their Manifesto when it did not communicate effectively.

2.3 Comprehensive Documentation

2.3.1 What is Comprehensive Documentation?

Bob did not understand the fourth proposition. The manifesto said “... there is value in the items on the right ...”. In a survey article on the manifesto, Martin Fowler and Jim Highsmith [2] say, “... comprehensive documentation is not necessarily bad ...”.

Bob was confused, so he looked up “comprehensive” and found two definitions: (1) including all or everything (2) broad in scope. The first made the definition of “comprehensive documentation” unappealing in any situation while the second left it vague. Bob’s confusion deepened. What exactly is “comprehensive documentation” and when, if ever, is it valuable? When comprehensive documentation is valuable, does it make sense to prefer working software more? When it is not valuable, preferring working software has little meaning for example ... we have come to value working software over wooden nickels.

Bob wondered if “working software” included executable tests. More questions.

2.3.2 Economical Information

To explore further, Bob tried to think of a substitute phrase naming something that clearly has value. He settled on “economical information” with the definition: **minimum information necessary for cost-effective communication.**

A document is a written account or text file containing information. It is information that is being communicated and information that has value. A document can have less value than the information it contains if valuable information is difficult to recognize for example, because of poor selection (buried in worthless information), poor expression, or poor organization.

The value of information depends on both its form and content. Poor form can significantly increase cost, decrease understanding or discourage use. Consider the specification of a complex Boolean expression in natural language. Bob knew that cost included creation and management as well as use. To be economical, the value of information must be (much) greater than its cost.

2.3.3 Economical is Context-Sensitive

Bob focused on three types of economical information: requirements, design, and usage. He realized that in all these cases, the idea of “minimum necessary information” is a context-sensitive concept involving communication with both present and future stakeholders. Sometimes, sources may have necessary, but inadequate information i.e., below the minimum.

Deciding if information is necessary and minimum requires knowledge about the experience, understanding, information needs, alternative source availability, and personal preferences of the stakeholders who receive the information. Hard-coded “magic numbers” are not magic to the coder – at least when written. Five years later, a maintenance programmer may feel differently.

Bob remembered the following illustration from a presentation on specifying requirements [4], showing how the need for written detail varies with the early understanding of the providers and receivers of the information.

Table 1. Understanding Should Determine Amount of Detail

Shallow PU	5%	Agile	Research
Medium PU	5-15%	Agile or 30-35%	Agile
Deep PU	5-25%	45-55%	70-85%
	Deep RU	Medium RU	Shallow RU

PU – Provider Understanding of problem domain and specific needs and wants for solution capabilities

RU – Receiver Understanding of solution domain and relevant solutions

% of requirements detailing that is needed

Corner-case examples:

Upper left – A merchant needs an e-commerce website and goes to an experienced company that specializes in such websites. The merchant provides details by answering questions. Little will be written. [Note that user stories are not needed.]

Lower left – A successful serial entrepreneur needs an e-commerce website for his newest company and goes to an experienced company that specializes in such websites. He can behave as the previous merchant or provide one or two of his existing sites as examples. [Note that user stories are not needed.]

Lower right – An experienced software engineer with a detailed design for a new system goes to an experienced software development company that is unfamiliar with the new system.

Even more written details may be needed to create consensus within a customer group or because of third-party testers, government regulations, or legal considerations. It is likely the Agile Manifesto itself exists, in part, to create consensus among the founding group.

All of this means the need for detailed requirements information varies widely depending on the situation and that a mandated set of requirements documentation can be economical only if it is used in similar situations.

Bob wondered how Manifesto authors would feel about

... we have come to value:

Working software over documentation or

Working software over mandated documentation or

Working software over economical information

Bob felt (1) that “comprehensive documentation” should be clearly defined or (2) the proposition should be restated to identify something that has value or (3) the proposition should be deleted.

3. SOME TROUBLING PRINCIPLES

Bob had no problem with 10 of the 12 principles behind the Agile Manifesto, but found these two troubling:

- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- The best architectures, requirements, and designs emerge from self-organizing teams.

Bob knew that principle A was not true. The dictionary definition of “conversation” only refers to speech. It does not include writing or diagrams. Information about a complex Boolean expression cannot be conveyed just by speaking.

Bob thought that principle B was also untrue. If “teams” means more than one person, the claim that “the best” architectures, requirements, and designs only emerge from self-organizing groups has many counterexamples – **Bill Atkinson** (HyperCard), **Alain Colmeraner** (Prolog), etc. While these individuals probably had help, there is no evidence that they worked in self-organizing teams. Therefore, even though one might prefer to work in self-organizing teams and these teams might produce “the best” result, the claim that this is always true is false.

Bob thought the following changes would help.

- A. The most efficient and effective method of conveying information to and within a development team is **face-to-face communication**.
- B. The best architectures, requirements, and designs **sometimes** emerge from self-organizing teams.

4. THE BOUNDARIES OF AGILE

At this point, Bob wanted to understand (1) what Agile development is and what it is not and (2) when Agile should be used and when not. Bob had already decided the first two value propositions were not unique to Agile. Neither was iterative and incremental development – a powerful risk reduction tactic usable by other development methodologies [5].

Bob decided that Agile was distinguished by:

- embracing and reacting to change, learning from mistakes, delivering customer-valued features early and often and
- minimizing planning, documentation, and analysis before code

Bob wondered if the Agile bias toward customer-valued features and a willingness (eagerness?) to refactor, would discourage “big picture” thinking by customers and developers. Was “Don’t sweat the big stuff, we can always refactor” an Agile motto?

As to when Agile should be used, Martin Fowler [3] wrote, “People who work on methodology are not very good at identifying boundary conditions: the places where the methodology passes from appropriate ... [to] inappropriate. Most methodologists want their methodologies to be ... [used] by everyone, so they don’t understand nor publicize their boundary conditions.” Bob found this to be true of Agile proponents as well as those promoting alternative approaches.

He thought Agile development would be well matched to inherently fluid situations (little understanding of needs by both customers and developers causing significant change) on entire projects or on parts of a project. He worried that an Agile bias might create artificially fluid situations with change caused by inadequate requirements elicitation, analysis, or specification.

Bob also worried that the Agile bias toward features and working software might be a problem when a product architecture needed to be defined. Many architectural choices are driven by nonfunctional requirements (safety, security, privacy, performance, robustness ...). It is not clear how Agile integrates these product-wide concerns into its approach. Microsoft discovered how expensive it is to retrofit security.

Bob wondered about using Agile on “large” products where global refactoring would grow to be uneconomical. Are there product size limits on using Agile or does Agile make assumptions about architectural stability? He would have to find out.

5. CONCLUSION

Bob was still excited to get agile, but he had questions. Bob felt like the boy who sees the emperor. It is not that the emperor has no clothes; it is just that his shirt is wrinkled and his socks don’t match.

6. ACKNOWLEDGEMENTS

Many thanks to Brian Marick for thoughtful comments on a previous draft.

7. REFERENCES

- [1] ... 2001 Manifesto for Agile Software Development agilemanifesto.org
- [2] Fowler, M and Highsmith, J 2001 The Agile Manifesto www.ddj.com/showArticle.jhtml?articleID=184414755
- [3] Fowler, M 2000 The New Methodology martinfowler.com/articles/newMethodologyOriginal.html#TheUnpredictabilityOfRequirements
- [4] Gelperin, D 2008 Just Enough Precision www.clearspecs.com [technical papers]
- [5] Larman, C and Basili, V 2003 Iterative and Incremental Development: A Brief History IEEE Computer (June 2003)