Food for Thought™

An e-newsletter published by
Software Quality Consulting, Inc.

March 2008
Vol. 5 No. 3

Welcome to **Food for Thought™**, an e-newsletter from **Software Quality Consulting**. I've created free subscriptions for my valued business contacts. If you find this newsletter informative, I encourage you to continue reading. Feel free to pass this newsletter along to colleagues by using this **Forward Email** link. If you've received this newsletter from a colleague and would like to subscribe, please use this **Enter New Subscription** link. If you don't wish to receive this newsletter, use the **SafeUnSubscribe** link at the bottom of this newsletter, and you won't be bothered again.

Your continued feedback on this newsletter is most welcome. Please send your comments to **steve@swqual.com**

In this issue

In **This Months' Topic** I discuss the need for agility in software development processes...

Regular features to look for each month are:

- **Monthly Morsels**
  Hints, tips, techniques, and references related to this month's topic
- **Calendar**
  Conferences, workshops, and meetings of interest to software engineers, QA engineers, and anyone interested in software development

This months' topic

Agile with a Lowercase "a"

The dictionary defines the word **agile** as:

1. quick and well-coordinated in movement; lithe;
2. active; lively;
3. marked by an ability to think quickly; mentally acute or aware;

Many software development organizations confuse the need to become more **agile** (with a lowercase "a") with **Agile** (with an uppercase "A"). In case you've been living under a rock, **Agile** with an uppercase "A" refers to **Agile Development Methodologies** and includes eXtreme Programming (XP), Crystal, Scrum, Lean, and others...

Before we get into a discussion of **agile vs. Agile**, it would be appropriate to answer one key question first, and that is...

What types of projects are good candidates for Agile with an uppercase "A"?

Agile Methods are particularly well-suited for IT projects where:

- software is being developed for use solely within the company
- a real, live "customer" (end user) is on the project team 24x7
- risk of failure is low
- project team is highly motivated and led by an experienced leader

For other types of projects (i.e., where software is intended for use by people outside the company), Agile Methods often aren't the best choice. Why? Because as we will see, for Agile Methods to work effectively, you need to follow **all** of the key practices associated with an Agile Method. And in many cases this is not possible for they types of projects that don't meet the criteria above.

Further, as Roger Pressman observed:

> "I contend that software engineering principles *always* work. It's never inappropriate to stress solid problem solving, good design, and thorough testing (not to mention the control of change, an emphasis on quality,..). A specific software process might fail because it is overkill, or the work products it requires are unnecessary or burdensome, or a person or team becomes overly dogmatic in the application of the process. But **history is on the side of a solid engineering approach**." [3]

And for projects that are high-risk or life-threatening, Mark Paulk said:

> "Should organizations use XP, as published, for life-critical or high-reliability systems? Probably not. XP's lack of design documentation and de-emphasis on architecture are risky." [4]

Agile with an Uppercase "A"

In talking with software development and SQA folks at many companies, the message that I have been getting is that many organizations are already using either Agile Development Methods or are planning to. There has been tremendous interest in Agile Development Methods since most businesses believe they need to become more **agile** (with a lowercase "a"). And wouldn't you know it, there just happens to be a development methodology called "Agile". However, becoming more **agile** and **using Agile** are very different...

An observation I have made is that companies who say they are using an Agile Development Methodology are often not using the methodology as it was defined in the literature. Many have changed the methodology to "suit their needs". For example, it seems that some companies who claim to be using **eXtreme Programming (XP)** aren't using the following key **XP** practices:

- Pair Programming
- On-site customer 24x7
- Collective code ownership

I'm not the only one who has seen this. Others have made this same observation. In a recent **Wiki blog** posting, **Alistair Cockburn** said:

> "Well, from my experience, most teams that say they're doing **XP** don't actually do the practices."

And, Matt Stephens observed that:

> "As **XP** increases in popularity and hits the mainstream, more and more teams will attempt **XP**, probably without a clear understanding of what is really involved. They will most likely be drawn in by **XP**'s 'low discipline' practices (such as no big up-front design and minimal documentation), but without applying the high discipline practices that act as an essential safety net (such as unit testing, pair programming, collective ownership and constant refactoring)." [2]

Let's review the list of **XP** Practices as defined by **Kent Beck** [1]

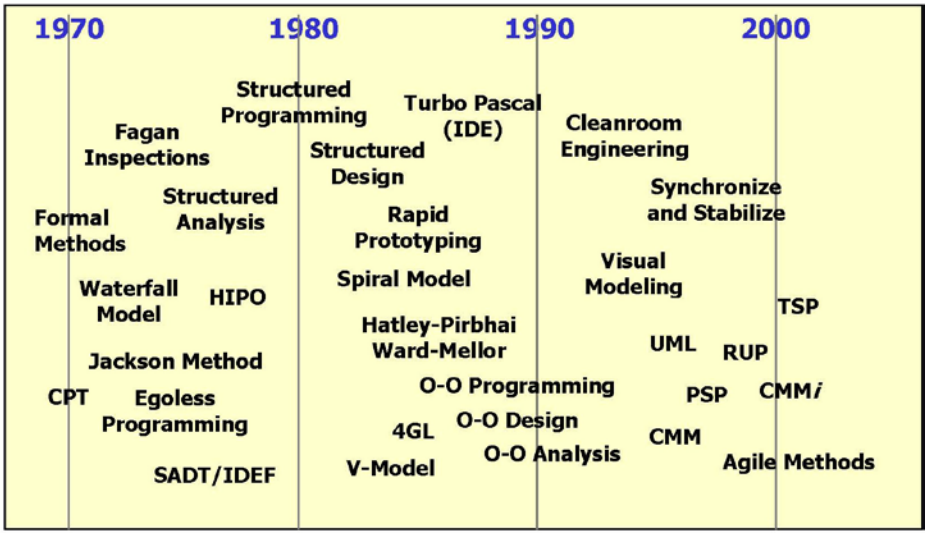
Planning	Quickly determine scope of next release by combining business priorities and technical estimates. As reality overtakes plan, update plan.
Small releases	System into production quickly, then release new versions on short cycle.
Metaphor	A simple shared story of how the whole system works.

Simple design	Extra complexity removed when discovered.
Testing	Developers continually write unit tests, which must run for development to continue. Customers write tests for features.
Refactoring	Restructure system without changing its behavior to remove duplication, simplify, etc.
Pair Programming	All code is written with two developers at one computer.
Collective ownership	Anyone can change any code anywhere in the system at any time.
Continuous integration	Integrate and build the system many times a day, every time a task is completed.
40-hour week	Work no more than 40 hours per week as a rule. No overtime two weeks in a row.
On-site customer	Include real customer on team 24X7.
Coding Standards	Developers agree to standards.

Some obvious questions are:

- Can you claim to be following **eXtreme Programming** if you're not doing **ALL** of the practices above?
- If you're not doing **ALL** of the practices above, can you expect to see the same benefits from those that do use them?

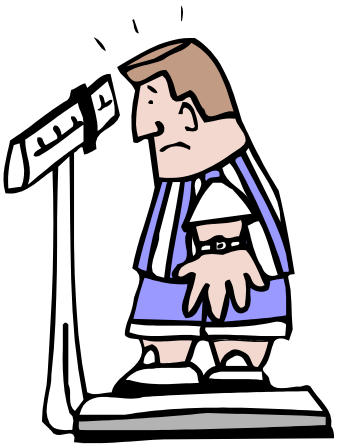
Let's look at some examples of other software development methodologies that have identified key practices:



In my experience with many of the software development methodologies shown above, you need to follow **ALL** the individual practices of a given methodology in order to see benefit.

This is a lot like choosing a diet. If you pick a diet and follow **ALL** the rules for that diet, you'll usually lose weight. If you're like me and choose to ignore some of the rules, you usually won't lose weight.

To derive benefit from Agile Methods, you need to follow them as they were defined. You can't pick and choose which of the key practices to use and which to ignore.



Agile with a Lowercase "a"

There is a basic principle that businesses of most every type need to follow to be successful - reduce costs and increase efficiency. This principle can lead to improvements in your bottom line.

If you routinely work on projects that don't meet the criteria for Agile Methods defined above, there are several ways that you can improve the effectiveness of your development process, and as a result, become more **agile**.

For inspiration, I recommend reading the Poppendieck's book [6] on Implementing Lean Software Development. The concepts in their book are excellent examples of how to become more **agile** (with a lowercase "a") without all of the hoopla of **Agile** (with an uppercase "A").

For example, their Seven Principles of Lean Software Development are derived from the lean manufacturing practices developed by Toyota. I find these principles to be extremely valuable in assessing effectiveness, even though I don't always agree with the Poppendieck's interpretation of these principles.

Let's look at the Seven Principles [5]

- **Principle 1 - Eliminate Waste**

  In Lean Manufacturing, waste is broadly defined as anything that doesn't add value. For software development, I define waste as anything that **doesn't contribute to meeting the needs of customers**. We need to look at all of the tasks, activities, artifacts and other things that happen on projects and determine which things can be eliminated based on this definition of waste.

- **Principle 2 - Build Quality In**

  Build Quality In means getting it right the first time and focusing on what it is that provides value to your customers. You want a process that helps prevent defects from creeping into the code. How can you achieve this? The source of most defects is poorly written, ambiguous requirements. Removing ambiguity from your requirements can go a long way to getting it right the first time and building quality in...

  Learn more about writing better requirements...

  Sometimes we can build quality in by detecting defects earlier in the development process. To do this requires effective peer reviews and inspections.

  Learn more about peer reviews and inspections...

- **Principle 3 - Create Knowledge**

  Creating knowledge is all about learning and sharing information. For software development, this can be done by following a few key recommendations [6]:

  - Release a small subset of key features early for review and evaluation
  - Perform daily builds and provide rapid feedback from on-going testing
  - Find a project/team leader with experience and instincts to make good decisions
  - Use a modular architecture that enables features to be added

more easily

"It is important to have a development process that encourages systematic learning throughout the development cycle, but we also need to systematically improve that development process." [5]

- **Principle 4 - Defer Commitment**

  The principle here is that the more information we have, the better able we are to make an informed decision. The Poppendiecks recommend deferring irreversible decisions until the last possible moment so that you have the most information available to make that decision.

  I have found an estimating and scheduling technique that can help accommodate this principle. It's called the Yellow Sticky Method...

  Learn more about the Yellow Sticky Method and estimating and scheduling best practices...

- **Principle 5 - Deliver Fast**

  "Companies that compete on the basis of time have a huge advantage over their competitors: they have eliminated a huge amount of waste and waste costs money." [5]

  The ability to deliver fast is, in my opinion, determined by how predictable your organization is. Unpredictable organizations tend to be much slower in delivering because there is much more waste and confusion. Predictable organizations tend to be well-organized and led by experienced managers who know what needs to get done and how to make it happen.

  Learn about helping your organization become more predictable...

  Attend a Predictable Software Development workshop...

- **Principle 6 - Respect People**

  I often coach managers in management skills and respecting people is a key topic that I cover. As a manager, you need to trust your people to make good decisions. You can't undermine them and you can't think for them.

  Read more about managing, coaching and mentoring...

- **Principle 7 - Optimize the Whole**

  When we look at software development practices, we often tend to micromanage and focus on the parts rather than the whole. To improve software development, we need to take a holistic view and focus on the whole process, not just the parts.

  One really good way to do this is via a Project Retrospective. This activity can help identify where problems are and how they can be addressed from a holistic perspective...

  Learn more about Project Retrospectives...

Summary

Becoming **more agile** is critical. Eliminating waste and improving efficiency are essential to survive in today's global economy.

You don't need to follow an **Agile Development Method** to become **more agile** especially if your projects are not IT software developed for use in-house. There are many ways any software development method can be improved to minimize waste

and improve efficiency. Becoming **more agile** should be the goal regardless of the software development methodology you use...

'Til next time...

Every month in this space you'll find additional information related to this month's topic.

- **References**

1. Beck, K., Extreme Programming Explained, Addison-Wesley, 2000.

2. Stephens, M. and Rosenberg, D., Extreme Programming Refactored: The Case Against XP, Apress, 2003.

3. Pressman, R., "What A Tangled Web We Weave", *IEEE Software*, Jan/Feb 2000, pp. 18-21.

4. Paulk, M., "Extreme Programming from a CMM Perspective", *IEEE Software* Nov/Dec 2001, p. 19-26.

5. Poppendieck, M. and Poppendieck, T., Implementing Lean Software Development - From Concept to Cash, Addison-Wesley, 2007.

6. McCormack, A., "Product Development Practices that Work: How Internet Companies Build Software", *MIT Sloan Management Review,* Winter 2001, Vol. 40, No. 2.

Every month you'll find news here about local and national events that are of interest to the software community …

- **Software Quality Calendar**

  There are many organizations that sponsor monthly meetings, workshops, and conferences of interest to software professionals. **Find out what's happening...**

- **Workshops Offered by Software Quality Consulting**

  Software Quality Consulting offers workshops in many topics related to software process improvement. **Get more info...**

Software Quality Consulting provides consulting, training, and auditing services tailored to meet the specific needs of clients. We help clients fine-tune their software development processes and improve the quality of their software products. The overall goal is to help clients achieve Predictable Software Development™ – so that organizations can consistently deliver quality software with promised features in the promised timeframe.

To learn more about how we can help your organization, **visit our web site** or **send us an email**.