Why Agile? And Why it is Not a Silver Bullet

Agile: nimble; able to move quickly and easily
(Webster's New College Dictionary)

Is agile for you? Is it for your company? In this paper, we will first summarize agile, and its principles, benefits, and processes. Then we will review some of the challenges faced when going agile. Agile is not a magic solution. There are no silver agile bullets so with anything new, a frank and objective presentation is essential.

Think about it – you are about to drive from New York City to San Francisco. Do you:

A – Plan the entire trip, including deciding the exact route and make hotel reservations for each night, assuming that you will drive about 700 miles per day? **OR**

B – Plan the first part of the trip, say between New York and Pittsburgh? And, worry about the next leg when you get to western Pennsylvania knowing that there are always hotels to be found or that your sight-seeing priorities may change?

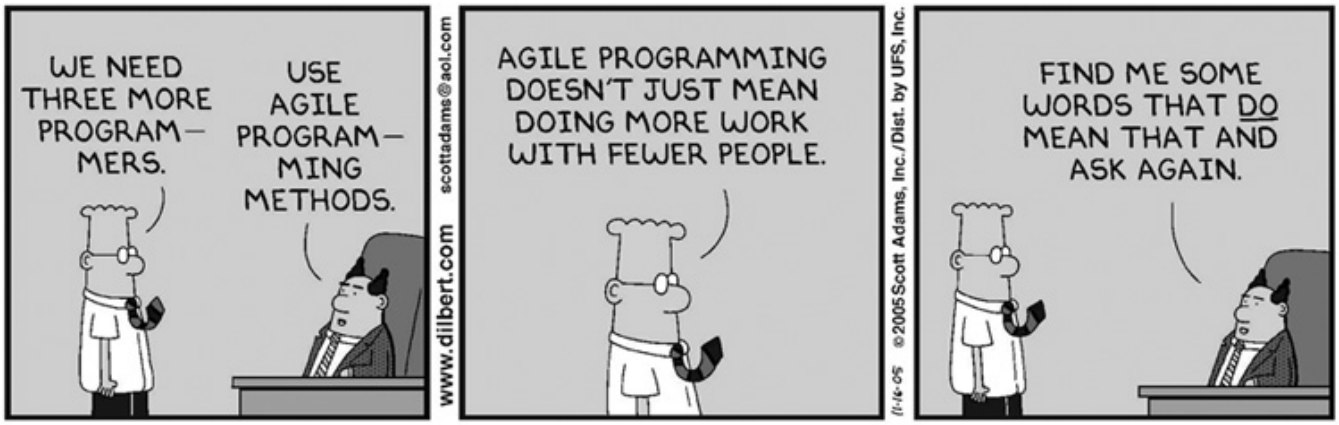
The vision in both scenarios remains the same: to get to San Francisco. But, in the B scenario, you have many interim goals and you can achieve them individually.

If you chose B, then you may be agile already. You can deal with "chunks" at a time, and want to see quicker feedback and results. You want to be a bit flexible in case your plans change. You don't want to wait for one big event. If you chose A, then your initial agile experience may challenge your assumptions about your own behavior and perspective of achievements.

WHAT IS AGILE?

What have you heard about agile? Have you heard anything that concerns you? If you were asked if your organization should use an agile process, what would your response be?

First, some thoughts from Dilbert about agile preconceptions:



Agile may be a straightforward concept, but everything about it is different as compared with the traditional approach to software development. "What do we need to build?" is still the hardest and most challenging problem faced by all software development teams. "When will you be done?" is equally challenging. But requirements, development, and project management in an agile world is fundamentally different.

Many companies have been trying to determine the best and most efficient way to develop software. As applications / systems / software products have become larger and more complex, the challenge to deliver on time has also become harder. Organizations are experimenting with project management offices (PMOs), eliminating project managers, asking business analysts to play a dual role, and other variations in an effort to deliver faster.

Many have felt that the effort of documenting complete requirements, prior to beginning development, was inefficient. After all, no *one* user knew *all* the details of what they wanted, business priorities more often than not changed during development, and the length of time it took to gather and document complete requirements was becoming longer and longer.

Past wisdom proposed that the longer we spent on analyzing requirements, the less time we needed to spend on development and testing, the less time we needed to spend on managing change. Unfortunately this never became the reality.

In the last 10 years or so, many in IT and software engineering advocated that a more agile and "leaner" approach to software development – still focused on defining "what to build" – was necessary. The agile method is now a significant factor in software development, challenging many of the assumptions that had been held sacred in traditional methodologies.

Iterative, spiral, and RAD (rapid application development from James Martin) methodologies were early precursors to the agile process. They attempted to move away from the heavy reliance on the requirements specification / waterfall approach. Agile has challenged many other practices included in project management and testing methods.

Agile software development methodologies arose out of the recognition that the development process needed to be more adept at delivering value faster, and without the overhead of anticipating and documenting *all* the requirements up front. In 2001, many of those involved with exploring different methods – scrum, extreme programming (XP), RUP, lean – documented their beliefs in an Agile Manifesto and set the scene for identifying the principles for creating software

THE AGILE PRINCIPLES

Early agile adapters reminded us of the success many manufacturing companies enjoyed using lean (or just-in-time) processes to improve return on investment, and deliver value as early in the product development process as possible.

The seven agile principles of software development, introduced by Mary Poppendieck, are based on 1990's Japanese automobile manufacturing practices. They are now a proven set of principles that organizations can use to adapt lean tools and techniques to their own software development efforts. (Poppendieck, 2006)

1. Eliminate waste

- Remove non-value added "things" and processes
- First recognize waste, then eliminate it!
  - Requirements change "churn"
  - Test and fix cycles
  - Extra or low priority features – users asking for too many requirements for fear of not getting anything of value at all
  - Keep the code and the architecture simple

2. Build quality in

- Incorporate testing strategies to find defects early and fast
- Perform continuous system integration

3. Create (product) knowledge

- Create daily builds for rapid feedback from the test team
- Release early and frequently to elicit customer feedback

- Hold demos and retrospectives to solidify product knowledge

4. Defer commitment

- Take irreversible decisions at the last minute when most information is available
- Planning is essential / but be flexible with the plan

5. Deliver fast

- Focus on providing value as soon as possible, even if it is piecemeal rather than in a "big bang" release
- Delivering fast eliminates waste (as waste costs money) and provides a competitive advantage
- Make faster decisions through empowered teams
- Continuous process improvement via the agile "retrospective" will help deliver faster in the next iteration

6. Respect people

- There is no "one best way" to do things
- Do what's right for the context of the iteration / product
- Ask the team for suggestions / empower them to make decisions

7. Optimize the whole

- See the end-to-end system and make it better, not just parts of it
- Take a "big picture" approach – see the forest through the trees

AGILE BENEFITS

Now we will take each agile principle and express it as a benefit. The value to our organization will become readily apparent.

- **Agile adds business value more quickly** because development and implementation is done in iterations. New "chunks" of functionality can be available to users faster and more frequently. Return on investment can be achieved more quickly.
- **Agile creates better products** by focusing on the highest value features. More frequent planning and prioritization allows users to get what they want. Research has shown that 43% of software features are never used, and 19% are rarely used. (Chaos Group Study, 2002)
- **Agile projects have a higher rate of customer satisfaction** because users are directly involved with planning and decision making. They provide critical feedback at key points to ensure what is being built is what is needed.
- **Agile can improve team performance** because teams are smaller and collaborative. The agile project management approach ensures constant communication and faster, transparent decision making.
- **Agile minimizes risk** as the agile development horizon is short and users provide more frequent feedback. Without the need to spec *all* the requirements in advance, *the cost of change is minimal*.

Beware, there are no silver bullets. Agile projects won't necessarily need fewer people to do more. There is no longer an imposed methodology to hide behind. There are no longer the excuses of "we need the requirements spec to start development" or "changing requirements caused us to deliver late." Agile exposes poor performing teams and lack of management buy-in and support that is often indicative of project failure. But more about this later.

HOW IS AGILE DIFFERENT?

What does this all mean? And how is agile different? Many books and web sites document the differences and details, so the table on the following page is just a generalized, high-level summary.

Dimension	*Traditional Methodologies*	*Agile*
Driving force	The end product or deliverable.	Time is the only driver for the iteration and release.
Time horizon	The time line is as long as the project.	Short – typically a few iterations / 2 – 4 months.
Culture	Command and control, top down management.	Collaborative, empowered, and team based.
Key roles	Project manager.	Agile project manager (aka scrum master) and product owner (aka product champion).
Project management	Formal and typically driven by weekly meetings and status report.	Daily and collaborative. Less formal, more visual reporting.
Requirements	All the project requirements are assumed to be known.	The requirements take the form of short "user stories" – focused on one piece of functionality that can provide value. They are not assumed to be known in detail.
Documentation	Requirements documentation is written, complete, and detailed.	User stories are documented on an as needed basis, based on their priority as determined by the product owner.
Development	Based on the requirements document; develop the whole.	Collaborative, frequent builds, one iteration at a time.
Testing	"Hand off" to test. Defects are found late in the process.	Test driven development – test as you go so that defects are found quickly, before considerable development investment is made.
Scope management	A separate and formal project management process. Sometimes change is viewed as a negative rather than a positive to meet customer needs.	Scope change is expected in order to meet customer needs. Since less is planned, less needs to be changed after the fact.

REQUIREMENTS IN AN AGILE WORLD

"The hardest single part of building a software system is deciding precisely what to build." (Brooks,1995)

This has not changed in an agile world. Deciding what to build is still a challenge. Ed Yourdon characterized software development as a "rock problem." (Leffingwell, 2003) A customer asks you to deliver a rock. But when you actually deliver a rock, the customer says that what she really wanted was a small, blue rock. But, it turns out that the customer was imagining a small, blue marble. Some similarities, but some critical differences.

Dilbert, as usual, says it very well:

Requirements in agile are top down and iterative. They start with a vision, ensconced in an "epic." Product features, once identified, are decomposed into "user stories" that identify small pieces of functionality that are valued by users. User stories are the requirements.

User stories are written so that functionality is understood by the entire team – users, designers, developers, and testers. They are key to ensuring that the solution is aligned with user expectations. User stories can be developed in a period of days or weeks, and are short and easy to read. They are often documented one per "card" to be managed, tracked, and prioritized. They also provide value that is easily recognizable to the team, can be used to estimate the work effort, and are self contained to support testing and documentation.

User stories are not requirements as we traditionally know them and they are not the detailed specifications found in typical requirements documents often taking the form "the system shall." They are not a complete portrayal of an entire system or product. They are not detailed in a defined phase of a project, but are created in a just-in-time manner to respond to the iteration demands, business needs and user priorities.

User stories take the form of:

As a <role> I can <activity> so that <business value>.

- **Role** – who performs the action or receives value from the activity.
- **Activity** – the action or functionality performed by the system.
- **Business value** – the value to the business; the "why" the activity needs to be performed.

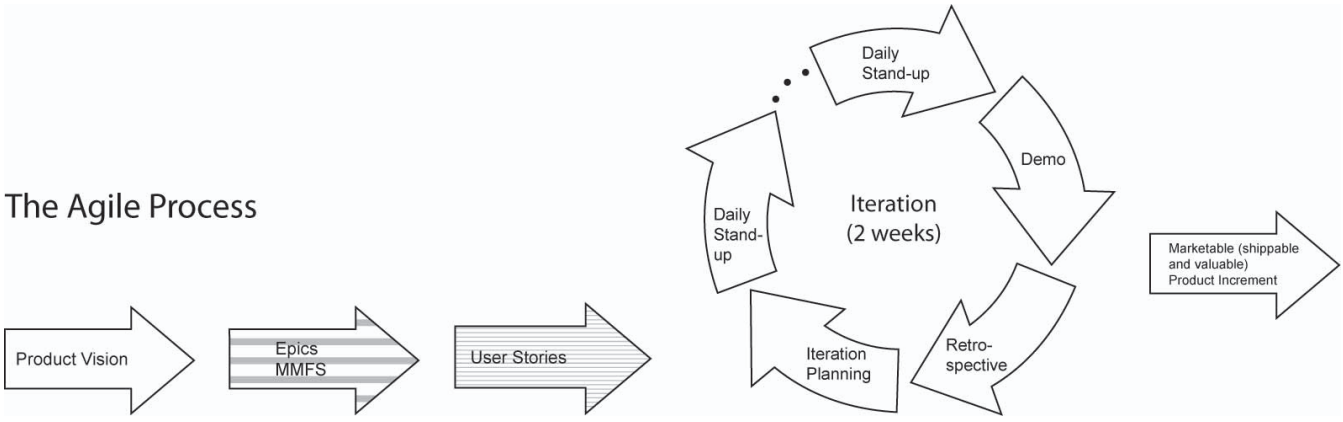
PLANNING AND MANAGING IN AN AGILE WORLD

"Management is the art of getting things done through people." (Follett, 1942)

Agile is characterized by:

- iterations lasting 2 to 4 weeks based on the scope of user stories
- short planning timeframes – detailed planning is done one iteration at a time
- daily meetings to plan, react and communicate
- estimates based on actual performance in pervious iterations
- frequent and required user interaction for feedback
- continuous process improvement applied to the next iteration.

The graphic below illustrates the release planning and iteration planning efforts. As can be seen, user stories form the critical input into each iteration. It is still about what to build! The demo and the retrospective ensure that users are involved, and knowledge is created.



AGILE CHALLENGES

Agile is a big change; agile is a culture change; agile is a paradigm shift. Each of the key stakeholders – management, the "customer," and the team – face challenges as agile is introduced. Thes challenges are listed here not to scare you off, but to make you aware of the realities of change.

Management & Organizational Challenges

Lack of executive support – management must understand and support the significant cultural and organizational changes that agile brings:

- time based "mini-projects"
- shorter horizon for project planning and commitment
- localized decision making
- "good enough" documentation
- informal but visible status reporting

Organizational problems – agile cannot solve strategic, operational or structural problems. Agile requires team work, quick and frequent decision making, and clear priorities in the short to medium term. Agile can expose these shortcomings, but will fail if they are not addressed.

The "mini-waterfall" (Crisipin & Gregory, 2010) – organizational culture wants to impose the more methodical waterfall phases on each iteration, and expects status reports to reflect the same. This will add unnecessary overhead and waste.

Project Challenges

The right project – there is considerable debate about whether agile can be for every project. Some would say a resounding "yes," others say that agile only works for small projects. Careful consideration, planning, and scoping would be recommended for projects such as the ones listed below:

- Very large, complex, enterprise-wide projects
- Infrastructure / systems architecture / data architecture projects
- Projects in which contractor pre-approval of specifications are required

In many of these projects, once the planning and analysis is complete, development, testing and implementation may be able to follow a more agile approach.

Customer Challenges

Impatience without a road map – management may ask, "when are you going to be done," and customers may ask, "what is the roadmap for next year?" The horizon in agile becomes more blurry the further out one looks. The immediate future – say one to two months looks clear, as if under a microscope. Beyond that ...

Lack of user involvement – the need for users to be involved with product development has not lessened since the original 1994 Chaos study indicated that it was a key success factor. Customers who say "just do it" or "you know what I mean" will not make an agile project any more successful than a traditional project.

Vision – the need for a clear product or release vision has not gone away. The customer still has to be able and available to help the team understand the business problem and priorities of the iteration and its user stories.

Team Challenges

Teams are not empowered – the agile team needs to be empowered to succeed and must be allowed to make decisions and move forward with minimal guidance and leadership from above. They cannot be micromanaged or hide behind a methodology and its constraints.

Poor product owner decision making – the product owner must take responsibility for product direction and requirements, or there is a risk that the team will not create the best product. A product owner who changes focus too often will lead the team to waste time, waste effort, and delay the release.

Non-performing teams are exposed – if a team is not performing well, agile will exacerbate the problem. Agile requires team work, constant communication and an open and transparent work style. Agile cannot solve team problems.

SO - WHAT TO DO?

Challenge assumptions. (Leffingwell, 2007) Challenge the "but that's the way we've always done it." Challenge the fact that requirements must not change once signed off. Challenge testing practices. Challenge the invisible walls and barriers created by the more traditional development process. Embrace change. Agile creates a whole new set of assumptions in order to deliver the promised benefits. Dream big, but deliver small things, more frequently, and on time and within budget.

REFRENCES

- Brooks, James F. (1995) *The Mythical Man-Month Anniversary Edition*. (Originially published in 1975). New York: Addison-Wesley
- Cohn, Mike (2004) *User Stories Applied: For Agile Software Development*. Boston, MA: Addison Wesley.
- Crispin, Lisa & Gregory, Janet (2010) *Agile Projects: 6 Ways to Avoid the "Mini-Waterfall"*. Software Test & Performance, January 2010
- Leffingwell, Dean & Widrig, Don (2003) *Managing Software Requirements*. New York: Addison-Wesley.
- Leffingwell, Dean (2007) *Scaling Software Agility*. New York: Addison-Wesley.
- Metcalf, H. & Urwick, L.F. editors (1942) *Dynamic Administration: The Collected Papers of May Parker Follett*. New York: Harper & Row,
- Poppendieck, Mary & Poppendieck, Tom (2006) *Implementing Lean Software Development – From Concept to Cash*. New York: Addison Wesley
- http://www.agilemanifesto.org/

Sys·tem·a'·tion®

Tel 800-747-9783 Fax 303-756-2211 www.systemation.com