

Architectural manifesto: Adopting agile development, Part 1

Is agile development right for your organization?

Skill Level: Introductory

[Mikko Kontio \(mikko.kontio@softera.fi\)](mailto:mikko.kontio@softera.fi)

Director
Softera Oy

18 Mar 2008

Mikko Kontio is back with his *Architectural manifesto* column. Learn how an organization can move toward using agile processes and about issues related to the resulting changes. In this first article on the topic, find out what agile processes are, the benefits of using them, and the requirements placed on the organization that implements them. Next month, Part 2 will discuss the use of agile processes in different kinds of companies, including old and new, and how small and large projects affect the customer and seller experience.

Introduction

As agile development has grown in popularity over the past few years, many companies are considering how to take advantage of agile development processes and techniques. Though I'm not an agile evangelist, agile development techniques have proven to be beneficial in certain types of organizations and projects.

In development processes and process development-related issues, agility has become a hot topic, because:

- Businesses want to be able to react faster to market changes.
- IT departments want to deliver issues without the formal (or normal) six-month release cycle.

- Developers like a steady development environment where they can fully concentrate on the features and quality of the software they're building.

Agile development is not for all organizations, projects, or clients. There are certain criteria that make agile development fit some organizations better than others. And, as always, it's the people that make the process happen (and people come in all types).

In this column I'll discuss the criteria to help you evaluate whether agile development suits your organization. Learn what the benefits of agile development are and the "people" aspect of the agile development process.

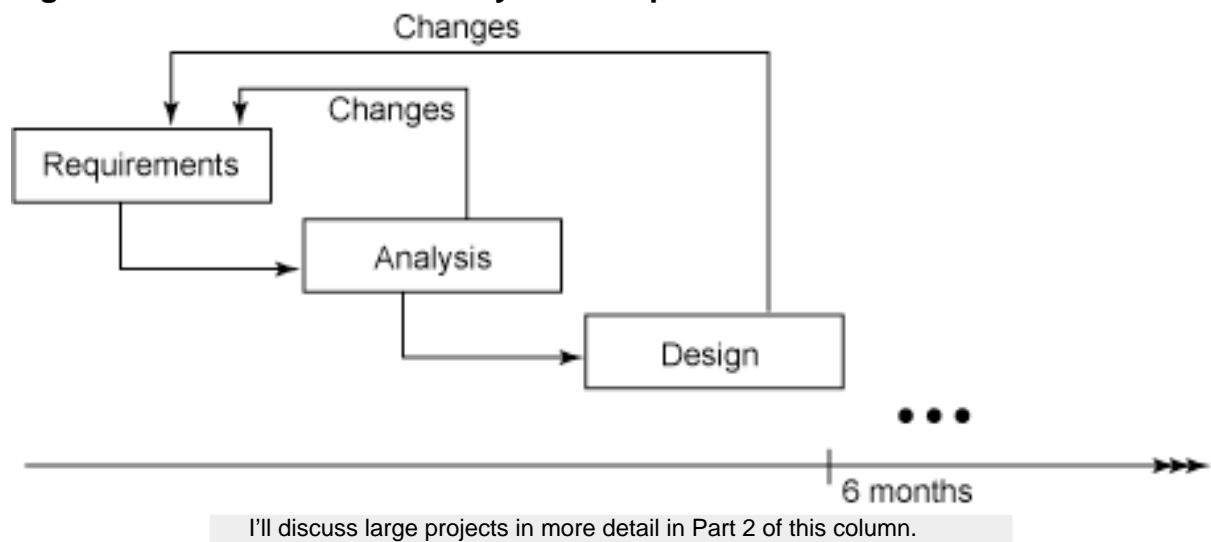
The meaning of agile

Let's first define what we mean by the word "agile." There are many, varying agile development methods. They all focus on face-to-face communication and fast release cycles, and they aim to deliver a working piece of software in short time periods. Agile methods are an attempt to ease bureaucracy in the development process to release working software that implements the customer's most important requirements. Some might say that agile development is the sound of reason in the middle of the overly bureaucratic, documented development process, but that is not entirely accurate. It all depends on the circumstances.

Contrary to some views, agile developers are not mavericks, writing code with no rules or restrictions. "Cowboy coding" is a sign of lack of discipline and poor management and is unprofessional. If there are people cowboy coding in your company, or even worse—on your team—do everything you can to change that for the sake of your customers.

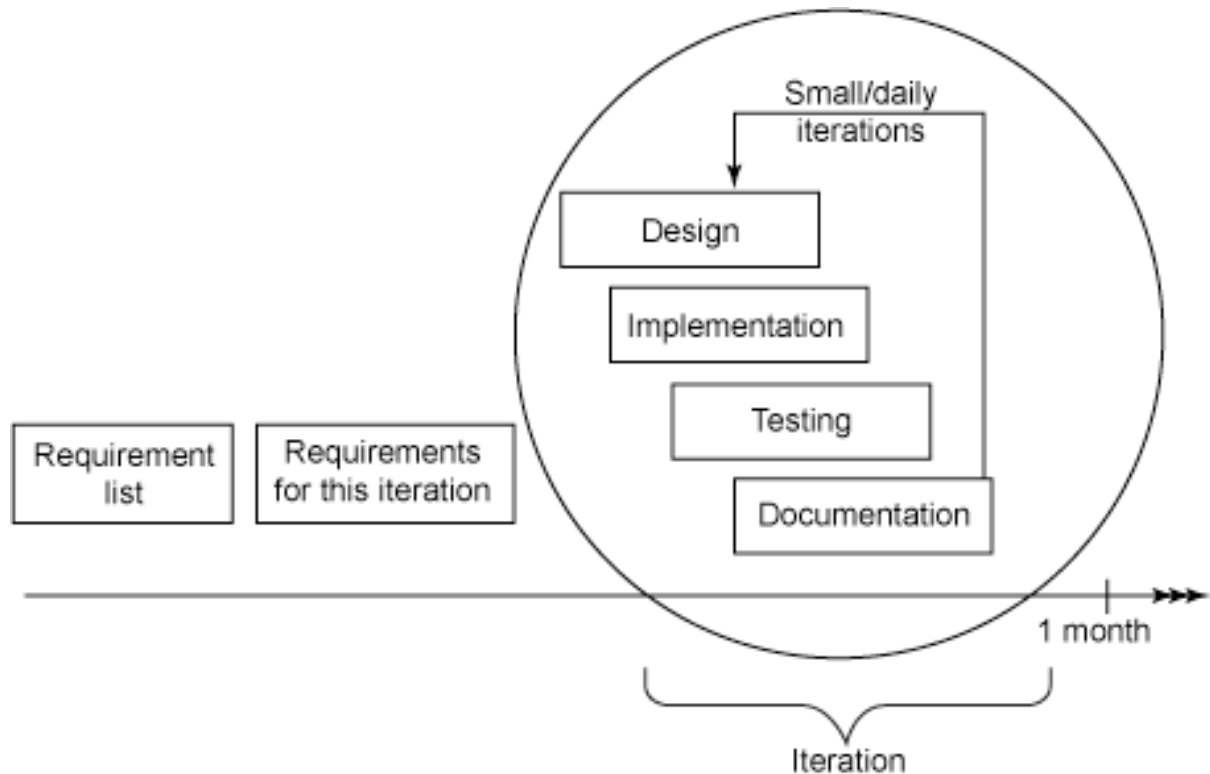
Agile development is often described with the typical story of a two-year project developed by a small project team (say five people). If the team were to use a traditional waterfall-style method, as shown in Figure 1, it would take two to three months to record all the requirements in detail. Then the team would analyze the requirements. The requirements analysis would be followed by system design, including architecture. By this time, the customer would have introduced the first changes, which would enter the change management process. The change management process would itself require a small rework of requirements, another analysis, and possibly more designs.

After six or seven months, the team would finally be ready to implement. As you might guess, the customer would need additional changes that would go through the change management process again. Now don't get me wrong—there are situations where this is probably the only possible process—such as for mission-critical or life-supporting systems.

Figure 1. Traditional waterfall-style development

If the team in the example had been using an agile approach, as shown in Figure 2, it would have roughly sketched out the requirements, but in a way that allowed the project owners to prioritize them. Then the project owners and developers together would have selected a small group of features from the requirements and designed and implemented them in a series of iterations lasting two to three weeks each. With a couple of iterations, they would have developed a working, running version of the application, and the project owners could have started trying or using the application. After a couple more iterations, the application would have had its basic functions and be ready to launch.

Figure 2. First iteration in an agile development process



The agile method depicted in Figure 2 produced a working version of the application faster than the waterfall-style method was able to produce just the application design. The working application contains only the basic features, but it still is a working version. It is helpful to have a working version—albeit one with only the basic features—so you can start using an internal application to make business processes faster, or release a beta version of a commercial service to start luring customers.

The next section discusses the benefits of using agile processes.

Principles and benefits

If your organization will not reap any benefits from agile methods, then there's no point in using them. The benefits are easiest to discuss by listing the agile principles, as follows:

Rapid, continuous delivery

Gaining customer satisfaction with rapid, continuous delivery of useful software. Is this critical to your organization? Is your company a start-up that would like to start luring customers with a beta version of an application? Will your application save internal expenses by replacing manual work? If so, you may benefit from agile development.

Frequent delivery

Working software can be delivered frequently, in weeks rather than months. If your application is a Web application, you might want to push frequent updates to add new features, or improve the application as you get feedback from customers. You don't have to worry about heavy version control, or maintain files to track which client has which version.

If your version release means changes or work at the client side, you might not want to make updates frequently. Still, frequent iterations might be a good idea, because you know that you can implement and release changes in weeks rather than months.

Working software

The principal measure of progress is working software. Written documents and slideware aren't enough to meet most business requirements—you need working software for that.

If you are in consulting, then perhaps documents and slides are enough, but deploying working software is eventually the goal for most organizations.

Accommodation

Even late changes in requirements are welcome in the agile development method. For a long time, software professionals tried to do all they could to make late changes impossible or very expensive. However, because business environments can change quickly, software requirements ought to follow.

Close, daily cooperation

Business people and software developers should have daily exchanges and cooperate on the solution. Late changes in requirements may come from the business people, and developers should implement the requirements. Daily cooperation is necessary if the process allows for requirements to change. For applications that implement interfaces or specifications, the requirements are the same as the specification document published by a designated authority. Changes to this document are more than a big deal, and they don't just appear.

Motivated, skilled people

Projects are built around motivated, skilled individuals who are trusted. (That should really be the basis of any organization.) I could easily write another column about why certain people are motivated and others are not.

Do you have the resources to motivate and train workers who are unmotivated and unskilled, or do you need to find people who are already motivated and highly skilled to hire?

Self-organizing teams

Self-organizing teams are not a reality in most software development efforts. They require a lot of experience on the development side and from management. A self-organizing team will decide what portion of requirements they can implement at a certain iteration and will decide who does what. The roles of the team members are based on their interests and knowledge, rather than being dictated by management.

A poorly organized team will accept only a few of the requirements and not produce much. In order to work properly, the team has to understand what they are doing, and the management has to trust them.

Self-organizing teams should be evaluated against other teams, not against vague estimates of the work that is required. Does your organization have skilled people? Do you feel, as a team member or as a manager, apprehensive about anything getting done?

At this point, you might be wondering if agile principles, or a self-organizing team, could work in your organization. That uncertainty leads to the next big question.

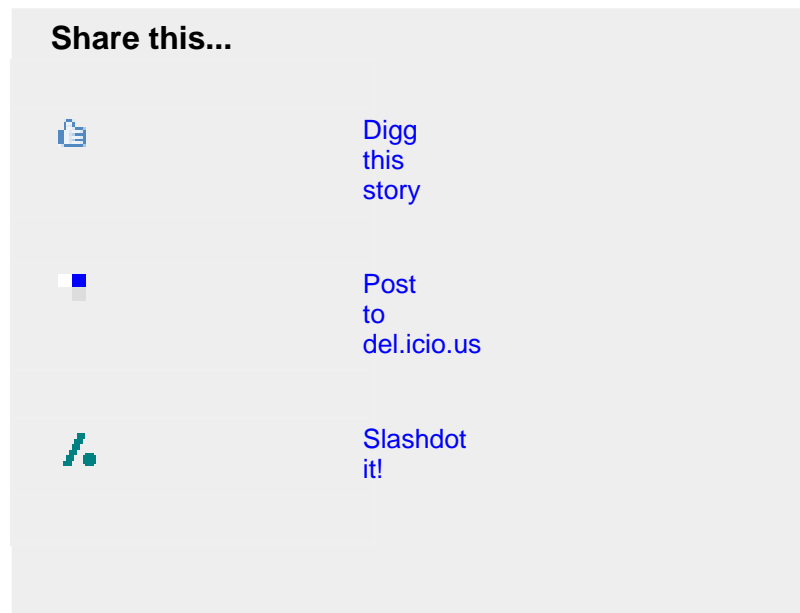
Is your organization ready for an agile process?

Adapting to agile methods is easier in organizations that have certain characteristics. Originally noted by [Cohen et al](#), those characteristics are:

- A culture of negotiation.
Open and honest discussion is important in any organization, but if you're planning to adopt agile methods, the different parts of your organization must communicate well and be able to compromise when necessary.
- Trust among the people who work there.
If management doesn't trust developers or developers don't trust sales people, you're in trouble.
- Smaller staff, with higher levels of competency.
You can get a lot done with only a handful of very good developers who don't have to deal with extra bureaucracy.
- An environment that facilitates rapid communication among team members.
Business requirements need to be met now, not next week. Your organizational culture needs to be one of rapid response, rather than one that gets mired in process.

- Small project-team size (fewer than 20 or 30 people).
As the size grows, face-to-face communication becomes more difficult.

Some years ago, I worked in a company where we were applying a simplified version of IBM Rational® Unified Process (RUP®). The characteristics of the process were quite agile, though we didn't call it agile. I was talking about iterations, how they were used, and why they were good. For some reason, the people weren't really responsive. A project manager later explained that people interpreted "iteration" to mean doing the same thing over and over again because the requirements were unclear. Their previous experiences caused them to shut their ears to what I was saying. I learned that no matter how prepared you are, it's the people who implement the processes, and people have history that is worth knowing.



Stay tuned

Next time I'll discuss agile development methods in different kinds of organizations, such as start-ups, in-house development outfits, and large companies. Small and large projects will be covered. I'll also look at agile development from the customer's point of view, cases where the development is bought outside, and how fixed-price and hourly projects affect the deal.

Resources

Learn

- "[An Introduction to Agile Methods](#)" (David Cohen, Mikael Lindvall, Patricia Costa, 2004) discusses what it means to be agile, the role of management, popular agile methods, and guidelines for deciding where an agile approach is applicable.
- Read Wikipedia's discussion on [Agile software development](#).
- Learn about [Scrum](#), an agile process that can be used to manage and control complex software and product development using iterative, incremental practices.
- "[Why Fixed Bids Are Bad for Clients](#)" (ScrumAlliance, Sep 2007) discusses agile projects and fixed-price bids.
- "[OpenUP In a Nutshell](#)" (developerWorks, Sep 2007) explores OpenUP, a recently developed process framework for software development that focuses on agile practices derived from the Rational Unified Process.
- Read "[Effective agile delivery toward globalization](#)" (developerWorks, Oct 2007) if you have overseas markets that require considerable planning in the development life cycle to accommodate cultural and language differences.
- [developerWorks Interviews: Scott Ambler on Agile development](#) (developerWorks, Apr 2007) explains this iterative and incremental approach to development, lays out the business case for it, and dispels some myths in the process.
- Make use of [IBM on demand demos](#) to learn about various software products and technologies from IBM.
- Stay current with [developerWorks Live! webcasts](#).
- Check out [developerWorks Live! technical events and briefings](#).
- Browse the [technology bookstore](#) for books on these and other technical topics.

Get products and technologies

- Download [IBM product evaluation versions](#) and get your hands on application development tools and middleware products from IBM DB2®, IBM Lotus®, IBM Rational®, IBM Tivoli®, and IBM WebSphere®.

Discuss

- Read what everyone is talking about in [developerWorks blogs](#) and get involved in the [developerWorks community](#).

About the author

Mikko Kontio

Mikko Kontio has a background in software development and consulting. He is currently a director in Softera, a software development company focusing on business portals and telecom-billing solutions.

Trademarks

IBM, the IBM logo, DB2, Lotus, Rational, Rational Unified Process, RUP, and WebSphere are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.