An Overview of Agile Software Development Methodology and Its Relevance to Software Engineering

Niko Ibrahim
Jurusan Sistem Informasi
Fakultas Teknologi Informasi, Universitas Kristen Maranatha
Jl. Prof. Drg. Suria Sumantri No. 65 Bandung 40164
Email: niko.ibrahim@eng.maranatha.edu

Abstract

Agile Software Development Methodology mungkin kurang dikenal dan jarang digunakan di lingkungan akademik. Namun pada prakteknya, metodologi ini sangatlah umum digunakan oleh para praktisi pengembang perangkat lunak. Jurnal ini ditulis untuk memberikan pandangan sekilas mengenai metodologi agile serta relevansinya di dalam setiap tahapan rekayasa perangkat lunak secara umum.

Keywords: Agile Methodology, Software Engineering

Introduction

Agile software development approaches have become more and more popular during the last few years. Agile practices have been developed with the intention to deliver software faster and to ensure that the software meets changing needs of customers. Some people say that agile software development is the "modern" replacement of the waterfall model (Larman, C. & Basili, V.R. ,2003).

The problem with traditional plan-driven software development methodologies (e.g. waterfall) are they are too mechanistic to be used in detail. As a result, industrial software developers have become sceptical about new solutions that are difficult to grasp and thus remain unused (Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J., 2002).

There are many methods classified as agile software development:

- Adaptive Software Development,
- Agile Modelling,
- Crystal,
- Dynamic System Development Methodology,
- Feature Driven Development,
- SCRUM and
- Extreme Programming (XP) (http://www.extremeprogramming.org).

Different authors emphasised different aspects of software development. Some focused on approaches to planning and requirements; some focused on ways to write software that could be changed more easily; and some focused on the people interactions that allow software developers to more easily adapt to their customers' changing needs. These various efforts created a focal point for a community that promoted the set of practices that succeed without many of the activities required by more defined methodologies.

Overview of Agile Software Development

Refer to Addison-Wesley Longman dictionary, the term "**Agile**" means able to move quickly and easily. Thus, "**Agility**" for a software development organization, means the ability to adapt and react quick and effectively and appropriately to changes in its environment and to demands imposed by this environment (Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J., 2002).

In the fall of 1999, Extreme Programming, Embrace Change1 was published and the trend had found its catalyst. In early 2001, the different innovators who were creating different agile methodologies held a retreat and scribed the "Agile Manifesto for Software Development." (Lowell, L. & Ron, J., 2004)

This manifesto emphasises the development on following things (http://www.agilealliance.org):

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

They declared the 12 principles of Agile Software Development as illustrated in figure 1.

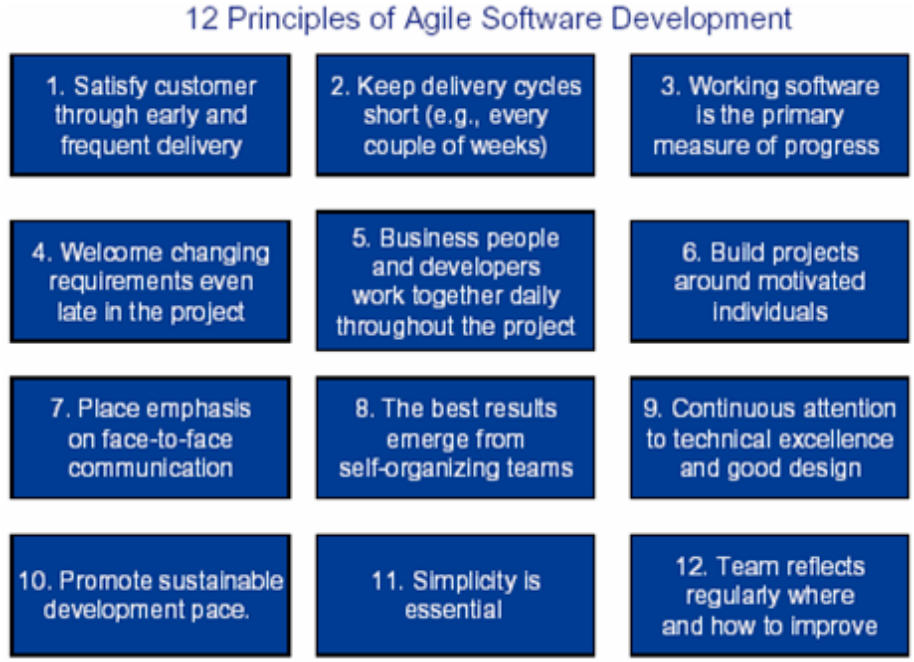

Figure 1. Twelve principles of Agile Software Development

Motivation

Agile or lightweight development methodologies are a solution attempt to challenge the difficulty and failure faced by the traditional / heavyweight / plan-oriented methodologies. With the characteristics of agile development methodologies (described later in section 2.2), developers are trying to develop softwares quickly without compromise the requirement as well as the quality of it.

In the heavyweight environment, developers have the challenge to bring a seemingly infinite backlog of software projects, while keeping side by side of the latest advances. Survey after survey continues to prove that most software projects fail against some measure of success. Software are delivered late, over budget and do not meet the quality requirement. Furthermore, it is also difficult to research the causes of these failures. However, typically, projects fail for one or more of the following reasons (Lowell, L. & Ron, J., 2004):

- Requirements that are not clearly communicated
- Requirements that do not solve the business problem
- Requirements that change prior to the completion of the project
- Software (code) that has not been tested
- Software that has not been tested as the user will use it
- Software developed such that it is difficult to modify

- Software that is used for functions for which it was not intended
- Projects not staffed with the resources required in the project plan
- Schedule and scope commitments are made prior to fully understanding the requirements or the technical risks

At the same time, numerous projects were very successful that did not follow methods with binders of documents, detailed designs, and project plans. Many experienced programmers were having great success without all these extra steps. The determining factor of project success seemed more and more to be the people on the project, not the technology or the methods that were being used. These become a motivation for developers to leave the heavyweight methodologies and to start developing software with agile methodologies.

Characteristics

Some of the common characteristics of agile development methodologies are:

Lightweight

Agile methods are easier to use than traditional (heavyweight) methods because they include fewer instructions when analysing, designing, or implementing the software requirements (Aksit, M., Mezini, M., & Unland, R., 2002, p. 412–430).

Adaptive

Heavyweight methodologies for software estimation and project planning work well if the requirements are clearly identified and they don't change. However, in most projects, the requirements do change during their lifetime, and therefore, developers need methodologies that can adapt well to the changing requirements. Agile methods permit a fast response to requirement changes since changes are considered as the rule, not the exception.

Iterative / incremental

In agile development, developers only need short project cycles. An executable system is not built near to the end of a project. Instead, it is built very early and delivered to the customer to be validated.

Cooperative

Agile development is cooperative since customers and developers working constantly together with close communication. The customer is assumed to be present at the development site and is involved in the development process.

Straightforward

The method itself is easy to learn and to modify, and also well documented. The website www.agilealliance.org provides a very good introduction and documentation on agile software development.

People-oriented

Agile development methodologies truly give power to the developers. The developers make all the technical decisions, they make estimations for work to be done, they sign up for tasks for iteration, and they choose how much process to follow in a project. So, it is people-oriented rather than process-oriented.

Despite those above characteristics, plan-oriented and agile methods are actually not strictly rivals. Both have their range of application or 'home ground'. They are used in projects with different characteristics (Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J., 2002):

- Plan-oriented methods in large projects with stable requirements in a critical application domain
- Agile methods in dynamic environments and in small teams producing rather non critical applications

The following is a table describing the range of application (home ground) of agile methods and plan-driven methods:

Table 1. Home ground for agile and plan driven methods

Home-ground area	Agile methods	Plan-driven methods
Developers	Agile, knowledgeable, collocated, and collaborative	Plan-oriented, adequate skills, access to external knowledge
Customers	Dedicated, knowledgeable, collocated, collaborative, representative, and empowered	Access to knowledgeable, collaborative, representative, and empowered customers.
Requirements	Largely emergent, rapid change	Knowable early, largely stable
Architecture	Designed for current requirements	Designed for current and foreseeable requirements
Refactoring	Inexpensive	Expensive
Size	Smaller teams and products	Larger teams and products
Primarily objective	Rapid value	High assurance

Figure 2 (Cockburn, A., 2002), shows one particular aspect of these differences. In this figure, the two diminishing curves show the potential damage to a project from not investing enough time and effort in planning. The two rising curves show the potential damage to the project from spending too much time and effort in planning.

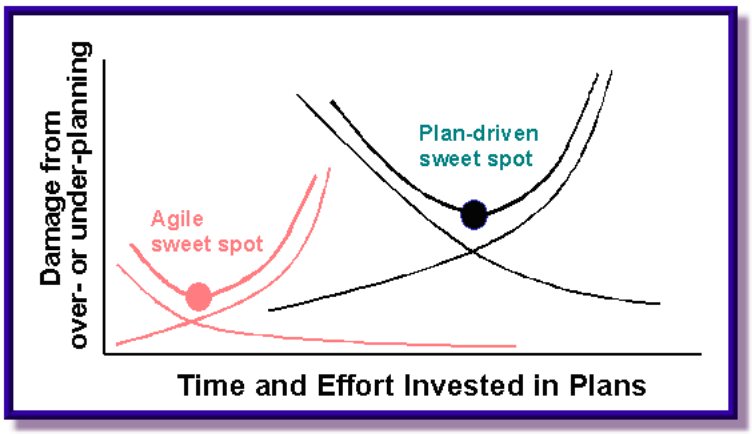

Figure 2. Balancing Discipline and Flexibility with the Spiral Model and MBASE

The lines crossing on the left indicate a project for which potential damage is relatively low with under-planning, and relatively high with over-planning. Much commercial software, including Web services fall into this category. The lines crossing on the right indicate a project for which potential damage is relatively high with under-planning, and for which much more planning would have to be done before damage would accrue from delays due to planning. Safety-critical software projects fall into this category.

The curves should make it clear that when there is risk associated with taking a slow, deliberate approach to planning, then agile techniques are more appropriate. When there is risk associated with skipping planning or making mistakes with the plan, then a plan-driven approach is more appropriate. The curves illustrate clearly the home territory of each (Cockburn, A., 2002).

The Main Benefits of Agile Software Development

In comparison to traditional software development, agile development is less document-oriented and more code-oriented. This, however, is not its key characteristic but rather a reflection of two deeper differences between the two styles (Frauke, P., Armin, E., & Frank M., 2003).

In this section we try to explain the main benefits of agile software development which is hardly obtained in the traditional software development.

Easier to plan and monitor

Agile software development emphasise a close collaboration of the customers and developers. This practice makes it easier for developers to plan and monitor the project. Moreover, agile methodologies usually recommend iterations and present a new version of the software of one week to three months.

Provide early feedback

Because of close collaboration between the customers and developers, agile methods are very good in providing communication between them. With their nature in frequently delivering working software, developers can get early feedback from the customers.

Gives early value to the customer

Again, because customer is involved throughout the process of software development, it may greatly improve customer satisfaction especially if the customer really understands their nature of requirement and is willing to get involved.

Enables the creative process

Agile methods are people-oriented rather than process-oriented. They rely on people's expertise, competency and direct collaboration rather than on rigorous, document-centric processes to produce high-quality software.

Responsive to changes

With traditional methods most of the software process is planned in detail for a longer time period. This works well if not much is changing and both the application domain and software technologies are well understood by the development team. In an application domain where changes often take place, agile methods really show its capability (Fowler, M., 2000). With an on-site customer who always available to provide answers to any clarification developers need, the development is really responsive to changes.

The Main Issues Surrounding Agile Software Development

While it seems that there have been many software development project successes based on agile processes, so far most of these success stories are only based on personal experiences (Dan T., Robert F., & Bernhard R., 2002). In this section, we try to explain the main issues and limitation surrounding the agile approaches:

Limited support for development involving large teams

In agile methods, control and communication mechanisms used are applicable to small to medium sized teams. If the software development project involves a large team, it will require less agile approaches to tackle issues that particular to large management (Dan T., Robert F., & Bernhard R., 2002).

Fail to provide an adequate level of structure and necessary documentation.

Because the nature of agile software development that emphasise the development on people and codes, it then hardly presents an adequate level of structure of the software. Different with plan-driven development, where documentation is one of the most important part of development, agile methods tend to fail in providing the necessary documentation because it stress early involvement of people and the need for rapid feedback.

Hard to develop large and complex software

In agile software development, it is assumed that refactoring can be used to remove the need to design for change (Dan T., Robert F., & Bernhard R., 2002). However, in large complex systems it may not work properly because there may be important architectural aspects that are very hard to change. In such system, models / designs really play an important role in which functionality is so tightly coupled and integrated that it may not be possible to develop the software incrementally.

Limited support for reuseability

Agile processes such as Extreme Programming focus on building software products that solve a specific problem. While there seems to be a case for applying agile processes to the development of reusable artefacts, it is not clear how agile processes can be suitably adapted.

Hard to handle non-functional requirements

Customers or users talking about what they want the system to do normally do not think about resources, maintainability, portability, safety or performance. Some requirements concerning user interface or safety can be elicited during the development process and still be integrated. Agile methods should include more explicitly handling non-functional requirements because they can affect the choice of database, programming language or operating system (Frauke, P., Armin, E., & Frank M., 2003).

Limited support for distributed development environments

Development environments in which team members and customers are physically distributed may not be able to accommodate the face-to-face

communication supported by agile processes (Dan T., Robert F., & Bernhard R., 2002).

Agile development can affect the structure within an organization

Agile software development spreads out the decision-making authority. This decision-making approach differs from what we see in many organizations. In some, the programmers make many key decisions, including those who directly connected to business, process, and systems requirements, while their managers either happily or unwillingly accept that way (William, L, & Cockburn, A., 2003).

Incorporating Agile Methodology to Software Engineering

Agile Methodology has general process that similar to the classical software engineering methodology. It starts with the requirement analysis, then followed by a design process, implementation, and testing. However, the details that happen in each step are quite different as it does not focus on the model but the rapidly changing requirements. This section describes the way in which agile approaches are different but can be integrated to the software engineering process in general.

Requirement Analysis

In agile software development, to have the customer accessible is the key point. This is the basis for fast feedback and communication which leads to better understanding of requirements and development process. For example in Extreme Programming method, it is assumed that customer is an ideal user representative that can answer all question correctly and has authority to make right decisions.

All agile approaches emphasize that talking to the customer is the best way to get the information needed for development and to avoid misunderstanding. If anything is not clear or vaguely defined, customers or developers should try to communicate with the person responsible to avoid indirect transfer of knowledge specifically (Frauke, P., Armin, E., & Frank M., 2003).

However, although agile methods are based on strong customer involvement during the whole development process, there is still a problem. In most cases, agile methods ask for several customers with different backgrounds, but sometimes only one customer is available to work with the development team. This means that not all the questions arising can be answered in enough detail.

Design / modelling

One of agile methods that incorporate modelling very much is Agile Modelling (AM). However, although modelling is used in AM, the purpose is

different. In AM, models are for example used to communicate understanding of a small part of the system under development. Most of the models do not become part of the final model of the system because they are mostly throw-away models and are drawn on a whiteboard or paper and erased after fulfilling their purpose (Frauke, P., Armin, E., & Frank M., 2003).

With this practice, agile method is suitable for small to middle size projects. For large and complex projects, plan-driven method is more suitable because it incorporates a comprehensive model of software in the design phase.

Implementation

The most interesting and extreme method when it comes to implementation is a method used in Extreme Programming. It is called "*pair programming*". In XP, programmers work together in pairs and as a group, with simple design and obsessively tested code, improving the design continually to keep it satisfy the current needs. However, pair programming doesn't mean all code is always written by two programmers working together using one computer. Some code is best implemented individually, and some code is best implemented in a pair.

Testing

In Extreme Programming, developers write unit tests for everything before they write the main code (Sharma, P., 2004). They write unit tests for all the normal conditions, as well as for any unusual circumstances (boundary conditions, etc.). Then, they run all unit tests and if they find some problem with the code, they add more unit tests to isolate the problem. After that they modify the code to fix the bug, and rerun all the tests.

Tools for Agile Methodology

There are several tools in the market that can be used for the agile approach. One of the tools is called VersionOne (www.versionone.net). Using such tool, we can plan and manage our agile software developments. VersionOne is focused on the planning, management, and delivery of rapidly changing, unpredictable software development projects. It also provides several template projects for Scrum, Extreme Programming or DSDM methods in a web based environment.

Another tool is called TP (www.targetprocess.com) that specifically designed to simplify planning, tracking, and quality assurance activities.

Conclusion

In conclusion, we can say that the agile methodology is an alternative to plan-driven methodology (eg: UML or Waterfall Model) in the development of quality software. It is also very suitable for small to middle size projects because developers do not usually concentrate on models as the

main component of the final software. In each software engineering phases, agile methods have unique approach that focus on feedback and change. Therefore, this method is believed can leverage the productivity of programmers.

Bibliography

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). 'Agile Software Development Methods: Review and Analysis', VTT Publications.

Agile Alliance Page, [Online], Available: http://www.agilealliance.org [20 October 2004]

Aksit, M., Mezini, M., & Unland, R (Eds.). (2002). 'Do We Need 'Agile' Software Development Tools?', Springer-Verlag Berlin Heidlberg, pp. 412–430.

Cockburn, A. 2002, Learning From Agile Software Development - Part One, [Online], Available: http://www.stsc.hill.af.mil/crosstalk/2002/10/cockburn.html [29 November 2004]

Dan T., Robert F., & Bernhard R. (2002). Limitations of Agile Software Processes, [Online], Available: http://www.agilealliance.org/articles/articles/LimitationsofAgile.pdf [27 November 2004]

Fowler, M. 2000, The New Methodology, [Online], Available: http://www.thoughtworks.com/us/library/newMethodology.pdf [20 November 2004]

Frauke, P., Armin, E., & Frank M. 2003, Requirements Engineering and Agile Software Development, [Online], Available: http://www.sern.ucalgary.ca/~milos/papers/2003/PaetschEberleinMaurer.pdf [26 November 2004]

Larman, C. & Basili, V.R. (2003). 'Iterative and Incremental Development: A Brief History', IEEE Computer, vol. 36, no. 6, pp. 47-56.

Lowell, L. & Ron, J. (2004). 'Extreme Programming and Agile Software Development Methodologies', Information Systems Management, vol. 21, pp. 41-61.

Sharma, P. (2004). An Introduction to Extreme Programming, [Online], Available: http://www.advisor.com/doc/13571 [28 November 2004]

William, L, & Cockburn, A. (2003) 'Agile Software Development: It's about Feedback and Change', IEEE Computer Society, vol. 3, pp. 39-44.