Agile Software Development

Asha Garg

Aeronautical Development Establishment, CV Raman Nagar, Bangalore-560 093

ABSTRACT

The bases of Agile software development is on the need to satisfy the customer through early and continuous delivery of the valuable software. It’s a set of best practice that allows rapid delivery of high quality software to meet customer needs and also accommodate changes in the requirements. Agile development is the ability to develop software quickly keeping pace with the rapidly changing requirements. In order to achieve this agility the practice with necessary discipline and feedback are required and the design principles should keep the software flexible and maintainable. Agile methods are a family of development process which can be chosen based on the project needs. They are more adaptive methods whose focus is on adapting quickly. Some of the Agile methods are Agile Modeling, Extreme programming, Pair Programming, Test Driven Development, Feature Driven Development, Open Unified process, Re-factoring etc. Agile practices benefit in terms of increased project success rate and user acceptance, better risk management, delivery of quality content on time and most important adjust to changing requirements. Today in the field of military embedded systems, developers face the pressure to shorten the life cycle and adapt to the changing requirements without forgoing the quality and reliability required for the safety critical applications. This requirement has led the thinking in the direction of Agile approaches. Agile methods chosen with special care for some of the concepts like change management, traceability, test driven development and automated documentation could form an adaptable approach for the embedded systems. This paper describes Agile Software Development methodologies and practices, and about how Agile can we get with the military embedded systems.

Keywords: Agile software, pair programming, software development, software

1. INTRODUCTION

Agile software development is a group of software development methodologies that are based on similar principles that allow rapid delivery of high-quality software in alignment with the customer needs. Most of the agile methods promote development iterations, teamwork, collaboration, and process adaptability throughout the life-cycle of the project.

In Agile development things are done in small increments with minimal long-term planning. Each iteration is worked on by a team through a full software development cycle, including planning, requirements analysis, design, coding, unit testing, and acceptance testing. This helps to minimize the overall risk, and allows the project to adapt to changes more quickly. Team composition in an agile project is usually cross-functional and self-organizing. Most agile methodologies include a routine and formal daily face-to-face communication among team members. This specifically includes the customer representative. Agile methods emphasize working software as the primary measure of progress. In an agile project, documentation and other project artifacts all rank equally with working product. Specific tools and techniques such as continuous integration, automated Unit test, pair programming, test driven development, code refactoring and other techniques are often used to improve quality and enhance project agility.

Agile methods are a family of development processes, not a single approach to software development. In 2001, these were coined as Agile with the basic values as:

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

That is, while there is value in the items on the right, the items on the left have been given more value.

Some of the principles behind the Agile Manifesto are:

- Customer satisfaction by rapid, continuous delivery of useful software
- Working software is delivered frequently
- Working software is the principal measure of progress
- Even late changes in requirements are welcomed
- Close, daily cooperation between business people and developers
- Face-to-face conversation is the best form of communication
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
• Simplicity in design
• Self-organizing teams
• Regular adaptation to changing circumstances

Agility is dynamic, context specific, aggressively change embracing and growth oriented. Agility implies maneuverability a characteristics that’s becoming more important in the field of software development. Agile software development thinking is based on empowering the people who are working on the software development to constantly improve their product and processes.¹

In serial development each workgroup waits for the upstream workgroup to achieve full stability before starting the task. (Fig. 1) While in concurrent development each group starts as early as it gets the information from upstream (Fig. 2) and as work progress upstream jobs passes on updated information to the downstream team. This shortens the elapsed time and provides feedback opportunities at the cost of rework. Good communication and right judgment of rework capabilities are most important in this model.

2. AGILE METHODOLOGIES

Agile Methodologies are the conventions that are adopted for a project and it can vary from one project to another. Some of the well-known agile software development methods are described below.

2.1 Agile Requirements Modeling

At the beginning of a project this is the stage to envision the high-level requirements and to understand the scope of the release (what the system should do). In this stage emphasis is to understand the higher level of requirements as to what the system should do. Typically initial requirements model can start as a Usage model exploring the users view of the system. This could be Use-cases for Unified process project, a collection of features for a feature Driven Development project or a collection of user stories for Extreme programming project. Then the domain information needs to be added containing the main domain entities, their major attributes, and the relationships between these entities. This should have enough information to be comfortable with the primary domain concepts. The level of details in this stage depends on the kind of system which is getting modeled.

2.2 Agile Modeling & Agile Model Driven Development

Agile Modeling (AM) is based on a collection of principles, such as the importance of assuming simplicity during modeling and embracing change because requirements will change over time. Ability to recognize that incremental change of the system over time enables agility and it is the rapid feedback which ensures accuracy.² Furthermore, multiple models are effective. Agile modelers believe that content is more important than representation, that there are many ways you can model the same concept yet still get it right. To model in an agile manner AM’s practices are appropriate. Fundamental practices of Agile Model Driven Development include:

2.3 Extreme Programming

Extreme Programming (XP) is a high discipline methodology which calls for tight adherence to the design and coding standards, developing strong unit test suites up front which must pass all the time, user defined acceptance
tests, constant working in pairs and vigilance to make design simple and aggressive refactoring.  

2.4 Refactoring

Refactoring is the process of changing a software in such a way that it does not alter the external behavior of the code yet improves its internal structure. Every software module has three functions. First there is a function it performs while executing, which is the basic reason for its existence. Second function of the module is to afford change. Almost all the modules go through changes. Third function is to communicate to the readers. Developers unfamiliar to the module should be able to read and understand a module without undue mental gymnastics. A module that is hard to change or understand is actually broken and needs to be fixed even though it works functionally. Refactoring neither fixes bugs nor adds new functionality, though it might precede either activity. Rather it improves the understandability of the code and changes its internal structure and design, and removes dead code, to make it easier to comprehend, more maintainable and amenable to change. Refactoring is usually motivated by the difficulty of adding new functionality to a program or fixing a bug in it. Some example of refactoring are:
- Changing identifier to more meaningful
- Changing a block of code into subroutine
- Changing a conditional statement to polymorphism

In extreme programming and other agile methodologies, refactoring is an integral part of the software development cycle.

2.5 Pair Programming

Pair programming is the technique where two programmer work together on programming of one module. While one is actually keying in the information another one works as reviewer and they frequently change roles. The benefits of pair programming are:
- Improved Design quality due to shorter programs, better designs, fewer bugs, more readable, more-maintainable designs, as well as catch design defects very early.
- Reduced cost of development due to large reduction in defect rate
- Automatic Learning and training by Knowledge passing, sharing and learning new programming techniques
- Overcoming difficult problems due to working together
- Improved morale
- Decreased management risk as knowledge of the system is shared among programmers
- Increased discipline and better time management

2.6 Test Driven Development

Test-driven development (TDD) is a software development technique that uses short development iterations based on pre-written test cases that define desired improvements or new functions. Each iteration produces code necessary to pass that iteration’s tests. Finally, the programmer or team refactors the code to accommodate changes. A key TDD concept is that preparing tests before coding facilitates rapid feedback changes. Test-driven development is a software design method, not merely a method of testing. With this method debuggers are very rarely used. Programmers also apply the concept to improving and debugging legacy code developed with older techniques.

2.7 Feature Driven Development

Feature Driven Development (FDD) is an iterative and incremental software development process. It is one of a number of Agile methods for developing software, FDD blends a number of industry-recognized best practices into a cohesive whole. These practices are all driven from a client-valued functionality (feature) perspective. Its main purpose is to deliver tangible, working software repeatedly in a timely manner.

3. AGILE FOR SAFETY CRITICAL EMBEDDED SYSTEM

Though the adoption to Agile methodologies started with small organization, now the large teams have successfully implemented complex systems using Agile methods. The organizations developing safety critical and mission critical systems are also tending to use Agile methods in order to take advantage of the many improvements agile practices can bring in the software development process. However, for organizations developing mission and life-critical systems, adopting all agile practices is not a viable option because some agile practices are not meant for the systems having characteristics of being large, complex, and long development periods. Now it is not the question of whether agile methods can be used in the safety-critical world, rather a question of how agile methods can be used in the safety-critical world. Following sections describe some of aspects of the safety critical system where additions to Agile practices are required. Hybrid agile/plan-driven approaches are feasible, and necessary for systems having a mix of agile and plan-driven characteristics.

3.1 Extension of Agile to Co-Design of Hardware and Software

In the system hardware and software are getting developed simultaneously thus resulting in co-design. In the embedded domain, hardware itself sets tight requirements constraints on the software. The dynamics of co-design i.e., the way it affects the concurrent software development processes, has to be analyzed in order to enable the use of agile software development methods.

3.2 Reliability and Safety Impact Analysis

High reliability is a desired system quality of mission and life-critical systems because it helps promote safety. One of the approaches used to attain high reliability in a system is to do Safety Impact Analysis which is conducted
before the design and coding phases. While the high level requirements should be frozen for doing safety impact analysis before design and coding phase, low level requirements can undergo iterations.

3.3 Preliminary Architecture Design

In safety critical systems meeting hard real time requirement is the first priority. In addition in the embedded systems there are reliability, performance and changing hardware requirements which needs to be dealt with. A considerable amount of architecture development is practically mandatory in composing the functionality for the system. Some of the architecture emerges through experience gained during development, but preliminary architecture design cannot be avoided. Hence for such systems preliminary architecture design needs to be added.

3.4 Refactoring

Refactoring needs to be done with care in the embedded systems. As far as application layer of the software is considered its fine to apply Agile practice of refactoring but refactoring of lower level code could give problems. The interactions between the software and the hardware are very sensitive to changes in timing. Changes in code – even if the code logically remains the same – may cause slight changes in timing or other behavior, which are very difficult to detect. Moreover to overcome the negative effects of refactoring use of software configuration management and automated testing are necessary. Automatic unit testing ensures that refactoring preserves correctness.

3.5 Configuration Control

Pervasive use of version/configuration control is one key ingredient in enabling fast-paced development work in an environment where seemingly harmless changes may cause bugs that are very difficult to locate and fix. This also has to entail relevant hardware development versions (simulation models etc.), as the functionality of software always has to be verified against the hardware, and vice versa.

3.6 Traceability

The iterations in the software will lead to issues related to the traceability management. Usually top down traceability is maintained but in safety critical system it will be better to maintain round trip traceability with the help of automated tools.

3.7 Automated Documentation

Minimal Documentation is a prevalent agile practice which is not suitable for the safety critical system due to the long development time. Personnel turnover and reallocation is bound to happen in these systems and hence explicit, comprehensive documentation is required. Another reason is that these systems gets maintained by third party and minimal documentation will fail to provide the necessary documented information that is needed by third-party organizations in maintaining the system for extended periods of time. Hence for the safety critical system automated documentation with the help of integrated tools is required which will be easy to generate and update.

4. CONCLUSIONS

Agile implies effective and maneuverable. An agile process is light and sufficient. Agile stresses on capturing of the good requirements right from the starts and iterated it with the end user/customer. Agile practices help in delivering the right product at the right time. Agile methods work on feedback mechanism and make the software much more maneuverable.

This study indicates that for safety critical systems development attempting to utilize both agile and traditional approaches is a solution with the benefits of both agile and traditional development.

The characteristics of embedded system development shows that the problems faced in the turbulent software/hardware boundary are largely those the agile methods are intended to solve. In particular, constant change in requirements and the need to experiment already necessitate the use of an iterative and incremental development process. Testing is also vital in embedded software development, yet another highly encouraged practice in agile development methods. Finally, efficient and timely communication between hardware and software developers is paramount. What is thus required is a method with the ability to scale smoothly during development to cater for the increasing need of formal communication, change management methods, and documentation.

REFERENCES
1. www.wikipedia.com
2. www.agilemodeling.com
Do Agile Methods Have a Chance? XP 2003, Genova, Italy.

