Comparing Agile Software Processes Based on the Software Development Project Requirements

Malik Qasaimeh, Hossein Mehrfard, Abdelwahab Hamou-Lhadj
Department of Electrical and Computer Engineering
Concordia University
1455 de Maisonneuve West
Montreal, Quebec H3G 1M8 CANADA
\{m_qasaim, h_mehrfa, abdelw\}@encs.concordia.ca

Abstract

Agile software development principles and methodologies have gained a lot of popularity in recent years. Unlike traditional software processes, agile processes advocate flexible practices that allow coping with changing requirements. Recently, several agile processes have been proposed, which differ in a variety of ways. In this paper, we analyze the characteristics of many agile processes and classify them based on key requirements for a software development project. The objective is to help project managers and software engineers select an agile process that is most suitable to the requirements of their software projects.

1. Introduction

Software development practices have evolved significantly since the term software engineering was popularized by F. L. Bauer during the NATO Software Engineering Conference in 1968. There exist today a large number of software processes that have been introduced and studied at a great extent, but up to date none has proven to be the golden standard.

A software process is defined as a set of activities, methods, practices, and transformations that are used to develop and maintain software and its associated products [3]. Agile processes have been proposed to overcome the flexibility issues of traditional processes. They have been developed by practitioners based on their experience working on several software development projects [8,10]. Although existing agile processes can differ significantly in the way they approach software development, they all share one key characteristic which consists of favoring close collaboration between software development and business teams via face-to-face communication, as opposed to putting an emphasis on written documentation [3, 8]. However, selecting the appropriate agile process is not an easy task due to many differences that are associated with each agile software process. These differences must be studied before choosing the appropriate methodology that fit the stakeholders’ needs [3, 4, 5]. In this paper, we surveyed and analyzed several agile processes, with the ultimate objective being:

- To help software engineers and project managers understand the strengths and weaknesses of existing agile processes.
- To help managerial staff understand the capabilities of agile processes to support software project requirements.
- To provide researchers in the area of software process engineering with a reference work in which key characteristics of several agile processes are discussed.

The remaining part of this paper is organized as follows. In Section 2, we describe the agile software processes considered in this study. In Section 3, we analyze these processes based on software development project requirements. We conclude the paper in Section 4.

2. Agile Processes

In this section, we describe the agile processes studied in this paper. We selected these particular processes in order to achieve good coverage of the types of activities available.

The processes selected in this study are the following:

- Extreme Programming (XP)
- Scrum
- Feature Driven Development (FDD)
- Adaptive Software Development (ASD)
- Crystal Methodologies (CM)

2.1. Extreme Programming (XP)

To many people, extreme programming (XP) has come to embody the agile methodology itself. This is, perhaps,
because it is one of the first agile processes that has been proposed. In general, XP consists of a set of individual practices that when put together yield a successful software practice [1].

Figure 1. Extreme programming (taken from [14])

Figure 1 depicts the activities involved in XP. Using XP, a development team starts by a planning phase, referred to as planning the game. During this phase, the technical team evaluates the feasibility of the project and the effort required as well as the schedule for the first release. The customer then defines the most valuable features, represented as stories, which need to be developed. Every story represents the smallest possible increment to which new features of the system can be added, which usually takes only a few weeks to be developed.

XP supports the idea that the design should be as simple as possible. To achieve this objective, XP puts an emphasis on using refactoring techniques such as removing duplicated code, improving as well as to improve the existing design. It should be mentioned that programmers must verify that the system is still operational after a refactoring activity takes place. The XP process requires that design, implementation, and testing of the system should be carried out by a pair of programmers sharing one computer. This allows programmers to spend more time finding solutions to challenging problem and less time doing routine debugging. Pair programming has been shown to be a useful technique for building robust software [7].

XP is a test driven development method such that, before writing code for a story, programmers must implement the automated tests that will verify the story functionality. As shown in Figure 1, programmers rely on unit tests to verify the correctness of the story. The work on a story is not considered complete until it has been shown to be defect free. Integration tests are run to verify that the overall functionality of the system is bug free. The story is not considered successfully implemented until it passes the acceptance test, which is normally written by the customer so as to verify that the system functionality satisfies the customer’s needs [1].

Using XP, the resulting implementation is owned by all team members. This collective ownership of the artifacts of the system allows programmers to make modifications to parts of the code that have been created by others. The main advantage of this practice is to speed up the development process such that when programmers detect a fault in the code he has the right to fix it. A coding standard is used to make sure that the development team use the same design and coding conventions. To keep the development team motivated, XP discourages team members from working more than 40 hours a week. In addition, overtime weeks are usually limited to no more than two weeks in a row [1].

2.2. Scrum

Scrum is an agile software development method that inherits many of the characteristics of the traditional iterative and incremental approaches (e.g., [1]). The term "scrum" is originated from the game of rugby, referring to a strategy of getting a ball back into play.

Figure 2. Steps of the Scrum process (taken from [14])

Figure 2 describes the activities involved in the Scrum process. The key activity consists of the concept of “sprint”, which consists of a 30-day working period with a set of specific goals. The Scrum process starts with a planning phase, during which a backlog list is developed to define the functionality of one or more releases of the system along with the risk associated with each release. The appropriate risk controls are also determined. The product backlog lists contain the total work of the project to be done. After that, a sprint planning meeting takes place. It usually starts every 15 to 30 days after the planning phase. During this meeting, customers, users, managers, and developers discuss the objectives of the next sprint release and the sprint backlog lists to be completed.

Scrum encourages team members to meet every day for 15 minutes in order to improve the overall knowledge of each member. Non-members may also attend but do not have the right to speak during the meeting, and any further discussions are avoided. These meetings are led by the Scrum master, a role given usually to the project manager. During these meetings, the Scrum master ensures that standards, conventions, and best practices are followed throughout the project. It is also the responsibility of the Scrum master to provide the necessary resources to team members so as to complete the sprint goals.

Scrum requires at least one daily integration and regression test of the code. In addition, a sprint review session of four hours maximum is organized regularly to discuss and report to the manager and the customer what has been
accomplished so far during the sprint. The sprint review session is also a way to receive feedback on regular basis from the various stakeholders involved in the project.

2.3. Crystal Methodologies

The Crystal methodologies are a set of processes that can be applied to different projects depending on the size and the complexity of a project. The framework in Figure 3 includes the factors that influence the selection of a particular methodology. The X-axis indicates staff size while the Y-axis represents the system criticality. The more critical the project, the more rigorous and formal processes are required. Crystal methods define four levels of critically:

- Life (L): A system failure is critical and may cause loss of life.
- Essential money (E): A system failure may cause loss of money.
- Discretionary money (D): A system failure may cause loss of money but can be fixed by referring to the system’s user manual.
- Comfort (C): A system failure may cause a loss of customer comfort.

Crystal methods put an emphasis on a set of policy standards that govern the way the project is managed. These standards are common among all crystal methods [1] and include incremental delivery of releases, progress tracking, direct user involvement, etc.

2.4. Feature-Driven Development (FDD)

The Feature-Driven Development (FDD) approach focuses on the software features of the system as the main driver of the development process. It differs significantly from the other agile processes by putting a strong emphasis on planning and upfront design [9].

As shown in Figure 4, the first step of the FDD process is to build a detailed model of the system to be developed, which captures the stakeholders’ assumptions and requirements. Once the domain model is built, the team members print a list of the features of the system. Each feature should be developed in a few hours or days, but no longer than 2 weeks. Using FDD, development teams are formed specifically to design and implement a particular feature. The work is usually performed in parallel on multiple
features. The teams are disbanded once the feature is completed and verified. Each team is led by a feature owner, who is responsible of the code segment that implements the feature. This is contrasted with the XP approach where the ownership of the code belongs to the whole development team and not to a specific member [15].

The FDD process utilizes rigorous inspection guidelines in order to find defects in the system. It also enforces coding standards. It also encourages regular builds on a daily or weekly basis in order to add newly designed features to the baseline system. Due to the fact that features are developed in parallel, it is important to have a configuration management system that allows proper integration of the changes made to the system.

Unique to the FDD approach is a tracking and papering mechanism that assesses the project status based on the number of features that have been implemented as well as the overall progress of the design, coding, and testing activities. Each feature is scored using a value ranging between 0 (for a feature that has not yet been worked on) and 1 (a completed feature) and anything in between refers to a feature in progress.

2.5. Adaptive Software Development (ASD)

Highsmith defines the adaptive software development as “a complex adaptive process that involves interaction between agents (stockholders), environment (organization) and the product (software)” [2]. The ASD process involves essentially three phases that are geared towards planning, reassessment and review. These phases are: Speculate, Collaborate, and Learn. During the speculation phase a definition of what to achieve in each iteration is determined. The collaboration phase highlights the importance of team work, by sharing the knowledge among the software developers. The learning phase is carried out after each iteration in order to improve the developer’s expertise as well as to enhance the quality of the work [1].

The ASD life cycle focuses on results rather than the tasks performed to reach these results. The results are mapped directly to form the application features. In other words, the results are combined together to form the overall functionality of the system. Similar to other agile processes, ASD encourages interaction between the developers, managers, and customers. In addition, the ASD approach favors sharing of knowledge among the software developers since building a complex system would typically require the collaboration of multiple domain experts bringing the needed knowledge and experience.

Figure 5 depicts the main phases of the ASD process. ASD starts with a project initiation phase. During this phase the project mission statement is established, which is defined to guides the overall process. It must be clear and well organized. The project normally starts with unclear requirements but after each adaptive loop the overall mission becomes clearer. ASD is a feature-oriented approach rather than task-oriented. The main focus is always on the features of the systems rather than the tasks needed to implement these features. During the concurrent component engineering phase, the developers may work in parallel to implement one or more features at the same time. One of the most important aspects of ASD is the quality review phase where the customers, developers and managers meet to discuss and assess the overall quality of the work performed. The review phase session, known as the joint application development session (JAD), is important for demonstrating the functionality of the system developed as well as to keep the project within the boundaries of the mission statement. Finally, a quality assurance and release phase is held at the end of the project to fix any problems regarding the quality of the work performed [1].

ASD is a risk-driven process, that is, the risk management in ASD should start as early as possible in the cycle and should be implemented in all stages, i.e., Speculate, Collaborate, and Learn.

3. Requirements for Software Development Projects

In this section, we discuss key criteria that characterize software development projects against which the above software processes have been compared (see Table 1). The list of requirements is by no means exhaustive, but we believe that it captures the most common requirements found in the literature (e.g., [4, 6, 8, 12, 13]).

3.1. Customer Involvement

Customer involvement is a key practice in all agile processes, as shown in Table 1. From this table, we can clearly state that agile processes consider customers as an integral part of the development process. For example, XP, crystal methodologies, and Scrum require on-site visits to customer’s venues to allow end users to verify and prioritize the requirements during the requirement phase. The
involvement of customers is also reflected during acceptance testing, where most agile processes require these tests to be written and executed by customers.

3.2. Time to Market

Most agile processes favor early delivery of software. The motivation behind this is to allow customers to start using the software system and provide early feedback that would further enhance the upcoming releases. Early deliverables help also improve the customers’ understanding of the expected software features. Agile processes provide a delivery schedule time which varies between two weeks (e.g., in FDD) to almost six months such as in Crystal orange.

3.3. Responding to Change

As changes are costly to accommodate later in the project, the ability to respond rapidly to changes would therefore reduce project risks and costs. The surveyed agile processes rely on some sort of review sessions involving customers to identify changes in the requirements. XP and crystal methodologies, for example, rely on frequent on-site customer visits and direct user intervention techniques to identify any possible changes triggered by customers. Scrum and ASD rely on daily meetings where customers are involved. FDD, on the other hand, identify changes during the specification of the features that need to be implemented.

3.4. Documentation

The documentation in agile processes is reduced to informal documentation, face to face communication, and on-site customer visits. Using informal documentation reduces the cost associated with maintaining it [4]. The agile processes studied in this paper vary in the level and the type of documentation they provide. For example, XP uses user stories to capture the software features that need to be implemented. Scrum’s main documentation consists of product and spring backlog lists. FDD and Crystal methodologies use UML diagrams such as use cases, class diagrams, and object models to document the design. Test cases have also been used by XP and Crystal methodologies as documentation artifacts.

3.5. Verification and Validation

Agile software processes vary in the amount of verification and validation required. This requirement will enable a project manager to select an agile process that would fit the verification and validation requirements of the software project at hand. For example, a critical safety system will most likely require extensive testing than other types of software.

Agile processes put a strong emphasis on unit and integration testing. For example, XP is known as a test-driven design approach where test cases for unit and integration testing are written before writing code. In addition to unit testing, regression testing is used in Crystal methodologies and Scrum.

Other quality review techniques are also used such as design and code inspection (in FDD), sprint review (in Scrum), and code quality review (in ASD).

3.6. Team Management

This requirement is important for organizing the team from many prospective such as team size, team communication and the use of standardized procedures (e.g., design conventions), etc. Team size is one of the important factors that may affect the selection of the development process. Although agile processes emphasize a face-to-face communication instead of formal documentation, the number of developers considered is a serious obstacle to the effectiveness of the communication. Except crystal orange, all other agile processes suggest at most 20 persons per team. Team communication considered as the second factor in team management. Agile processes tend to be people-oriented processes by allowing team members to take appropriate decisions when required without being restricted to any procedure or technique.

The use of code standard guidelines has been proposed in XP and Crystal methodologies to facilitate exchange of information among team members. This is also due to the fact that these processes favor collective ownership of the system artifacts. In other words, any member can modify the code or design of someone else. In such cases, standard coding guidelines facilitate the collaborative work.

3.7. Collaborative Work

Some software projects require coordination of project teams located in different regions, countries, and time zones. This is typically the case for large companies that have shifted parts of their software development activities to geographically distributed locations. A software process that is selected for this type of projects needs to allow for smooth collaboration among geographically distributed team members.

Agile processes do not discuss how collaborative work can be performed. This lack of support for collaborative work has also been reported by Turk et al. in [13].
Table 1: Mapping agile process to software development project requirements

<table>
<thead>
<tr>
<th>Agile Process</th>
<th>Time to Market</th>
<th>Responsibility Change</th>
<th>Documentation</th>
<th>Verification and Validation</th>
<th>Team Management</th>
<th>Collaboration Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>XP</td>
<td>2-3 months</td>
<td>One-on-one customer</td>
<td>User stories</td>
<td>User Stories</td>
<td>Test cases</td>
<td>1 team per project (2-10 per team)</td>
</tr>
<tr>
<td>Scrum</td>
<td>30 days</td>
<td>Daily Scrum meetings</td>
<td>Sprint backlog</td>
<td>Sprint Review</td>
<td>Integration testing</td>
<td>1 team per project (2-10 per team)</td>
</tr>
<tr>
<td>FBD</td>
<td>2 weeks</td>
<td>Plan and design</td>
<td>Overall model design</td>
<td>Design inspection</td>
<td>Code inspection</td>
<td>1 team per project (2-10 per team)</td>
</tr>
<tr>
<td>ASD</td>
<td>4-6 weeks</td>
<td>Ad hoc cycle planning</td>
<td>Project data sheet</td>
<td>Unit testing</td>
<td>Regression testing</td>
<td>1 team per project (2-10 per team)</td>
</tr>
<tr>
<td>Crystal Clear</td>
<td>Direct user</td>
<td>Direct user involvement</td>
<td>Object models</td>
<td>Unit testing</td>
<td>Regression testing</td>
<td>1 team per project (2-10 per team)</td>
</tr>
<tr>
<td>Crystal Orange</td>
<td>Direct user</td>
<td>Direct user involvement</td>
<td>Object models</td>
<td>User manuals</td>
<td>Feature description</td>
<td>1 team per project (2-10 per team)</td>
</tr>
</tbody>
</table>

4. Conclusion and Future Directions

In this paper, we presented our analysis of five agile software processes and compared them based on criteria relate to software development projects. The objective is to help software engineers and project managers understand the key characteristics of these processes and therefore select the most suitable process with respect to the type of software projects they develop.

As future work, there is a need to include other agile processes not covered in this paper such as the Dynamic Systems Development Method (DSDM), the Lean Software Development (LSD), etc.

5. References


