eXtreme Programming and SCRUM: 
A Comparative Analysis of Agile Methods

Nicholas R. Zuiderveld, Portland State University

Abstract—Agile methods are playing an increasingly important role in the realm of software development. In an attempt to develop improved methods to designing and implementing software systems, several pioneering efforts have been suggested. This paper examines two such agile methods, eXtreme Programming (XP) and SCRUM.

The values and principles of the eXtreme Programming movement are described, including commentary on the effectiveness of XP in practice. The SCRUM development process is described in detail, followed by a brief look at the advantages and disadvantages SCRUM provides.

XP and SCRUM are compared in full, focusing on the crucial differences between the two agile methods. Finally, the way in which XP and SCRUM can be used together to obtain maximum benefits is suggested.

Index Terms—Agile Methods, eXtreme Programming, SCRUM.

I. INTRODUCTION

Frequently software development is characterized by the phrase “code and fix,” where most of the code is written without a great deal of planning, resulting in rather problematic software which must be extensively debugged. To rectify this problem, several methodologies have been introduced to guide the software development process along a more predictable path. This has resulted in plan-driven approaches to development, where a strong emphasis on early planning and strict documentation is intended to more accurately predict the pace of the project life cycle. However, such methodologies often slow down the pace of the entire project since so much paperwork must be worked through to move forward.

Unlike these bureaucratic methodologies, agile methods have become a popular alternative for software development. The essential values held by agile programmers can be found in the “Manifesto for Agile Software Development” [1]:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires no prior specific permission or fee.

N.R. Zuiderveld is a graduate student of Computer Science at Portland State University, Portland, OR 97201 USA (e-mail: nrzuiderv@cs.pdx.edu).

In this paper, we look at two such agile methods, eXtreme Programming and SCRUM, both of which profess to improve upon existing software development methods, though in different ways. First we describe eXtreme Programming and SCRUM, highlighting their goals and features. Following that, we provide a detailed comparison of the two agile methods, focusing on the different ways they attempt to resolve problems encountered in traditional software development methods. Finally, we examine ways in which eXtreme Programming and SCRUM could be used together, including what advantages such a symbiotic relationship could yield.

II. eXtreme Programming (XP)

A particularly popular agile methodology is eXtreme Programming (XP). XP was developed by Beck, Cunningham, and Jeffries and is a “lightweight discipline of software development based on principles of simplicity, communication, feedback, and courage” [2]. These four key values are described in further detail below:

A. Simplicity

Rather than attempting to predict future requirements of the software system at the outset, “extreme programmers do the simplest thing that could possibly work,” and “leave the system in the simplest condition possible.” [3]. This improves the overall speed of development while still retaining an emphasis on working software.

B. Communication

Poor communication in software teams is one of the central causes of failures within projects. In XP, good communication is stressed between all project members—customers, team members, and project managers. A representative from the customer should be present on site at all times to answer questions and clarify project requirements. Pair programming is used, so each programmer can constantly review the other’s work.

C. Feedback

There should always be some way of getting information
about the system, to accurately determine the state of the development process. Such feedback serves as an indicator of the project’s progress and informs project leaders when changes need to be made.

D. Courage

XP programmers are encouraged to experiment and rewrite code if they are dissatisfied with the existing code or design. This helps maintain morale about the project and supports further communication with other project members.

Based on these four core values, Beck [4] and Tomek [6] describe 12 core XP practices.

1) **Planning Game**—There is no long-term strategy in an XP project, but rather a planning phase of development, where “user stories” are collected from the customer. The development team performs and estimating process to determine what user stories will be implemented in the next release, and provides a short-term schedule as to when the next release of software will be completed.

2) **Small Releases**—Similar to the evolutionary software development model, XP sets out to deliver a series of fully-functional and completely tested releases leading up to the delivery of the final product. In so doing, the development team avoids the pitfall of having to integrate several large components of an overall system at the end of the product development period.

3) **Metaphor**—A simple yet effective way of getting all members of the project team to visualize the project is by providing a metaphor, which “should provide inspiration, suggest a vocabulary, and a basic architecture” [5]. This is the only principle not strictly required in every XP project.

4) **Simple Design**—Given the emphasis in XP that simplicity should be favored, the design should reflect the desire for the bare minimum of code necessary to meet the requirements at each step of development. The design should be simple enough to allow for refactoring at any point necessary.

5) **Testing**—XP takes a much different stance than traditional software development practices in that test code is written prior to an implementation code. By taking this step, test code becomes essential to further design of the software. This strong emphasis on testing appealed to Fowler [6]:

While all processes mention testing, most do so with a pretty low emphasis. However XP puts testing at the foundation of development, with every programmer writing tests as they write their production code. The tests are integrated into a continuous integration and build process which yields a highly stable platform for future development.

6) **Continuous Integration**—To ensure that code is kept consistent and available to all development team members, daily updates are made to a shared code repository. Before it can be released into the repository, though, it must first pass all the test cases, so that the integrity of the code is constantly maintained.

7) **Pair Programming**—This method has proven to be very successful, where two programmers share a machine. The main advantages of this method are: continuous inspection by two people, learning (each programmer learns from the other), and mutual stimulation and encouragement.

8) **Collective Ownership**—No single individual or group “owns” a piece of software. Once code has been deposited into the shared repository, it is available for any team member to modify, so long as no tests are broken as a result. This improves the communication and understanding about the project across the entire development team.

9) **Refactoring**—Refactoring is a “disciplined technique for restructuring an existing body of code, altering its internal structure without changing external behavior” [7]. As one of the fundamental values guiding agile methods in general, design changes are possible even late in the development cycle.

10) **40-Hour Week**—Not strictly defining a specific “optimal” workweek, it rather refers to “work at a sustainable pace,” [5]. Simply speaking, programmers (and people in general) perform their best work when they are well rested, upbeat, and healthy. As evident in other values expressed in the XP community, people play an important role, and are not considered simply robotic programmers.

11) **On-Site Customer**—Having the customer available at all times during project development is crucial for XP practices, since at any time decisions affecting the project may arise, and often only the customer can accurately respond to such questions. Not only does this increase overall production speed, but is also a good way to avoid potential legal problems about contractual obligations.

12) **Coding Standards**—Coding standards are virtually unavoidable in XP, due to the continuous integration and collective ownership properties. Communication about the project is also maximized when everyone has a common approach to coding and documentation.

By adhering to these values and principles, many XP has garnered much praise as an efficient alternative to older plan-driven methodologies. Fowler [6] writes,

The result is a design process that is disciplined, yet startling, combining discipline with adaptivity in a way that arguably makes it the most well developed of all the adaptive methodologies.

Yet XP has not been the silver bullet that some followers hoped it might become. Not every project is suitable for the XP methodology, especially in development projects involving more than 100 people, and many people doubt the ultimate effectiveness of XP compared to more traditional
methods.

**The Case Against Extreme Programming**

There is currently an ongoing debate as to where the future of software development is headed, with supporters of agile methods calling for a complete upheaval of the current ways, while defenders of plan-driven methods are holding fast to their foundations. Opponents of XP accuse the method of being unrealistic in many cases, especially in its demand for “premium people” [8], which may be difficult to come across. Other points raised in [8] include customer dissatisfaction despite customer interaction, misapplication of refactoring techniques, and the size of teams required for XP projects to succeed (which are only optimal at below 25 people). One author’s displeasure with XP practices lead to the reworking of a John Lennon classic:

**Imagine**

*Imagine there’s no requirements, It’s easy if you try*
*Just a bunch of coders, reachin’ for the sky*
*Imagine all the people, coding for today*

*Imagine there’s no schedules. It isn’t hard to do*
*No silly project deadlines, no one supervising you*
*Imagine all the people, coding hand in hand*

*You may say I’m an extremer but I’m not the only one*
*I hope someday you’ll join us and make coding lots more fun.*

*Imagine oral documentation. I wonder if you can*
*No need for UML diagrams. Just words passed, man to man*
*Imagine just refactoring, playing in the sand*

*You may say I’m an extremer but I’m not the only one*
*I hope someday you’ll join us and make coding lots more fun.* [9]

Despite the current debate over XP and agile methods in general, it is reasonable to assume that the old plan-driven methods are not going to be supplanted by XP, but neither is XP going to cause a full-scale revolution. As Boehm [8] makes clear, there are situations in which one or the other method is preferred, and it ultimately comes down to a variety of factors that must be evaluated before any methodology should be adopted.

**III. SCRUM**

XP developed partly in response to inadequacies and limitations in traditional software development methods. Similarly, the SCRUM development process was introduced to make development of software systems more flexible and lightweight than traditional heavier methods. SCRUM was first implemented by Jeff Sutherland [10] in 1993 while working at the Easel Corporation. Following to the initial successes achieved, SCRUM was further developed by Sutherland and Ken Schwaber into a formal process in 1996. Since that time, the SCRUM development process has gained in popularity, and several companies, both large and small, have successfully taken advantage of the method’s agility.

Stated simply, “Scrum is an iterative, incremental process for developing any product or managing any work” [11]. The primary characteristic of SCRUM relative to more traditional development methods, is that it assumes an element of chaos in the development process. Compared to the traditional methods such as Waterfall, Spiral, and Iterative, SCRUM is best prepared to handle changes in the environment, and can more easily respond to changes in requirements, schedule, and other externally or internally defined updates. “SCRUM acknowledges that the underlying development processes are incompletely defined and uses control mechanisms to improve flexibility” [12].

SCRUM achieves this flexibility and adaptability through a well-defined development process designed to recognize and respond to changes in the environment. The key features of the SCRUM methodology are the Planning phase, Sprint phase, and Closure phase, which are described in further detail here:

**A. Planning Phase**

During the planning phase, a Product Backlog is created, which is a list containing the features desired by the customer [11]. The delivery date for the final product is specified, prioritization of system components are laid out, project cost is estimated, and potential risks to the product development are assessed [12]. All information regarding what the intended product should be is determined at the outset, before any development begins.

**B. Sprint Phase**

The actual project development occurs in the Sprint phase. A Sprint is an “iterative cycle of development work,” [12] and generally lasts between 1 and 4 weeks. At the outset of a Sprint, a subset of features from the initial Product Backlog is assigned to be completed during the Sprint. During the course of a Sprint, no other features can be added, but features within the Sprint’s Backlog can be updated or changed depending on environment variables. A more detailed description of the Sprint process is provided below.

**C. Closure Phase**

Once a product has met expectations of the development team, management and customer, the product is prepared for general release as part of the Closure phase. At this point, the product is final and tasks such as preparing training materials, adding user documentation, and preparing marketing material are completed [12].

**SCRUM Sprints**

Sprints begin by meeting with each development team to determine what the Sprint time length will be, what features from the Backlog are to be implemented, and what standards to which the product must conform. At the end of a Sprint, a deliverable product will be available. Once the preliminary guidelines and requirements for the Sprint are specified, the development begins. A Scrum Master is assigned to manage
the development during a Sprint.

Crucial to the Sprint phase are *Scrum Meetings*, which are held daily to determine the progress of the release and to respond to problems encountered along the way. Scrum Meetings are led by the Scrum Master, who asks the same three questions to each team member every day to evaluate the development progress [13]:

1. *What did you do yesterday?*
2. *What will you do today?*
3. *What’s in your way?*

By asking these questions at a daily meeting, everyone is encouraged to report how they are proceeding on the project, and problems can be rooted out quickly and corrected. The Scrum Master tracks what task each member is working on, and is responsible for resolving problems. Not only do Scrum Meetings allow for greater flexibility when adapting to encountered problems, but “meeting at the same time, and with the same people, enhances a feeling of belonging, and creates a habit of sharing knowledge” [14]. Thus, the team’s morale and knowledge base is strengthened, thereby improving the overall effectiveness of each team member.

At the conclusion of each Sprint, a review session occurs, to evaluate the resulting deliverable and examine how the release was implemented. New items can be added from the Product Backlog, and a new Sprint can begin if necessary.

Figure 1 provides a graphical representation of the SCRUM development process, emphasizing the Sprint phase [15]:

![Figure 1. SCRUM Development Process](image)

**SCRUM – A Productive Advancement in Software Development?**

Several software development companies and organizations have adopted SCRUM as the preferred method to produce functioning, deliverable products in an environment where system requirements are not fully defined at the beginning of the development process. Independent Software Vendors (ISVs) such as ADM and VMARK have applied SCRUM and yielded outstanding results, citing advantages that the SCRUM model provides:

- It breaks down large products into manageable chunks – a few product features that small teams can create in a few months.
- It enables projects to proceed systematically even when team members cannot determine a complete and stable product design at the project’s beginning.
- It allows large teams to work like small teams by dividing work into pieces, proceeding in parallel but synchronizing continuously, stabilizing in increments, and continuously finding and fixing problems.
- If facilitates competition based on customer feedback, product features, and short development times by providing a mechanism to incorporate customer inputs, set priorities, complete the most important parts first, and change or cut less important features. [16]

But SCRUM is not necessarily for everyone. As can be seen from the process model, management is critical, as teams require constant and thorough oversight to ensure that problems can be caught and corrected quickly. At the same time, managers external to the Sprints must allow the teams a degree of freedom to make decisions to aid the current Sprint development. Perhaps most importantly, SCRUM is a relatively new model, quite different from the traditional methods most managers and developers are used to. Not everyone will be prepared to adapt to this new ideology for software development [17].

Before adopting a new and unfamiliar methodology to produce software, a number of factors should be considered, from what sort of features the product will provide, to what the customer’s use of the product will be, as well as what sort of development skills the software team possesses. In cases where requirements for a software system can be clearly stated at the outset, SCRUM may not provide much of an improvement to the development process, as it does require more time to meet daily and monitor each person’s progress closely. Yet in many situations, SCRUM allows development to be more flexible and adaptable to changing needs of a system, making it an excellent model to employ when such conditions are expected.

IV. COMPARISON OF XP & SCRUM

We have looked at two agile methods that claim to improve the software development process. eXtreme Programming establishes a list of core principles that support and promote the values of simplicity, communication, feedback, and courage. Proponents of XP argue that adhering to their principles can lead to fast and efficient software development. SCRUM sets forth a specific development process that emphasizes frequent meetings to constantly monitor the progress of the product. In so doing, suggest those using SCRUM method, reliable releases can be delivered, despite shifting system requirements. Both XP and SCRUM make attempts at addressing problem areas in traditional software
development methods, but are their similarities only superficial? Where exactly do their differences lie? What needs does one method meet that the other does not? We now turn to a closer examination of the similarities and differences between XP and SCRUM.

**XP & SCRUM – The Similarities**

The greatest parallels between XP and SCRUM have already been suggested, that they both try to address shortcomings in bureaucratic plan-driven methodologies. But beyond that common goal, they share a number of values that are evident in their implementation:

- Both XP and SCRUM take into account a degree of unpredictability in system specifications. In the case of XP, a relatively brief planning phase is used, rather than attempting to define all system requirements at the outset. Similarly in SCRUM, planning is performed up to a point, but the understanding that specifications may change during the Sprints allows for rapid adaptation. In SCRUM, this unpredictability is often referred to as **chaos**.

- Both XP and SCRUM focus on producing working software deployments in a short time. The 2nd XP practice specifies the need for small working releases, so that as each component of a larger system is created, there is less danger of the overall product failing. Similarly for SCRUM, these functional deliverables are released at the conclusion of each Sprint, and updated however necessary. By emphasizing such short cycles until each component becomes available, developers avoid becoming entrenched with a specific design choice, and are better prepared to handle modifications in system requirements.

- Both XP and SCRUM stress frequent communication between team members. For XP, this is crucial when communicating with the on-site customer to clarify system requirements and priorities. When refactoring the system, communication is fundamental to ensuring that all programmers understand the design correctly. In the SCRUM development model, communication is required daily in the form of the Scrum Meeting. Without frequent communication, errors could go undetected, requirements or priorities misunderstood, or many other such problems. When such rapid development is being performed, proper communication is the only way to prevent outright chaos from overtaking the project.

**XP & SCRUM – The Differences**

Despite the frequent overlaps in XP and SCRUM ideologies, there remain several key differences between the two in practice. The fundamental difference between the two can be summarized by the following statement: SCRUM provides agile management mechanisms; eXtreme Programming provides engineering practices [18]. SCRUM pays no attention to the engineering practices, focusing instead on the business of managing the software development. XP, on the other hand, is primarily concerned with those engineering disciplines that will yield the best code. Due to this difference, several distinctions about the two agile methods can be made:

- SCRUM places great responsibility in the Scrum Master, who manages the development team. He is the central figure who controls the direction of product development. In XP, the practice of collective ownership allows any programmer to modify a section of code when it needs to be fixed. This is accomplished as the need arises, and is not regulated by any central managing figure.

- XP requires that the software be validated at all times, to the extent that tests are written prior to the actual software. In SCRUM, validation of the software is completed at the end of each Sprint during the Sprint Review, not at each step within the Sprint.

- XP also involves a 40-hour week, suggesting that when programmers are more rested, they produce better quality code. The SCRUM process makes no such fine distinction; the workweek could be more or less than 40 hours, depending on the place of business and the employer who specifies the hours worked.

- In XP, the practice of pair programming is one of its most famous features. With two programmers working at the same workstation, the code produced is purported to be of a much higher quality. SCRUM development is also silent on this subject, neither advocating nor contesting its use. Such a decision would rest in the hands of the project leader, who would define the engineering methodologies to be used during the Sprint.

Though XP and SCRUM differ in ways, could they be combined in such a way as to function cooperatively? Could the management techniques of SCRUM be applied to the more specific engineering practices of XP? We examine how XP and SCRUM can be utilized simultaneously in the following section.

V. **XP & SCRUM: A COOPERATIVE RELATIONSHIP?**

SCRUM and eXtreme Programming share the common goal of providing agile answers to problems encountered in traditional plan-driven development methods. Since SCRUM is a management model and XP a series of engineering practices, the two could potentially be combined to form an agile development process supported by agile programming
techniques that could itself be more agile than either of the two taken alone. The business-oriented model that SCRUM provides can be complemented by the high-quality software produced by eXtreme Programmers [19].

Working from the SCRUM framework, the Product Backlog could be compiled by speaking with the customer to determine system requirements and priorities. When a Sprint begins, the XP practices could be brought into play, initially using a simple design to plan out the Sprint’s goals. Then, once programming is started, the XP practice of testing could be used to first implement the tests before coding begins, all of which would be accomplished through pair programming conforming to coding standards. The daily Scrum Meetings would take place as normal, where the three questions are asked to determine what changes must be made to ensure that progress continues steadily. When the Sprint finishes and the release is evaluated at the Sprint Review, any refactoring of the system requirements can be added to the Backlog before a new Sprint begins.

The adoption of both SCRUM and XP for a development project can thus result in a cooperative relationship, taking advantage of the strengths of each method, and minimizing their weaknesses. Indeed, SCRUM’s primary weakness occurs when poor engineering practices are used. If the programming is low quality, no amount of wise management techniques will help raise the quality of the final product. Similarly, eXtreme Programming specifies no overarching framework for software development, instead focusing on the engineering practices at the coding level. Under poor management, where guidelines are not clearly defined and meetings established, the practices of XP may prove to have little advantage in the success of the final software product.

Some development teams have already applied both XP and SCRUM to a project, with successful results. Mar and Schwaber [19] led a project that combined the use of XP and SCRUM, and concluded the following:

We believe that the joint application of these methodologies instills practices and values in management, customers, and development that allow them to create a unified, disciplined team. In our experience, these two practices are complementary; when used together, they can have a significant impact on both the productivity of a team and the quality of its outputs.

Though both XP and SCRUM can have a tremendous impact when used individually as either engineering practices or management techniques, their complimentary features make them ideal when used in conjunction with one another.

VI. CONCLUSION
In this paper, we first described eXtreme Programming, or XP. XP is built upon four core values: Simplicity, Communication, Feedback, and Courage. From these four values, a dozen practices for XP projects were described, which were: the Planning Game, Small Releases, Metaphor, Simple Design, Testing, Continual Integration, Pair Programming, Collective Ownership, the 40-Hour Week, On-Site Customer, and Coding Standards. Though XP is gaining popularity, especially in younger programmers, some still doubt whether it can be practical in all situations.

We then looked at SCRUM, a development process which focuses on management practices to complete a project while operating on the brink of chaos. Elements of the SCRUM process include the Planning Phase, Sprint Phase, and Closure Phase. The crucial work of SCRUM is done in Sprints, where features taken from a clearly defined Backlog are implemented over the course of 1 to 4 weeks. Every day, a Scrum Meeting is held, where three questions are asked to determine how each team member is progressing. At the conclusion of each Sprint, a Sprint Review takes place, where the work accomplished during the Sprint is evaluated.

Finally, we compared XP and SCRUM, highlighting their similarities and differences. We noted how XP defines a set of engineering practices, whereas SCRUM sets forth a project management model. Following that, we speculated how XP and SCRUM could be combined in a complimentary fashion to develop high-quality software in a timely manner.

REFERENCES
[18] “XP @ Scrum”, http://www.controlchaos.com/about/xp.php