East Tennessee University – 5520 Research Methods

Agile Methodologies

Erin Lorelle

August 9, 2018

Abstract

Agile methodologies represent attractive alternatives to Waterfall-like software engineering. They provide an efficient approach to software development, focusing on frequent collaboration, rapid release of software products, and easy adaption to change. The core principles of agile have been applied in different ways, yielding different methodologies for different needs. Scrum and XP are the most practiced methodologies, while newer methods Kanban and BDD are gaining popularity. FDD and DSDM are methods for larger teams seeking structure for complex projects. For a small team in a capstone environment, the agile models most suitable are Scrum, XP, and also Kanban, which is adjustable to all size teams.

In 2001, seventeen software developers devised the Agile Manifesto, a rationale for software engineering that featured twelve principle guidelines for software development.

- Customer satisfaction by early and continuous delivery of workable software.
- Changing requirements are welcomed even in later development stages.
- Frequent collaboration and communication among customers and developers.
- Frequent delivery of working software.
- Support and motivate trusted people involved in software development.
- Use face to face communication.
- Working software is main measure of progress
- Constant pace is maintained through sustainable development.
- Pay attention to good design continuously.
- Keep things simple.
- Self-organizing teams can develop better architecture, requirements and design.
- Team regularly reflects how to become more effective. (Anwer, Aftab, Waheed, & Wyed, 2017)

The authors referred to their methods as "agile" (Sambare & Gulabchand, 2017) because of the methods' emphasis on generating "quick releases of software products that meet quality and budget constraints," (Sun & Schmidt, 2018). These guidelines were intended as an alternative to waterfall-like development methodologies, which can preclude a customer's "see[ing] the end product until the completion of the project", a process that could "take several months to years to complete" (What is Agile methodology? , n.d.).

Agile methodologies emphasize customer involvement, frequent product delivery, and requirements flexibility - i.e., adapting to mid-project changes for quick turnaround. They tend to be lightweight, meaning that "[process] overhead is [reduced to] as little as could reasonably be expected, to boost the measure of work done" (Sambare & Gulabchand, 2017). Documentation is minimized to allow team members to spend more time writing code. Short-term planning is emphasized in place of long-term planning, with development managed in short iterations to allow for frequent release and testing (Matharu, Singh, Mishra, & Upadhyay, 2015). Examples of agile methodologies include Scrum, which emphasizes development divided into sprints; XP, which emphasizes communication and simplification; TDD, which emphasizes continual testing and refactoring; BDD, which emphasizes behavior testing; Kanban, which emphasizes efficiency; FDD, which emphasizes quality features; and DSDM, which emphasizes project management.

Scrum is currently the most frequently practiced agile methodology. Scrum was created by Jeff Sutherland in 1993 (Sambare & Gulabchand, 2017). It decomposes a software development process into a series of sprints. The sprint process emphasizes four activities: developer collaboration, using product and sprint backlogs to guide development, daily project reviews, and keeping backlogs current.

Collaboration is Scrum's main focus. All work, discussions, and decisions are undertaken as a group. All team members participate equally in Scrum, with projects often undertaken by several collaborating teams (Matharu, Singh, Mishra, & Upadhyay, 2015). Two team members assume distinctive roles as part of Scrum. One, a Scrum master, helps ensure that rules and guidelines are followed. The other, the Product Owner, takes responsibility for the team's project, communicating customers' needs. A project's development team executes tasks based on the Product Owner's timeline and priorities.

A Scrum cycle begins with the establishment of an initial project backlog, followed by a series of sprints. Each sprint starts with the creation of a project-backlog-derived backlog, followed by the sprint's work cycle proper. Work cycles feature daily short meetings, usually no more than fifteen minutes, to discuss progress and possible changes to the project. At the end of each sprint, the product backlog is updated to reflect the sprint's accomplishments and any client-requested modifications.

Scrum is best for small to medium size teams, since larger teams will find it difficult to keep the meetings brief and more difficult to manage (Matharu, Singh, Mishra, & Upadhyay, 2015).

In **Extreme Programming (XP)**, projects are developed in small teams. Customer satisfaction is ensured through continuous interaction with a team's customers and quick response to ongoing customer feedback (Sambare & Gulabchand, 2017). Customer requirements are represented with story cards that depict possible scenarios for product usage. Team developers use these cards to define the functions that drive the product design. The simplest design that provides the required functionality is initially created. Features are then added as needed at customer request. Other than an XP Coach who assists with organizing the process, the team shares project ownership. No roles are assigned; all developers share responsibility for the product.

XP emphasizes refactoring as a means of ensuring a code's simplicity and reliability. Thanks to refactoring and XP's emphasis on implementing customer feedback, code can change frequently. Developers write tests before they write code to insure their code functions properly. Following the initial testing of their codes, developers send codes to their customer for further testing and evaluation (Matharu, Singh, Mishra, & Upadhyay, 2015).

According to Sambare and Gulabchand, XP is a small to medium size team agile method that "[e]stablishes reasonable plans and schedules." (ibid.) Developers create the basic design first, then update code incrementally, following each addition with testing and refactoring.

One potential difficulty with continuous customer interaction is a need to extend deadlines, due to customers' requests for frequent changes. Particularly troublesome are complex changes that affect other functions. Such revisions can run afoul of time constraints and increase costs for the development team and their customer. XP can also incur high travel and increase labor costs due to the need for frequent customer visits to the development team's site (Palmer, 2009).

Test Driven Development (TDD), like XP, was introduced by Kent Beck (Anwer, Aftab, Waheed, & Wyed, 2017). The goal of TDD is to "write clean code that works," (Sambare & Gulabchand, 2017). With TDD, clean code starts with requirements to create or upgrade a code, including existing code and tests if upgrading.

Like XP, TDD requires tests to be written before logic is coded. TDD's "test first" approach to program development differentiates XP and TDD from most other methodologies, which fail to specify when tests should be written (Sambare & Gulabchand, 2017).

The TDD cycle consists of multiple iterations. Each iteration has seven phases: validate existing code, create tests for a new code, write that code, run the tests, make changes based on the tests, rerun the tests, and refactor the code. Each iteration starts by running any existing tests on any existing code to confirm that these tests pass. When they pass, developers write new tests for a proposed code. Ideally, these tests should account for functionality to be added in future development as well as the current iteration. Once a test is written, developers run the test to verify that it fails; if the test passes, either the proposed functionality is redundant, or the test is redundant or faulty. Once the test "successfully fails," the new code is written. Ideally, this code should do only enough to ensure that the test passes. Once their new code passes their new tests, developers rerun all tests to confirm that the additional code preserved their existing code's functionality (Sambare & Gulabchand, 2017).

After all requirements have been met and all tests have been passed, developers refactor their code to clarify it and remove duplication. This cycle continues until all features are added and their project is complete (ibid.).

TDD's advantages include its emphasis on early testing, which acts as a check on design and helps to locate errors early, and its development and use of regression suites - sets of tests to confirm the code's continued integrity following changes to the software or environment - to detect and address the reintroduction of errors. Potential disadvantages include the possibility of stubborn problems blocking further development and lack of testing skills by programmers who are more versed in writing code than tests (Anwer, Aftab, Waheed, & Wyed, 2017).

Behavior Driven Development (BDD) is a test-based methodology that adds a behavioral cycle to TDD. In BDD, testing is based not on implementations, but "scenarios: "shar[ed] expected behaviors across all members of the team" (Wilcox, n.d.).

BDD starts at an earlier stage of development than TDD. First, the development team meets with their client to discuss requirements. The discussion is framed as a series of questions and answers. The clients tell their developers what they want, and their developers ask how they want their code to respond. Together the developers and clients create a set of scenarios written in plain, non-technical text statements that both can understand (ibid.).

Whereas TDD tests are written by developers alone to validate code, BDD scenarios are written in collaboration with clients to ensure the correct features are created. The scenarios are tested and checked for misunderstandings and gaps before advancing to the test-writing phase of the TDD cycle (Nair, 2018).

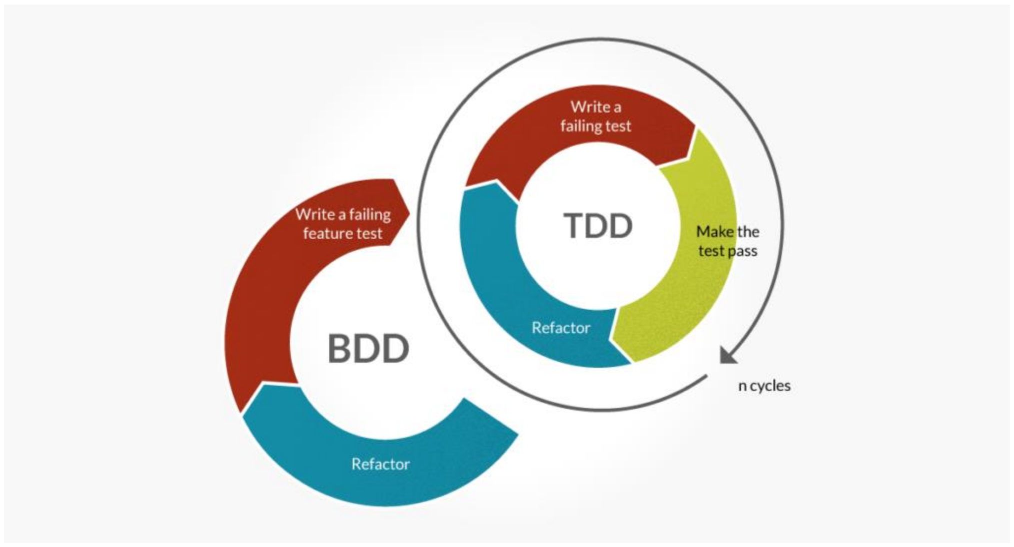

Figure 1. BDD works alongside TDD

Figure 1 shows BDD's relationship to TDD (ibid.). BDD enhances TDD by increasing communication between clients and developers. Collaboration between team and client allows requirements to be gathered from different perspectives and scenarios and tested by the client directly, requiring no translation from the developer. This differs from TDD, which requires developers to run tests and communicate the results to their client. Scenarios are easily rewritten if the test does not meet the clients' expectations (Wilcox, n.d.)

New developers can quickly learn BDD, although they would benefit from first learning TDD. A possible obstacle to using BDD is maintaining client communication, which is essential for modifying scenarios (Nair, 2018).

Kanban is "significantly gaining popularity in the software industry," (Matharu, Singh, Mishra, & Upadhyay, 2015). Toyota uses Kanban to manufacture vehicles on demand, termed "just in time" manufacturing. David J. Anderson adapted Kanban for software development, adopting the same principle to increase software development to flow and efficiency.

Kanban's primary focus is on scheduling to increase efficiency and productivity and reduce wasted work and time (Kanban Explained for Beginners, n.d.). Kanban has four basic principles:

- Start with what you do now. Kanban adds clarity to existing frameworks without major changes to the current process.
- Pursue incremental, evolutionary change. Small changes are encouraged and added slowly.
- Respect the current process, roles responsibilities and titles. Keep what's desirable and only change when desirable.
- Encourage leadership at all levels. This newest Kanban principle encourages all team members to have the mindset as a leader fostering collaboration (adapted from Kanban Explained, ibid).

Kanban methodology has six practices: visualize the workflow, limit work in progress, manage flow, make process policies explicit, use feedback loops to improve product and process quality, and improve collaboratively (ibid.).

Scheduling in Kanban is managed using the Kanban board, a flightboard-like visual representation of a project's workflow. The Kanban board organizes a project by displaying tasks with developer assignments for each task. It serves as a checklist to show progress and workflow. Project details are discussed by all team members.

Work is released in short iterations, tested, and delivered continuously to a project's customers. Customer feedback is used to adjust the Kanban board. All work and decisions are made as a team with no designated roles. Each team member provides a specialized skill and contributes to the board.

Kanban is responsive to change and can deliver product quickly due to continuous delivery. In order to assure Kanban's effectiveness, communication with customers should be maintained, so as to assure the board's currency and accuracy (Matharu, Singh, Mishra, & Upadhyay, 2015).

Feature Driven Development (FDD) was introduced in 1997 by Jeff De Luca. FDD, unlike Scrum, XP, and other small-team-oriented agile models, is suitable for complex, large-scale projects. FDD development is driven by a model of a project's features. Each feature should be scoped so that it can be implemented within two weeks. Features are developed in sequence, with each feature developed according to a five-phase model: "build up a general model, prepare a feature list, design by feature, outline by feature, and make by feature," (Anwer, Aftab, Waheed, & Wyed, 2017).

A team starts by designing a model of a project's features, with diagrams that show each feature's details and complexity. The feature's dependencies determine the order in which they will be developed. All details are reviewed as a community and an overall model is selected that suits customer needs. The model's key features are then organized into a feature list.

Features are organized by hierarchy starting with domain area, arranged into smaller feature sets. Once created, the feature list is presented to a project's customer for approval. Features are then assigned to developers based on developers' specialties and workloads. The feature list may be modified at this stage if workload is unbalanced causing dependent features to be created out of sequence. Next, a planning package is created, managed by a chief or central engineer, who collaborates with her team to create a bundle of diagrams and models, together with a plan for developing each feature. Each feature is then built in the sequence described in the planning package, progressed tracked by a feature chart. Each feature is tested, and refactored, and upon completion, is then incorporated into the main project (ibid.).

FDD-based development can potentially produce software quickly. Speed of development is enabled by FDD's allowing for as many teams as there are features, as well as its emphasis on producing features in short time frames. FDD requires effective task assignments and engineers who can handle responsibilities such as organizing planning bundles and overseeing projects. Part of guiding the team necessitates enforcing "Just Enough Design Initially (JEDI)" (Palmer, 2009), guiding the team to move on once a set of requirements has not been met (ibid.).

Dynamic System Development Model (DSDM) was developed in the United Kingdom in 1994 (Anwer, Aftab, Waheed, & Wyed, 2017). DSDM "combines the project management and product development related activities in a single process" (ibid.). It focuses on quick and quality software development while maintaining constant customer interaction. The "DSDM life cycle consists of six phases: Pre-project phase, Feasibility study, Business study, Functional model iteration, Design and build iteration, Implementation and Post project phase" (ibid.).

In DSDM's pre-project phase, the development team determines a project's scope, main team members, and cost. The project is then analyzed for feasibility and engineer availability, resulting in a feasibility report and a project plan. The project's needs and expectations are then discussed with its customer, elaborating on processes and requirements. The team then creates a function model for the project from information gathered in the previous step (ibid).

DSDM's function model helps a team to determine the order in which a project's functions should be developed and how they should be created. The model also supports the construction of prototypes that help the team analyze the functions' processes, verify the model's accuracy, and improve the overall project design. The finished prototype obtained from modeling is used to fabricate the deliverable. As features are implemented, the team's customer tests the implementation, providing feedback. This feedback prompts changes to the design.

After the customer approves these changes, the completed project is released to the customer. The team then creates manuals to assist users with their new software and provides training. In the final phase, the team conducts a survey to confirm their product performs to their customer's expectations (ibid.).

DSDM has up to fifteen different roles (i.e., developer, technical coordinator, advisor user), some of which require skill and experience in project organization. These requirements make DSDM more suitable for large-scale projects and companies (ibid.).

A common theme resonates across all agile models, with terms flexibility, frequent iterations, and continuous communication being the base chords. Agile methodology's flexibility extends outside the development process; Agile models are flexible and diverse enough for a variety of environments. FDD and DSDM are suitable for large teams, while XP and Scrum for suitable for small to medium teams.

Each model has defining characteristics that may appeal to certain environments. Kanban uses a Kanban board to help organize and track progress, Scrum releases iterations in sprints guided by the Scrum Master and Product Owner who ensure guidelines and timelines are followed, respectively. XP has shared project ownership and uses story cards to define functions. TDD writes tests before coding, and BDD writes scenarios to test behavior. FDD uses a model to organize features by hierarchy. DSDM is a defined process with assigned roles from start to finish, and Kanban can be started fresh on a new project or added to enhance an existing process.

An agile methodology would be suitable for a capstone environment. The capstone team would benefit from frequent iterations advancing through the project with basic working code. Errors can be addressed as they occur, rather than as part of a scavenger hunt through layers of code at a project’s end. Receiving frequent feedback helps the team adjust to mid-project changes. Open team collaboration would promote member involvement, each bringing value with her skillset. Frequent team meetings would help the team stay updated on all aspects of the project, and address issues and delays early.

When considering an agile model for a capstone environment, the capstone team needs to first consider the team size. The flexibility that a given methodology affords may determine the best methodology for a given project. Another consideration is the team members' individual skills, including a team member’s prior experience with specific methodologies. Some methodologies are complicated and learning them may take too much time. The team should assess the capstone's requirements and timeline and choose a methodology suitable for the environment and time constraints. For a small team in a capstone environment, the agile models most suitable are Scrum, XP, and also Kanban, which is adjustable to all size teams.

References

Anwer, F., Aftab, S., Waheed, U., & Wyed, S. M. (2017, March). Agile Software Development Models TDD, FDD, DSDM, and Crystal Methods: A Survey. *International Journal of Multidisciploinary Sciences and Engineering*, *8*(2), 1-10.

Eriksson, U. (2017, January 27th). *How to Make Your Regression Testing More Effective*. Retrieved July 31, 2018, from ReQtest: https://reqtest.com/testing-blog/how-to-make-your-regression-testing-effective/

Houston, D. X. (2014, May 26-28). Agility beyond Software Development. 65-69.

Kanban Explained for Beginners. (n.d.). Retrieved August 4, 2018, from kanbanize: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban/

Kuusinen, K., Gregory, P., Sharp, H., & Barroca, L. (2016, September 08-09). *Strategies for doing Agile in a non-Agile Environment.* Ciudad Real, Spain: ESEM '16 Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.

Matharu, G. S., Singh, H., Mishra, A., & Upadhyay, P. (2015, January). Empirical Study of Agile Software Development Methodologies: A Comparative Analysis. *ACM SIGSOFT, 40*(1), 1-6.

Meyer, B. (2018, March-April). Making Sense of Agile Methods. *IEEE Software*, 91-94.

Nair, J. (2018, April 18). *TDD vs BDD*. Retrieved August 7, 2018, from TestLodge: https://blog.testlodge.com/tdd-vs-bdd/

Palmer, S. (2009, November). Development, An Introduction to Feature-Driven. *Agile Zone*, 1-6.

Sambare, T., & Gulabchand, K. G. (2017, November-December). Agility: The need of an hour for software industry. *International Journal of Advanced Research in Computer Science, 8*(9), 41-46.

Sun, W., & Schmidt, C. (2018, March/April). Practitioners' Agile-Methodology User and Job Perceptions. *IEEE Software*, 52-61.

What is Agile methodology? . (n.d.). Retrieved August 2, 2018, from ISTQB Exam Certification: http://istqbexamcertification.com/what-is-agile-methodology-examples-when-to-use-it-advantages-and-disadvantages/

Wilcox, R. (n.d.). *Your Boss Won't Appreciate TDD: Try This Behavior-Driven Development Example*. Retrieved August 7, 2018, from Toptal: https://www.toptal.com/freelance/your-boss-won-t-appreciate-tdd-try-bdd