Agile Methodology

18

Abstract

This chapter discusses the Agile methodology which is a popular lightweight approach to software development. Agile provides opportunities to assess the direction of a project throughout the development lifecycle, and ongoing changes to requirements are considered normal in the Agile world. It has a strong collaborative style of working, and it advocates adaptive planning and evolutionary development.

Keywords

Sprints · Stand-up meeting · Scrum · Stories · Refactoring · Pair programming · Test-driven development · Continuous integration

18.1 Introduction

Agile is a popular lightweight software development methodology that provides opportunities to assess the direction of a project throughout the development lifecycle. There has been a growth in interest in lightweight software development methodologies since the 1990s, and these include approaches such as rapid application development (RAD), dynamic systems development method (DSDM) and extreme programming (XP). These approaches are referred to collectively as agile methods.

Every aspect of Agile development such as requirements and design is continuously revisited during the development, and the direction of the project is regularly evaluated. Agile focuses on rapid and frequent delivery of partial solutions developed in an iterative and incremental manner. Each partial solution is evaluated by the product owner, and the feedback is used to determine the next steps for the

G. O'Regan, *Concise Guide to Software Engineering*, Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-57750-0_18

project. Agile claims to be more responsive to customer needs than traditional methods such as the waterfall model, and its adherents believe that it results in:

- higher quality
- higher productivity
- faster time to market
- improved customer satisfaction.

It advocates adaptive planning, evolutionary development, early development, continuous improvement and a rapid response to change. The term "*agile*" was coined by Kent Beck and others in the Agile Manifesto in 2001 [1]. The traditional waterfall model is similar to a wide and slow-moving value stream, and halfway through the project 100% of the requirements are typically 50% done. However, 50% of the requirements are typically 100% done halfway through an agile project.

Agile has a strong collaborative style of working, and ongoing changes to requirements are considered normal in the Agile world. It argues that it is more realistic to change requirements regularly throughout the project, rather than attempting to define all of the requirements at the start of the project (as in the waterfall methodology). Agile includes controls to manage changes to the requirements, and good communication and early regular feedback is an essential part of the process.

A user story may be a new feature or a modification to an existing feature. The feature is reduced to the minimum scope that can deliver business value, and a feature may give rise to several stories. Stories often build upon other stories, and the entire software development lifecycle is employed for the implementation of each story. Stories are either done or not done (i.e. there is no such thing as 50% done), and the story is complete only when it passes its acceptance tests.

Scrum is an Agile method for managing iterative development, and it consists of an outline planning phase for the project, followed by a set of sprint cycles (where each cycle develops an increment). *Sprint planning* is performed before the start of the iteration, and stories are assigned to the iteration to fill the available time. Each Scrum sprint is of a fixed length (usually 2–4 weeks), and it develops an increment of the system.

The estimates for each story and their priority are determined, and the prioritized stories are assigned to the iteration. A short (usually 15 min) morning stand-up meeting is held daily during the iteration, and it is attended by the Scrum master, the project manager[1] and the project team. It discusses the progress made the previous day, problem reporting and tracking, and the work planned for the day ahead. A separate meeting is held for issues that require more detailed discussion.

Once the iteration is complete, the latest product increment is demonstrated to a review audience including the product owner. This is to receive feedback and to identify new requirements. The team also conducts a retrospective meeting to

[1]Agile teams are self-organizing, and small teams (team size <20 people) do not usually have a project manager role, and the Scrum master performs some light project management tasks.

identify what went well and what went poorly during the iteration, as part of continuous improvement for future iterations.

The planning for the next sprint then commences. The Scrum master is a facilitator who arranges the daily meetings and ensures that the Scrum process is followed. The role involves removing roadblocks so that the team can achieve their goals, and communicating with other stakeholders. Agile employs pair programming and a collaborative style of working with the philosophy that two heads are better than one. This allows multiple perspectives in decision-making which provides a broader understanding of the issues.

Software testing is very important in verifying that the software is fit for purpose, and Agile generally employs automated testing for unit, acceptance, performance and integration testing. Agile employs *test-driven development* with tests written before the code. The developers write code to make a test pass with ideally developers only coding against failing tests. This approach forces the developer to write testable code, as well as ensuring that the requirements are testable. Tests are run frequently with the goal of catching programming errors early. They are generally run on a separate build server to ensure that all the dependencies are checked. Tests are rerun before making a release.

Refactoring is employed in Agile as a design and coding practice. The objective is to change how the software is written without changing what it does. Refactoring is a tool for evolutionary design where the design is regularly evaluated, and improvements are implemented as they are identified. It helps in improving the maintainability and readability of the code and in reducing complexity. The automated test suite is essential in demonstrating that the integrity of the software is maintained following refactoring.

Continuous integration allows the system to be built with every change. Early and regular integration allows early feedback to be provided, and it also allows all of the automated tests to be run, thereby identifying problems earlier. The main philosophy and features of Agile are:

- Working software is more useful than presenting documents
- Direct interaction preferred over documentation
- Change is accepted as a normal part of life in the Agile world
- Customer involved throughout the project
- Demonstrate value early
- Feedback and adaptation employed in decision-making
- Aim is to achieve a narrow fast-flowing value stream
- User stories and sprints are employed
- A project is divided into iterations
- An iteration has a fixed length (i.e. time boxing is employed)
- Entire software development lifecycle is employed for implementation of the story
- Stories are either done or not done (no such thing as 50% done)
- Iterative and incremental development is employed
- Emphasis on quality

- Stand-up meetings held daily
- Rapid conversion of requirements into working functionality
- Delivery is made as early as possible.
- Maintenance is seen as part of the development process
- Refactoring and evolutionary design employed
- Continuous integration is employed
- Short cycle times
- Plan regularly
- Early decision-making.

Stories are prioritized based on a number of factors including:

- Business value of story
- Mitigation of risk
- Dependencies on other stories.

18.2 Scrum Methodology

Scrum is a framework for managing an Agile software development project. It is not a prescriptive methodology as such, and it relies on a self-organizing, cross-functional team to take the feature from idea to implementation. The cross-functional team includes the *product owner* who represents the interest of the users and ensures that the right product is built; the *Scrum master* is the coach for the team and helps the team to understand the Scrum process and to perform at the highest level, as well as performing some light project management activities such as project tracking, and the *team* itself who decide on which person should work on which tasks and so on.

The Scrum methodology breaks the software development for the project into a series of sprints, where each sprint is of fixed time duration of 2–4 weeks. There is a planning meeting at the start of the sprint where the team members determine the number of items/tasks that they can commit to, and then create a sprint backlog (*to do list*) of the tasks to be performed during the sprint. The Scrum team takes a small set of features from idea to coded and tested functionality that is integrated into the evolving product.

The team attends a daily stand-up meeting (usually of 15-min duration) where the progress of the previous day is discussed, as well as any obstacles to progress. The new functionality is demonstrated to the product owner and any other relevant stakeholders at the end of the sprint, and this may result in changes to the delivered functionality or the addition of new items to the product backlog. There is a sprint retrospective meeting to reflect on the sprint and to identify improvement opportunities.

The main deliverable produced using the Scrum framework is the *product itself*, and Scrum expects to build a properly tested product increment (in a shippable state) at the end of each sprint. The *product backlog* is another deliverable, and it is maintained and prioritized by the product owner. It is a complete list of the functionality (user stories) to be added to the product, and there is also the *sprint backlog* which is the list of functionality to be implemented in the sprint. Other deliverables are the *sprint burnout* and *release burnout* charts, which show the amount of work remaining in a sprint or release, and indicate the extent to which the sprint or release is on schedule.

The Scrum Master is the expert on the Agile process and acts as a coach to the team, thereby helping the team to achieve a high level of performance. The role differs from that of a project manager, as the Scrum Master does not assign tasks to individuals or provide day-to-day direction to the team. However, the Scrum master typically performs some light project management tasks.

Many of the traditional project manager responsibilities such as task assignment and day-to-day project decisions revert back to the team, and the responsibility for the scope and schedule trade-off goes to the product owner. The product owner creates and communicates a solid vision of the product and shares the vision through the product backlog. Larger Agile projects (team size > 20) will often have a dedicated project manager role.

18.3 User Stories

A *user story* is a short simple description of a feature written from the viewpoint of the user of the system. They are often written on index cards or sticky notes and arranged on walls or tables to facilitate discussion. This approach facilitates the discussion of the functionality rather than the written text.

A user story can be written at varying levels of detail, and a large detailed user story is known as an epic. An epic story is often too large to be implemented in one sprint, and such a story is often split into several smaller user stories.

It is the product owner's responsibility to ensure that a product backlog of user stories exist, but the product owner is not required to write all stories. In fact, anyone can write a user story, and each team member usually writes a user story during an Agile project. User stories are written throughout an Agile project, with a user story-writing workshop held at the beginning of the project. This leads to the product backlog that describes the functionality to be added during the project. Some of these will be epics, and these will need to be decomposed into smaller stories that will fit into the time boxed sprint. New user stories may be written at any time and added to the product backlog.

There is no requirements document as such in Agile, and the product backlog (i.e. the prioritized list of functionality of the product to be developed) is closest to the idea of a requirements document for a traditional project. However, the written part of a user story in Agile is incomplete until the discussion of that story takes

place. It is often useful to think of the written part of a story as a pointer to the real requirement, such as a diagram showing a workflow or the formula for a calculation.

18.4 Estimation in Agile

Planning poker is a popular consensus-based estimation technique often used in Agile, and it is used to estimate the effort required to implement a user story. The planning session starts with the product owner reading the user story or describing a feature to the estimators.

Each estimator holds a deck of planning poker cards with values like 0, 1, 2, 3, 5, 8, 13, 20, 40 and 100, where the values represent the units in which the team estimates. The estimators discuss the feature with the product owner, and when the discussion is fully complete and all questions answered, each estimator privately selects a card to reflect his or her estimate.

All cards are then revealed, and if all values are the same then that value is chosen as the estimate. Otherwise, the estimators discuss their estimates with the rationale for the highest and lowest discussed in detail. Each estimator then reselects an estimate card, and the process continues until consensus is achieved, or if consensus cannot be achieved the estimation of the particular item is deferred until more information is available.

The initial estimation session usually takes place after the initial product backlog is written. This session may take a number of days, and it is used to create the initial estimates of the size and scope of the project. Further estimation and planning sessions take place regularly during the project as user stories are added to the product backlog, and these will typically take place towards the end of the current sprint.

The advantage of the estimation process employed is that it brings multiple expert opinions from the cross-functional team together, and the experts justify their estimates in the detailed discussion. This helps to improve the estimation accuracy in the project.

18.5 Test-Driven Development

Test-driven development (TDD) is a software development process often employed in Agile. It was developed by Kent Beck and others as part of XP, and the developers focus on testing the requirements before writing the code. The application is written with testability in mind, and the developers must consider how to test the application in advance. Further, it ensures that test cases for every feature are written, and writing tests early help in gaining a deeper understanding of the requirements.

TDD is based on the transition of the requirements into a set of test cases, and the software is then written to pass the test cases. Another words, the TDD of a new feature begins with writing a suite of test cases based on the requirements for the feature, and the code for the feature is written to pass the test cases. This is a paradigm shift from traditional software engineering where the unit tests are written and executed after the code is written.

The tests are written for the new feature, and initially all tests fail as no code has been written, and so the first step is to write some code that enables the new test cases to pass. This new code may be imperfect (it will be improved later), but this is acceptable at this time as the only purpose is to pass the new test cases. The next step is to ensure that the new feature works with the existing features, and this involves executing all new and existing test cases.

This may involve modification of the source code to enable all of the tests to pass and to ensure that all features work correctly together. The final step is refactoring the code, and this involves cleaning up and restructuring the code, and improving its structure and readability. The test cases are rerun during the refactoring to ensure that the functionality is not altered in any way. The process repeats with the addition of each new feature.

Continuous integration allows the system to be built with every change, and this allows early feedback to be provided. It also allows all of the automated tests to be run, thereby ensuring that the new feature works with the existing functionality, and identifying problems earlier.

18.6 Pair Programming

Pair programming is an agile technique where two programmers work together at one computer. The author of the code is termed the *driver*, and the other programmer is termed the *observer* (or *navigator*), and is responsible for reviewing each line of written code. The observer also considers the strategic direction of the coding and proposes improvement suggestions and potential problems that may need to be addressed. The driver can focus on the implementation of the current task and use the observer as a safety net. The two programmers switch roles regularly during the development of the new functionality.

Pair programming requires more programming effort to develop code compared to programmers working individually. However, the resulting code is of higher quality, with fewer defects and a reduction in the cost of maintenance. Further, pair programming enables a better design solution to be created as more design alternatives are considered.

This is since two programmers are bringing different experiences to the problem, and they may have different ways of solving the problem. This leads them to explore a larger number of ways of solving the problem than an individual programmer. Finally, pair programming is good for knowledge sharing and learning,

and it allows knowledge to be shared on programming practice and design and allows knowledge about the system to be shared throughout the team.

18.7 Review Questions

1. What is Agile?
2. How does Agile differ from the waterfall model?
3. What is a user story?
4. Explain how estimation is done in Agile.
5. What is test-driven development?
6. Describe the Scrum methodology and the role of the Scrum Master.
7. Explain pair programming and describe its advantages.

18.8 Summary

This chapter gave a brief introduction to Agile, which is a popular lightweight software development methodology. Agile advocates adaptive planning, evolutionary development, early development, continuous improvement and a rapid response to change. The traditional waterfall model is similar to a wide and slow-moving value stream, and halfway through the project 100% of the requirements are typically 50% done. However, 50% of the requirements are typically 100% done halfway through an agile project.

Agile has a strong collaborative style of working, and ongoing changes to requirements are considered normal in the Agile world. It includes controls to manage changes to the requirements, and good communication and early regular feedback is an essential part of the process.

A story may be a new feature or a modification to an existing feature. It is reduced to the minimum scope that can deliver business value, and a feature may give rise to several stories. Stories often build upon other stories, and the entire software development lifecycle is employed for the implementation of each story. Stories are either done or not done, and the story is complete only when it passes its acceptance tests.

The Scrum approach is an Agile method for managing iterative development, and it consists of an outline planning phase for the project followed by a set of sprint cycles (where each cycle develops an increment). Each Scrum sprint is of a fixed length (usually 2–4 weeks), and it develops an increment of the system.

The estimates for each story and their priority are determined, and the prioritized stories are assigned to the iteration. A short (usually 15 min) morning stand-up meeting is held daily during the iteration and attended by the project manager and the project team. It discusses the progress made the previous day, problem reporting and tracking, and the work planned for the day ahead.

Once the iteration is complete, the latest product increment is demonstrated to a review audience including the product owner. This is to receive feedback and to identify new requirements. The team also conducts a retrospective meeting to identify what went well and what went poorly during the iteration, as part of continuous improvement for future sprints.

Reference

1. K. Beck et al., Manifesto for Agile Software Development. Agile Alliance (2001), http://agilemanifesto.org/