Software Development Lifecycle: 7-1 Agile Sprint Review and Retrospective

Walker Martin

Student, SNHU Online

CS-250: Software Development Lifecycle

Professor Jill Coddington

Aug 12, 2022

Software Development Lifecycle: 7-1 Agile Sprint Review and Retrospective

Throughout this course I've learned about the various stages of agile development and the agile team members within the scrum framework. From my experiences and from what I've learned from others I believe scrum to be the best suited agile method for most software development teams because of its flexibility and standardized practices such as events and roles. I've already worked with some of the events such as the daily scrum, sprint planning, customer input, testing, adapting to changes, and more. Now it's time for the final stages of the software development life cycle besides form maintenance; the sprint review and retrospective.

When first learning about the scrum agile methodology I was slightly confused about the difference between the sprint review and the sprint retrospective. Here's how I would explain the difference between the two. The sprint review typically occurs after the sprint and before the retrospective. The assignment guidelines described the sprint review to be more concerned about the "what" while the retrospective is prioritizing the "how". In an article comparing the sprint review and the retrospective (Palios, 2021) wrote about how the review is more focused on the product while the retrospective is more concerned with the process. This observation seems to be the major separating characteristic between the two, however in this essay they will be combined into one.

To initialize the scrum project the product owner and the scrum master first met with the customer to discuss the functionality of the application. The scrum master delivers the user stories to the rest of the scrum team who turn them into tasks. Product owners maintain the product backlog which consist of said tasks to be completed during the sprint. The team can create an online scrum board using whatever project management software they want such as azure. The developers can manage the tasks and categorize whichever way suits their project the

best such as by difficulty, duration or specialty. For instance, frontend tasks could be yellow, backend could be blue, machine learning could be red, cloud could be green, and data structures could be orange. Or the colors could be red is difficult, blue is easy and green is moderate. Organizing the tasks depends on the structure of the development team such as the size of the team and their specific skills. The product owner needs to keep an updated and precise product backlog in order for the team to properly develop the functionalities needed to make the customer happy.

The scrum approach truly allows for flexibility for changes in customer depends or to satisfy new user feedback. When the SNHU travel team received more feedback from the customers I was slightly annoyed because it changed the gameplay from what It was. However, you have to realize that it's not a bug, rather a functionality of the scrum framework. Which is why when being compared to the waterfall approach agile is superior based on its flexibility and time management. Though even if the product owner fails to fully deliver the user story to the developers or testers with adequate detail it is then the dev team's job to follow up to conform their task. Such as with the slideshow example, the tester reached out to the customer via email to ensure that their user stories met the needs of the customer despite lack of depth from the product owner. Which ensures appropriate test parameters for the developers code. However, the reason behind the lack of detail could be mentioned at the sprint retrospective in order to address the issue with the process that accounted for slight production delays before the future sprint. Without proper communications the scrum agile development approach doesn't run smoothly and diminishes its value over the waterfall approach.

In order for the user stories to be transformed into tasks the tester needs to dissect what the user is requesting then imagine how it will look. The tester breaks the story into tasks in a

scientific manner such that there will be no errors if the steps are followed fully. In order for the new sections of code to be implemented the first need unit testing then continuous integration testing. The use of test driven development seems to be very useful in developing code that meets the users demands. “Test-driven development is a somewhat widely used agile development practice. It is typically used in conjunction with unit testing by developers. Rather than writing some code first and then, as a second step, writing tests to validate the code, the developer actually starts by writing a test that the code needs to pass”(Charles, 2015). Unit testing could be used by developers to test an individual function. Then integration testing then takes the new function and implements it into the system as a whole, testing for every possible input to find any bugs that could cause major issues in production when the products further down the line. Which also leads to the concept of what done truly is.

Engineers tend to overdevelop code that only they can read just because one idea turns into another and they know the task can always be slightly better, but when is it done in terms of being good enough. Using test driven development in conjunction with project oriented development allows testers to develop tests that check for base conditions and if base conditions are met the program is good enough to be done. This is another aspect where communication is key between the developers and the testers. The concept of when it is done is interesting. In the project managers guide to mastering agile (Charles, 2015) wrote that the definition of done in agile development is a minimalistic shippable product. I tend to try and keep everything in my code minimal until I have a working product then I’ll add any additional details if there is enough time left in the schedule. Staying focused on test driven development combined with product drive development allows for the fastest possible shippable iteration since test driven development helps us steer clear of overengineering while project driven helps prioritize user

stories and the product backlog. Since one of the primary agile manifesto principles that I agree with is Deliver working software frequently. Similarly feature driven development helps accomplish similar goals. “Feature-Driven Development (FDD) is customer-centric, iterative, and incremental, with the goal of delivering tangible software results often and efficiently. FDD in Agile encourages status reporting at all levels, which helps to track progress and results”(Lynn, 2021). Which leads up back into communication.

One of the main tools used to increase communication is email. Which seems obvious, but besides from any direct message app like skype that your company may use it’s king. Besides the daily scrum which is only 15 minutes a day the team needs to communicate to each other frequently throughout the day. I’ve found email to be a simple and efficient tool to contact fellow employees, managers, and customers. It may sound simple but I’ve found many people for whatever reason don’t feel comfortable reaching out to others and instead make assumptions. A key rule to a successful product is to avoid making assumptions and rather confirm the tasks details with the necessary individuals. Especially as a manager if something is unclear you must speak up and request clarity. Which is why another agile manifesto principle I agree with fully is face-to-face conversations. In this age of remote interactions this may mean a zoom call which is still good for assuring the other person confirms your message. (Clare, 2017) “Whether you are starting the transition to a scrum process, or have been using scrum for a while, Zoom can help make your meetings more efficient and enjoyable for participants, regardless of where they are located”. Despite the author's clear bias I still happen to agree with her statement.

Overall the Scrum Agile approach was helpful for completing the SNHU Travel application. The journey mimicked a real software development scenario that experienced certain setbacks that the waterfall method couldn’t accurately account for. However, if the assignment in

whole was a single unwavering task then the waterfall method could have been more applicable. Management of uncertainty can be a defining aspect of whether or not to use agile practices. If the project has lots of variables that require estimation then you'll want to use agile. Agile can make estimations from multiple practices such as story points, planning poker, levels of estimation, and velocity determining burn charts. I would make the argument that even if your team's current project fits within the waterfall parameters you may want to begin making the switch to agile development anyway.

Resources

Charles G. Cobb. (2015). *The Project Manager's Guide to Mastering Agile : Principles and Practices for an Adaptive Approach*. Wiley.

Clare, S. (2017, October 5). *Using zoom for agile software development*. Zoom Blog. Retrieved August 14, 2022, from https://blog.zoom.us/using-zoom-agile-software-development/

Lynn, R. (2021, May 21). *What is FDD in agile?* Planview. Retrieved August 14, 2022, from https://www.planview.com/resources/articles/fdd-agile/

Palios, S. (2021, October 16). *Sprint reviews vs sprint retrospectives for Remote Teams*. Parabol. Retrieved August 13, 2022, from https://www.parabol.co/blog/sprint-reviews-vs-retrospectives/#:~:text=Differences%20between%20Sprint%20Reviews%20and,harmoniously%20and%20find%20flow%20together.