7-1 Final Project - Sprint Review and Retrospective

Joshua Kelly

Southern New Hampshire University

CS 250

Professor Konishi

8/17/2025

In the past sprint, I concentrated on delivering a truly "working software" increment for SNHU Travel and closing the manner in which we, as a team, collaborated. We maintained all of our interaction in the discussion board at our professor's behest. Our Professor, Haruka Konishi, instructed us to consolidate communication there so nothing got lost, and that pace worked well for following decisions and having a record we could refer to later. We also exchanged a single kickoff e-mail to validate roles. After that, everything existed in the thread. That framework defined the sprint, so I'm structuring this retrospective around how we utilized roles, completed user stories, dealt with interruptions, communicated, utilized organizational tools, and what I would retain or alter about the process.

I served as the Developer. My work centered on building a clean, runnable Java slideshow of “Top Destinations,” wiring the UI, and making sure the assets loaded reliably on different machines. I paired that build work with testable acceptance criteria that tied back to the user stories we defined earlier in the term. Jonathan took the Scrum Master role in our group discussion. He kept the conversation moving, asked clarifying questions, and suggested practices that would improve flow, like test driven development. Travis took on a product thinking lens in the thread. He challenged us to define “done” clearly and to refine the backlog, so we were working on the most useful slice first. When Travis asked for a precise definition of done, I wrote that a story is done when it meets all acceptance criteria, passes testing, is merged without issues, and is ready for release. That shared definition became our north star for review.

From that perspective, we closed the loop on a couple of user stories. Out of Module Three, we had stories for displaying a top list, filtering out results, and making selection clear to the user. As part of this sprint, I added a slideshow displaying five off-the-beaten-path destinations with consistent look and feel and one-click navigation. Smaller details did make a difference. I refactored the UI to make use of a CardLayout, included a clear title and description for each destination, and connected Previous and Next buttons in predictable cycle behavior. Content coalesced around Hallstatt, Chefchaouen, Ushuaia, Luang Prabang, and the Faroe Islands in keeping with our earlier decision to promote places that are not necessarily first places in mind. In the test side I got out my test cases from earlier in Module Four and ensured each one had a corresponding visible behavior in the slideshow. As an example, “each entry includes an image, short description, and link or navigation control” got translated into a concrete check against the label, the description pane, and functioning buttons. Holding those acceptance criteria as the target kept scope under control and made the sprint review a no-brainer: the element existed and worked, or it did not.

The largest interruption this sprint was environmental and, honestly, very real-world. Apporto froze after I attempted to upload a group of 4K photos, and I could not trust the virtual lab to create or execute anything for a bit. That roadblock could have thrown the sprint off track, but the Agile mindset prevented me from going off the rails. I changed horses and used my Mac, installed the Eclipse IDE locally, imported the project, and continued from there. That revealed a new issue: image assets were not loading due to differences in the paths and because the files were massive. I regularized the assets by resizing them to the same aspect ratio and storing them in a resources folder included in the project bundle. I also modified my codebase to use classpath resource loading instead of absolute filesystem paths so the slideshow would execute from a JAR. I confirmed the JAR in a terminal test, which conformed to the “ready for release without additional work” portion of our definition of done. None of that was in the plan, but the short feedback loops ensured the interruption just became the next thing to figure out in the current iteration. That is a good reminder that agility is just so much about handling the unexpected as it is about planning.

Our conversation was clear and concise. We published our initial proposals, responded to each other’s thoughts, and utilized the thread to maintain coordination. My key proposition was relying on pair programming and continuing integration in order to minimize bottlenecks and not having the work build up at the end. Jonathan suggested that test driven development would complement CI, since test first would provide us with consistent quality gates. Travis called for a more defined measure of done and for refinement of the backlog in order to keep everyone in sync. Those discussions were candid and straight, and they advanced the work. The thread mechanism provided excellent documentation of the decisions, and it kept the entire team in check since everything is in plain sight. Speed is the tradeoff. Asynchronous posting at times meant we had to wait a little longer for the following answer. I would, in the next sprint, retain the discussion board as the formal record and introduce a very short weekly check‑in on chat or video so the small roadblocks get resolved more quickly.

On the organization side, I tried to use Scrum ideas even though our team worked asynchronously. We had roles, a shared definition of done, acceptance criteria, and a working increment to demo. We essentially simulated backlog refinement in the thread by scoping the feature to an achievable slice. I treated each day as a tiny timebox: pick one acceptance criterion, implement it, run the app, and review. I also kept a lightweight checklist tied to the test cases from Module Four so I would know exactly what to verify before calling a story complete. None of this was fancy tooling, but it did the job. Research on agile teams backs up that simple, disciplined communication and focus on the work product are linked to better outcomes, especially when teams are distributed or asynchronous (Lindsjørn et al., 2016; Miller et al., 2021). I felt that in practice here. A small, visible plan and short feedback loops did more for progress than any big up‑front document could have. Considering the entire process, I believe Scrum was a good fit for this project. Scope changed in the middle of the sprint when I had to abandon Apporto, but the Agile process made it a trivial change and not a disaster. Public thread of the team provided us with transparency. Definition of done converted review comments into objective checks. Most significantly, we shipped something that runs. All the same, I would do two things differently. First, I would commit to test driven development from the onset. Writing a couple of tiny tests around image loading and navigating would have caught my initial resource path errors sooner, and I would have had greater confidence when I exported the JAR. Second, I would have one short, live touchpoint weekly. Fifteen minutes even would gain us the advantages of a daily standup at very marginal overhead. Those two adjustments would increase the quality of the teamwork and decrease the friction we encountered with purely asynchronous communication, the latter being in accordance with the literature regarding agile team performance and the practices of distributed teams in terms of communication (Lindsjørn et al., 2016; Miller et al., 2021).

For the sprint review itself, the demo is clear and transparent. The slideshow had a prominent title bar, shows each destination the same image and description, and the Next and Previous buttons cycle all the cards. There is no pause from massive images, and the JAR runs from the command prompt. That meets the acceptance criteria we had agreed upon and the definition of done that we had documented in the thread. Retrospective observation is that the clear roles, common definition of done, and small iterative increments served us well. Friction points were mainly environment and the rate of asynchronous responses. Both are solvable with the process refinements above. If we were going to schedule the very next sprint, I would retain the current increment and stretch it in tiny, testable increments: a toggle for captions, then a “More Info” link that opens a short details panel, then a lean filter for trip types. I would author the acceptance criteria first, write a few tests, and have a quick check‑in to agree on priorities before writing code. That schedule would maintain momentum while still having space to adapt to unexpected surprises, the practice I'd like to continue in life after this class.

**References**

Lindsjørn, Y., Sjøberg, D. I. K., Dingsøyr, T., Bergersen, G. R., & Dybå, T. (2016). Teamwork quality and project success in software development: A survey of agile development teams. Journal of Systems and Software, 122, 274–286. https://doi.org/10.1016/j.jss.2016.09.028

Miller, A., Sharp, H., Barroca, L., & Woodman, M. (2021). Communication in agile distributed software development: A systematic review. Information and Software Technology, 132, 106439. https://doi.org/10.1016/j.infsof.2020.106439