Strengths and Weaknesses of the Agile Process

We structured work into three short sprints with a backlog containing prioritized user stories. The major strength using the Agile process is the ease of identifying clear goals: requirement brainstorming turned rough feature ideas into user stories with acceptance criteria, reducing any ambiguity before coding. The backlog gave transparent prioritization of code modules to be implemented. The prioritization resulted in: student profile management -> student availability -> searching for other student availability -> planned schedules/sessions. Daily stand-up notes kept momentum by focusing on accomplishments and what needed to be accomplished, and also surfacing small issues early like requirement drift. An example of this occurred during sprint two where we developed functionality to schedule availability but did so without implementing a calendar. This was caught during a stand up discussion which was promptly added before finishing the sprint.

Intrinsic weakness of the agile process appeared due to the limitation of one-day sprints having no complete polished product. Not having a client to review the work led to an a very internal process of ensuring we were on track. This happened through using the backlog as a guide to ensure the sprints and standups were on track. And as such, our agile artifacts were not true to the Scrum nature: sprint reviews were mostly internal checks; stakeholder-style feedback (user demo) was absent, so “definition of done” drifted toward technical completeness rather than user value validation. Duplication and lack of integration tests required more awareness to accomplish, and improvement actions (shared normalization utility, CLI tests) were focused on by adding extra time to current sprints rather than appending to the backlog for future sprints, to ensure requirements were not modified.

Actual vs. Expected Outcomes

The actual vs expected outcomes did not vary significantly as we were not aware of the Scrum/agile process before starting this assignment. Any expected outcome would be due to thinking in a waterfall manner rather than an agile one. An example of this is writing the outlines for the backlog, sprint plans, stand up logs, and retrospective notes, as we did not have an expectation of how to structure them, but the actual outcome were well structured documents which guided our agile developmental process.

We expected the discussion portions to not take up much time, however due ChatGPT drastically speeding up application development, the discussion times took a significant portion in relation to the overall time.

Another expectation was for the overall process to be faster than it was, during class two groups on the waterfall side mentioned being halfway done thirty minutes after starting. However we did not reach the halfway point until an hour and a half in. We learned that the agile process may be less prone to rewriting large portions of code, however it may take more time overall due to having distinct sprints with discussion and ensuring following user stories was inplace before starting to code another module.

How ChatGPT & Gemini Influenced Requirements

The project theme was not influenced by ChatGPT and Gemini due to the general requirements. We along with the LLMs had much creative freedom to design the application as we wished. The only influence from LLMs on the Scrum-like requirements is developing what questions were asked during the different phases of development. The specific questions of ‘What are current plans’ or ‘What could be improved’ were chosen by Gemini, however since they align with the agile process it is not a negative influence.

Backlog/Story Writing: Gemini did not negatively influence requirements and accelerated drafting user stories (As a student, I can record availability…”) and good acceptance criteria (e.g. merging overlapping or adjacent slots). This reduced planning time and ambiguity significantly.

Sprint Planning: AI also helped split features into implementable slices and sequence dependencies (e.g. finalize profile model before availability). This encouraged optimistic estimates of progress by supplying ready-made scaffolds, compressing any perceived busy work and leaving little buffer.

Design: During implementation, ChatGPT nudged a service-layer architecture and dataclass usage, reinforcing separation of concerns—improving testability. Pattern replication was quick with AI guidance and good for consistency, but delayed abstraction since the immediate pattern “worked” for what was asked.

Coding: Parsing, helpers, and validation patterns were accelerated. Time parsing and slot merging logic benefited from AI proposing edge handling examples early (AM/PM, adjacency). Downsides of this were subtle assumptions that were passed into code without review or exploration by the team, Rather than pair deliberation a reliance on outputs was reflected.

Testing & Continuous Integration: GPT and Gemini promoted test-first for tricky logic like availability merging or overlap calculation. It provided representative negative test ides (invalid email, invalid course, or permission errors). However, like the prior requirement, a reliance on outputs without any review was reflected.

Net Impact: ChatGPT and Gemini acted as an accerlant for ideation, scaffolding, and unit-level correctness, enabling more story throughput in limited sprint time. Tradeoffs appeared mostly in reduced design thoughts or discussion and proper agile workflow experience.