**Explain the role of Agile in your project work, and contrast and compare this approach with alternatives from computing and other industries.**

Agile Software Development is a set of practices and processes designed to help software developers "create and respond to change in an uncertain and turbulent environment" (What is Agile Sofware Development?, 2018). In fields where static and rigid practices are often far from universally applicable, it provides a dynamic framework under which raw development is prioritised over rules that could limit its growth. The difference is much like the one between biological and mechanical systems: like biological systems, Agile development practices are self-correcting and responsive.

The quickest and easiest way to break down Agile development is by looking at its twelve principles, more explicit formulations of how the methodology can be applied in a software development environment. As we've worked on our project, we have tried to apply these principles in our decision-making and communication.

"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."

The first and most visible impact that Agile has had on our work is the way we have structured it. By dividing our understanding of our project into a number of individual and specific "stories", which each describe a specific use case in terms of our client's desires, our Agile structure achieves two things. First, it means that every feature our application will eventually provide satisfies our client, the party we're designing the application for in the first place. Second, this "story" structure segues nicely into the actual assignment of work. By treating each story as an individual feature to build independently of the others, multiple features can be built in parallel, scaling directly with the number of engineers available to work on stories. If any failures or delays should hinder the development of one feature, they will have no impact on any of the others.

"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."

Our story/ticketing structure also makes our client's requests for new features and requirements relatively pain-free. By digesting a request into a clearly worded story (and making sure with the client that this story captures what they were asking for), we can contain what can be large, amorphous tasks into a single feature that only a single developer needs to be concerned with until the time comes to integrate it with the rest of the application. This "waives" much of the cost associated with a shift in scope, while allowing the team and their customer to benefit from the benefits of these changes-- like being able to match a competing product on the market, or enhance the user experience.

"Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."

Being able to respond to changes quickly is partially a function of being able to produce content in general quickly, so this principle is quite critical to the essence of Agile. For our project, this has been enforced by the "sprint" style of production that we have been using, where engineers are given a number of features to complete within a set timeframe (in our case, two weeks). This has caused some stress for us, as we have also had to learn new technologies during these sprints. On the positive side, the successive series of deadlines has kept us productive and will continue to do so as our project evolves.

"Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."

Tying into our sprint model is the question of sustainability. This is a bit of a difficult one to talk about, since of the four sprints we've completed one has been rather non-standard (effectively a single feature of "learning how to use Laravel") and the other three have *escalated* in pace as we become more comfortable with this method of development. There's also the confounding factor of the other papers we're each taking this semester taking up a lot of our attention too. Combined, my impression is that a professional setting would be so different from our project on this count that it's hard to see any lessons to learn at all.

"Business people and developers must work together daily throughout the project."

Given that our team has no dedicated "business people" and our clients' time is taken up by several other commitments (including the other six or seven Software Engineering teams), it has been hard to schedule face time with "business people". However, in projects where both business people and developers are present, it is important that both parties are kept abreast of the others' activities and requirements. Software engineering is often a field where intuitive guesses are horribly wrong, so it is the responsibility of engineers to communicate well with their counterparts in business and resource management to give them a solid idea of what those resources will be able to achieve.

"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."

Moving on to teamwork practices that we *have* been able to demonstrate, face-to-face communication continues to be a strong point for us. Using the Agile SCRUM framework, we give each other consistent updates on our project status in person. This helps us keep the project at the back of our mind at all times, and gives us a sense of accountability for finishing our work in time for the end of each sprint.

"Working software is the primary measure of progress."

The main metric that we tend to report on in our SCRUMs is, of course, how well the features we've been assigned currently work. This is echoed in our end-of-sprint review period, when we show our clients all of the new (working) features we have added. The keyword here is "measure": code that produces no working result when it's executed is difficult to judge the merits of, especially for a client who may not even be capable of understanding it. Therefore, Agile developers should strive towards a plainly visible outcome, even if that means staggering feature releases or rushing the base functionality. Having it there is more important than getting it perfectly on its first iteration.

"Continuous attention to technical excellence and good design enhances agility."

Most of our less Agile moments have come of a lack of attention to technical excellence. Our first sprint was plagued by mistakes with Git, with three out of five members accidentally overwriting either their own work or the work of others. Although we were fortunate in that it happened on the least consequential sprint, when we were still learning the ropes of the GitHub workflow and so our workload was quite light relative to the last couple of sprints, this could definitely have caused problems in a project with higher stakes and more work to get through. Even though Git provided us with recovery methods for these mistakes, it's clear that more time would have been saved not making the mistake in the first place than making it and having to fix it-- and hence, it's clear that attention to technical excellence would have enhanced our agility.

"Simplicity--the art of maximizing the amount of work not done--is essential."

Another area that we struggled with in early sprints was intelligently reducing our workload. Some of this was caused by the aforementioned scrapes with Git, which took some work to resolve, but we also had some trouble fitting work to our existing skill sets. As students in a learning context, this was fine (and perhaps even ideal, since it gave our team members opportunities to learn new skills), but I can see how it could compromise a team's ability to develop sustainably in a professional context, especially because it's difficult to put a number on how much work learning a new technology will end up being.

"The best architectures, requirements, and designs emerge from self-organizing teams."

Coming back to principles that we've been able to apply more successfully, I feel that we have been able to leverage this strength of Agile development reasonably well. Throughout this project, we've organized ourselves quickly and decisively, both in terms of assigning features to individual team members and deciding on our overarching design architecture, which has recently become much more important with the introduction of Laravel to our project. We have routinely gone for noticeable, meaningful targets to reach on each sprint, and we have been able to meet those targets, although not always with the best time management.

"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

The ability for Agile teams to tune and adjust their behaviour is, in my opinion, the greatest strength of and argument for Agile development models. The self-correcting nature of Agile I described earlier isn't just useful because situations change, although that certainly is useful: it's important because no team starts off as effective as it can possibly be. Resources will start out misallocated, team members won't start off knowing exactly how to work with each other, and so on. It takes time and experience, and reflecting on that time and experience, to work out how to solve these problems. In our project, we noted a fairly universal problem with time management in all five of our team members early on, so we decided to make "early" hand-ins of features a priority. Similarly, our difficulties with handling Git on our first sprint made us more effective workers once we learned how to avoid those pitfalls.

Despite this essay's heavy focus on Agile process and practices, there are in fact other viable methods of developing software. One such method is "Spiral development", favoured by large, bureaucratic institutions that deal with highly complex and expensive physical assets, like NASA and national governments (Farr, p. 1). The key features of this development process, as distinct from Agile development, are as follows.

First, Spiral development puts heavy emphasis on "lowering development costs" (Farr, p. 2), both in terms of the time spent and in terms of cost. While this is a consideration for most teams that use Agile practices, Spiral development is more aggressive in shutting off avenues that have a low expected return, or a high ratio of risk to reward. To put it another way, if the Agile way of improvement consists of doing something and then reflecting about how it went afterwards, improvement under the Spiral paradigm consists of projecting how something is going to go first and then deciding whether or not to do it based on the projection. For most software development, where mistakes can easily be corrected, the Agile process is faster without much cost. In industries where a single unnoticed flaw can mean the loss of millions of dollars or even lives, the Spiral process wins out.

Second, there is a focus on risk and risk management. I touched on this when discussing the Spiral approach to cost reduction, but it extends much deeper than a different philosophy on improvement. In the Spiral paradigm, outcomes where the project fails entirely and produces nothing at all must be accounted for much more comprehensively than Agile development. After all, in Agile development every step of the process is expected to have a usable output, even in cases where the project is likely to peter out before too many of those steps are taken. Again, which model is better depends strictly on what is possible: a rocket can't be launched incrementally, so organizations trying to build one have to ensure that they're accounting for every single risk as they invest resources into it. It can't be tested until the time comes to launch it from the face of the earth, and many of the "bugs" that could interfere with the launch will render most of the rocket irrecoverable-- so it has to succeed.

Overall, though the two methodologies have clear strengths and weaknesses relative to each other, it's clear that Agile development is much better suited to the capabilities and requirements of software development.

# References

Farr, R. A. (n.d.). The Business Case for Spiral Development in Heavy-Lift Launch Vehicle Systems.

*What is Agile Sofware Development?* (2018). Retrieved from Agile Alliance: https://www.agilealliance.org/agile101/