Spring 2019

Analysis and Comparison of Multiple Approaches for Software Development Management as Applied to a Design Studio Project

Bethany Hage

University of Nebraska - Lincoln

ANALYSIS AND COMPARISON OF MULTIPLE APPROACHES FOR SOFTWARE DEVELOPMENT MANAGEMENT AS APPLIED TO A DESIGN STUDIO PROJECT

An Undergraduate Honors Thesis

Submitted in Partial Fulfillment of

University Honors Program Requirements

University of Nebraska-Lincoln

by

Bethany R. Hage, BS

Computer Science

College of Arts and Sciences

March 4, 2019

Faculty Advisors:

Stephen Cooper, PhD, Jeffrey S. Raikes School of Computer Science & Management

David Keck, PhD, Jeffrey S. Raikes School of Computer Science & Management

Abstract

This research analyzes multiple approaches to software development management through the lens of my experience in the Raikes Design Studio capstone program. The Design Studio project I participated in was a project for the company Hudl, and throughout its course we used techniques from the Agile framework of Scrum. I compared the Scrum principles to my team's own application over the course of the project, and I researched other software development methodologies such as Extreme Programming and Lean in order to determine whether they could improve the effectiveness of the current Design Studio experience. The proposed solution to this question is to continue using Scrum methods, but to also take inspiration from Extreme Programming and to encourage pair programming in all Design Studio teams, as well as to make an effort to improve Design Studio sponsor communication.

Keywords: Computer Science, Software Development, Agile, Design Studio, Software Project Management, Scrum, Lean, Kanban

Introduction

Design Studio is the capstone program within the Raikes School. The difference between Design Studio and Senior Design (i.e., the typical non-Raikes Computer Science capstone program) is that Design Studio is more geared towards introducing students to industry experience. Non-Raikes Honors students in Computer Science, such as myself, are allowed to apply and participate in Design Studio as "Associates."

Software engineering methodologies have been evolving throughout the years, from linear Waterfall development to the more flexible Agile development techniques (See Appendix A). Design Studio's methodologies are no exception; the program previously used Waterfall techniques but has since switched to Agile methods.

Through this thesis, I will examine the application of Scrum practices in Design Studio through the lens of my project experience, and I will attempt to evaluate the success of this methodology. In addition, I will propose a few simple improvements to Design Studio's unique implementation of Scrum. This proposal will be more of a recommendation than a prescription, because I'm aware that my project experience is just one of many and other teams may not have experienced the same issues over the course of the year.

Design Studio Project Background

The Design Studio project I worked on was for the company Hudl. Hudl is a sports technology company whose platform gives athletes the ability to review their performances and helps to them get recruited. According to their website, Hudl "offers the tools to edit and share video, interact with stats, and create quality highlight reels for entertainment and recruiting purposes" ("About Hudl", n.d., para. 1). Hudl has had nine projects in the Design Studio program, spanning the past thirteen years.

Each Design Studio project has a summary provided at the beginning of the year to serve as a description, as well as a way for students to make decisions about which team they are interested in joining. Hudl's summary for my team's project specified that they wanted to

> make it easier for athletes to build their own highlights and share it with their fans. … Athletes have to go through a lengthy process to find and create highlights from moments that they want to share. … We want you to focus on making this process as easy as possible for volleyball, soccer, and lacrosse teams ("Hudl design studio 2018/19", n.d., para. 4).

This summary was intentionally vague on the details of implementation because it was meant to serve as only a starting point. It instead placed an emphasis on auto-generating highlights and on cutting as many steps out of the process as possible.

As the project progressed, it became clear that Hudl's vision for the project was for our team to creatively come up with several solutions to improve the highlighting process for the aforementioned sports, build them out only to the extent that they could be tested with users, and gather statistical data on the viability of each approach. The end goal was to present this data and recommend a solution to Hudl that the company could then continue to pursue.

Methodology/Processes I Used in Design Studio

The methodology I used over the course of this project was Scrum (See Appendix A). The Design Studio program placed an emphasis on using Scrum methods, with the minutiae of how these processes were carried out being left to the individual teams. However, not everything was decided by the team. There were several mandated procedures that every team was expected to follow, starting with the organization of the team itself.

In accordance with Design Studio's typical procedures, our Design Studio teams were organized into hierarchies with Product Managers and Development Managers as the student leadership (see Figure 1). The rest of the teams, known as the developers, constituted the "Squads." There were Project Managers, Coaches and Tribe Leads who provided each team with guidance and ultimately issued grades in the course. Each team was required to meet for twelve to fifteen hours each week to work. Time spent working was called "team time."

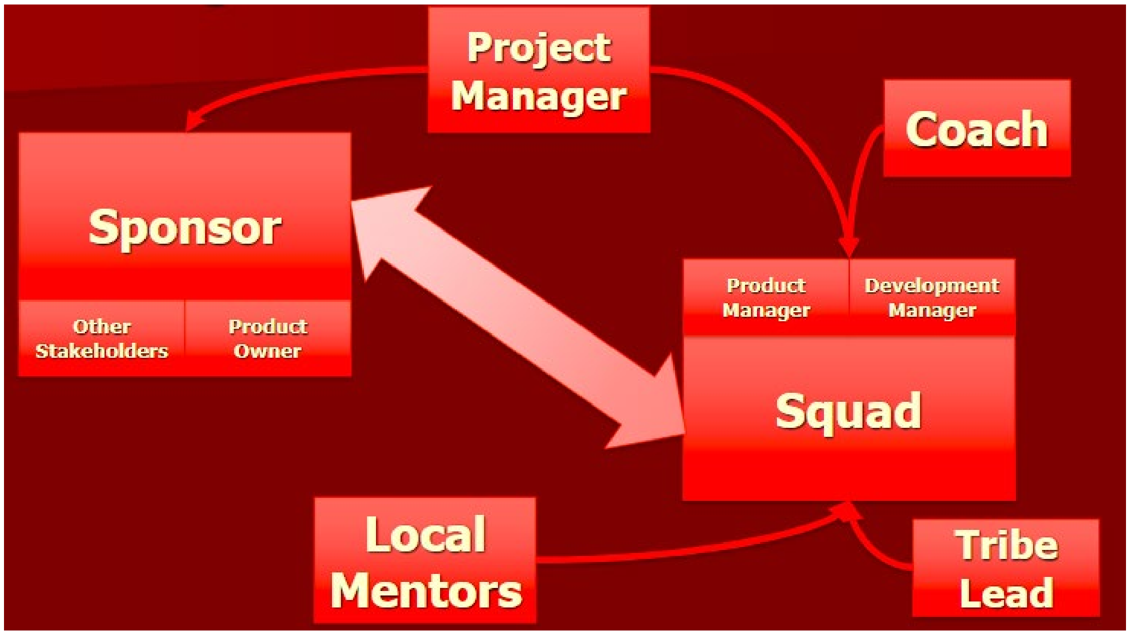


Figure 1. A diagram depicting the organizational structure of Design Studio, including participant roles and their relationships (Slide 17). Taken from: Antonson, Mark. (2018). Overview & Syllabus [PowerPoint slides]. Retrieved from *https://canvas.unl.edu/courses/45000/files/folder/Presentations?preview=2558982*

Each team was required to hold weekly update meetings with their sponsors, as well as three product releases each semester, or six in total. These "releases" occurred approximately once a month. Every team was expected to follow the same release schedule even though projects could be vastly different, because this schedule provided a sense of unity and structure.

Other than these guidelines, the teams were mostly free to choose their organizational methods, staying within the framework of Scrum but adapting specifics according to each project. Within the team, the Product Manager has the responsibility of making decisions on the details of the implementation. In my team, we followed a simple but effective plan which allowed us to be extremely Agile no matter what obstacles we faced.

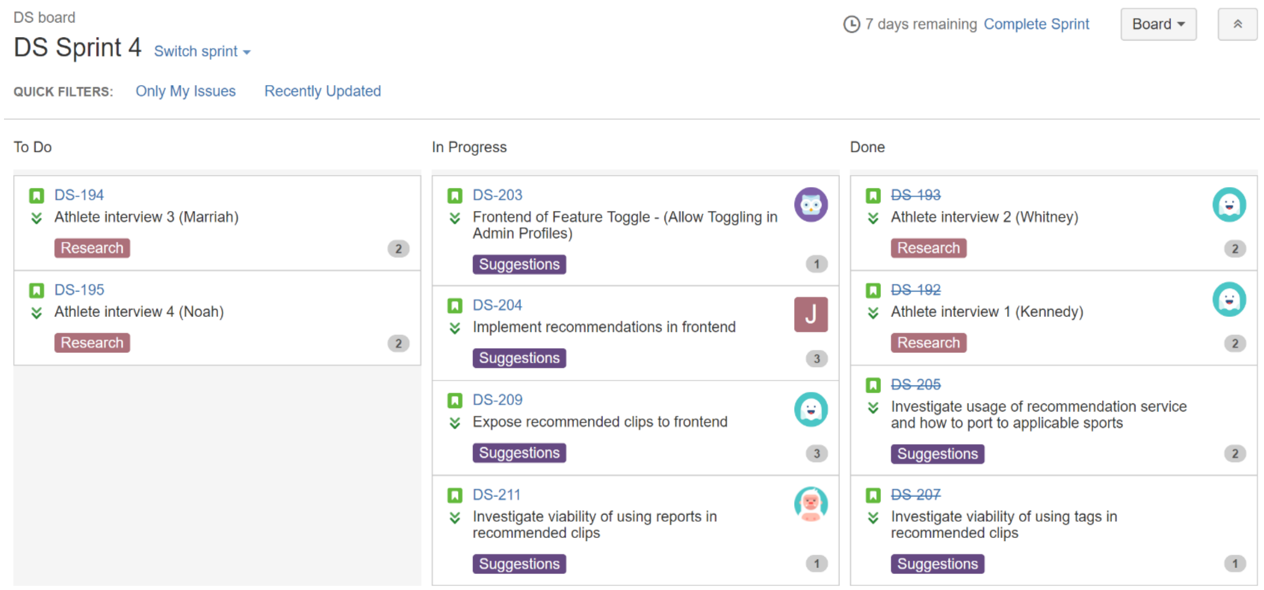
My team's version of the daily scrum was a daily stand-up. Each team time began with the daily stand-up. First, we each answered an interesting get-to-know-you question, ranging from topics such as "When you were a child, what did you want to be when you grew up?" to "What is your favorite brand of shoes?" or even, "What is your favorite song by the artist 'Pitbull'?" Next, each person described what they worked on the previous team time, and what they planned on working on once the stand-up session had adjourned. The stand-up was also a good time to raise any concerns or ask questions of the other team members.

Our next activities were sprint planning and estimation. Sprint planning and estimation took place at the beginning of each of our two-week sprints. Sprint planning involved identifying stories and allocating points to these tasks by using a simplified version of planning poker. We would

> briefly discuss each story and then each team member [simultaneously held] up their fingers with how many story points they [thought] the task [deserved] after a count of 3. If there [was] a discrepancy among the team, members with the "extreme" point value suggestions [stated] their cases and the story [was] repointed until a consensus [was] reached (Wiles, 2019a, para. 5).

The points were allocated in increments of Fibonacci numbers, with an added value of 1/2 points, suitable for any task which was deemed inconsequentially simple. This differed from the method usually recommended by Scrum - a lengthier process involving playing cards, deeper discussion, and more repetitions of the cycle (Rubin, 2013).

After allocating the story points, we tracked tasks using the tool Jira (See Figure 2). Most teams were encouraged by Design Studio to use a tool called Zenhub for this step, but Hudl provided Jira, so using Jira made the most sense for my team. We used Jira for sprint planning, story tracking, the product backlog, and burndown charts. Jira's tools gave each sprint their own board, with sections or lanes labeled "To-Do", "In-Progress", and "Done". Tasks were moved from one lane to another depending on our progress. Jira was able to track these changes and generate a burndown chart based on when tasks were completed. Burndown charts were particularly useful because "we [used] these charts to help understand what our typical velocity [was], which then [helped] us to plan how much work [could] be achieved in future sprints and releases" (Wiles, 2019a, para. 10).


DS board

DS Sprint 4 Switch sprint

7 days remaining Complete Sprint Board

QUICK FILTERS: Only My Issues Recently Updated

To Do

DS-194
Athlete interview 3 (Marriah)
Research 2

DS-195
Athlete interview 4 (Noah)
Research 2

In Progress

DS-203
Frontend of Feature Toggle - (Allow Toggling in Admin Profiles)
Suggestions 1

DS-204
Implement recommendations in frontend
Suggestions 3

DS-209
Expose recommended clips to frontend
Suggestions 3

DS-211
Investigate viability of using reports in recommended clips
Suggestions 1

Done

~~DS-193~~
Athlete interview 2 (Whitney)
Research 2

~~DS-192~~
Athlete interview 1 (Kennedy)
Research 2

~~DS-205~~
Investigate usage of recommendation service and how to port to applicable sports
Suggestions 2

~~DS-207~~
Investigate viability of using tags in recommended clips
Suggestions 1

Figure 2. View of our Jira board, depicting tasks in the 'To Do', 'In-Progress', and 'Done' categories. Allocated points are shown in the bottom-right hand corner of each task. Taken from: Wiles, B. (2019a). Design studio behaviors. Unpublished manuscript. Retrieved from https://unl.box.com/s/5ns780min9v5oxihmc14uhzghdzrou13

In terms of testing practices, we did have some amount of testing built into our project. Hudl's testing tools were available for us to run unit tests locally and remotely, and when we pushed code to a remote branch it was required by their continuous integration server to pass unit tests (Wiles, 2019a). Despite this, tests were more or less an afterthought, due to the nature of our project. If you recall, the project vision was for us to iteratively create solutions only to the extent that we could test them with users, so we operated out of the mindset that we were just gathering data. Writing tests wasn't our highest priority, and test-driven development wasn't something we considered at all, although it may have had its benefits. We knew anything we put into production would most likely be deprecated with the possibility of being rewritten in the future (Wiles, 2019a).

For telemetry and data analysis, the primary metrics my team was interested in testing involved highlight creation and the highlights that were being viewed. We were "fortunate to have a sponsor with extensive telemetry architecture. ... SumoLogic [was used to store] data from the past 30 days and re:dash [stored] longer-term data" (Wiles, 2019a, para. 9). We used these tools along with other services of Hudl's to create queries which measured the success of each of our product iterations. The queries returned data (e.g., the number of users utilizing our solutions) that allowed us to calculate the percentages of users that were funneling down each step of our project's flow (See Appendix C). It was important that we were able to obtain these metrics because the success of our approach depended on gathering this type of data and passing it along to Hudl at the end of the project (Wiles, 2019a).

Finally, for our release meetings, we met at Hudl headquarters with

> all team members, the sponsor, the technical sponsor, our tribe lead, our project manager, and ideally, our corporate sponsor. … This meeting typically [lasted] an hour and [discussed] the long-term plans of the project and what has occurred in the past release (Wiles, 2019a, para. 7).

We provided a deliverable at each of these meetings which consisted of a minimum viable product (MVP) or new feature and/or the results from user testing the last MVP. During these meetings, we also sought confirmation on the direction of the project in case we were heading in a direction the sponsor did not like. We took careful notes in case any important changes or decisions were made during the meeting (Wiles, 2019a).

Soon after each release meeting, my team went to a restaurant together to have a release retrospective in a casual setting. During dinner, we took out sticky notes and each wrote down two good things that occurred in the past release, two bad things, and two suggestions for how to improve the bad items. We read all the notes together and discussed them, and decided what our next steps could be.

There were a few ways in which our team's methods differed from traditional Scrum processes. I've already mentioned the changes we made to planning poker, but there were more significant ones as well. For example, we didn't have a designated ScrumMaster. Instead, we had a Product Manager. The differences between the two roles are subtle. A Product Manager traditionally has more authority over the team than a ScrumMaster in terms of the ability to dictate what they work on (Rubin, 2013). This structure suited us, however, because our Product Manager had many of the same duties as a ScrumMaster, and we didn't need another person to fill that role.

We also didn't do sprint reviews or sprint retrospectives. We had two-week sprints, but we found it more valuable to do a retrospective after each release. This retrospective covered a span of about a month, as opposed to sprint reviews/retrospectives which discuss the past two weeks. If we had done a sprint review for every sprint, it would not have been practical because it would have taken up valuable development time.

Results

The results of this project are largely qualitative. It is difficult to quantify the success of our methods because the success or failure of these management processes cannot necessarily be measured quantitatively. Due to the nature of the project, there would be nothing to compare it against, because that would require executing the same project multiple times concurrently using different processes. On another note, if we were only concerned with measuring the success of the project, which aimed to produce a software solution, it would be much simpler. Our team was successful according to these criteria because we produced several simple software solutions (See Appendix B) that we were able to put into production for a time, and we used metrics to test whether these software solutions made a significant impact for Hudl (See Appendix C). However, I want to focus more on the management process than the software produced. In any case, there were definitely areas in which we had more success than others.

For example, one of the things we did well during our process using the Scrum methods was that we were easily able to pivot and adapt our project when we realized we were heading down the wrong path. This occurred during Release 2, when we had a discussion with our sponsor one day and realized our concept of the project didn't match their vision. We thought we were going to be creating a fully-realized product that was ready for production, but Hudl wanted us to take a more iterative approach. In essence, they wanted us to research a few different concepts for how to improve the Highlights workflow and to deliver them statistics on which solution would be best for them to pursue in the future. After we had this discussion, our team was able to adjust our frame of mind and reevaluate our plans.

Although it was an unexpected circumstance, we were able to switch easily because we were following the Agile philosophy of responding to change over following a plan, and the

Scrum technique of iterative sprint planning. We weren't bound to the work we had already done. Additionally, after this switch, our sponsor was pleased with the new direction we were taking. In his comments on the end-of-release meeting for Release 2, our sponsor remarked,

> I think the team did a great job of taking a step back this release and really [looking] at what we are trying to do with the project. They did a much better job in pulling the right stakeholders to make sure that we adjusted the plan and that it was communicated effectively. [They] set some much better expectations for everyone else and I feel like we are headed in a great direction right now (Moore, 2018).

Another area in which our project succeeded was that our team was amiable towards one another and had a good balance of bonding time versus work time. Interpersonal relationships may only be tangentially related to the project, but they greatly affect the experience of working on a team, so it was valuable that we had built-in bonding activities to strengthen our team. For example, what we did for our Retrospective after each release was attend a team dinner together. We voted on what restaurant we wanted to dine at, and then we had a pleasant evening dining together and relaxing, while simultaneously discussing and reflecting on what had happened the past month or so.

Of course, there were several things with which my team struggled, one of which was communicating with our sponsor. This is what initially led to the predicament we found ourselves in during Release 2, when we had to pivot based on our understanding of the project. Even though we had weekly sponsor meetings and were at the Hudl headquarters for several hours each week, it felt like we seldom got down to talking about the things that were most important for our success. Specifically, these meetings were focused more on the big picture of the project, when more often than not we needed specific technical help. We would update the sponsors on what we worked on in the previous week, and they would comment on our progress, direction, and ask big-picture questions. Often, we waited until after the meeting to ask one of our sponsors about any specific programming questions. This also led to us needing to reach out to him and schedule a lot of meetings outside of team times (which unfortunately were not usually during business hours) and weekly sponsor meetings.

We also struggled with finding the right balance between estimation and work. In Release 3, we confidently started with the idea that we could produce another fully-realized MVP because we successfully created one within the span of Release 2. However, we failed to realize that the solution we were building was more complex this time and, furthermore, we had less work time to devote to these efforts due to Thanksgiving and winter break. We also had some unavoidable circumstances with team members getting sick during this release. It would have been wise if we had initially factored in the break time and allowed ourselves to build this solution over the span of two releases.

Most importantly, we struggled with dividing work and allocating tasks to those who could carry them out most effectively, particularly in the beginning of the project. For example, the amount of work needed was often underestimated and only one person was assigned to each task at a time. This led to developers having an overabundance of knowledge in one topic but no knowledge in others. As the project progressed, we learned from this problem and the situation improved as we increased our collaboration and pair programming.

Disadvantages of Scrum

Like any framework, Scrum has its shortcomings. Some of my team’s struggles were related to its faults. For example, we had a tendency to over-estimate how much work we could accomplish, particularly during Release 3. In Scrum, it is often difficult to estimate, particularly at the beginning of a project. Utilizing story points and planning poker allows a team to come to a consensus on task difficulty, which Scrum assumes approaches the reality of how much effort each task will require. In practice, it’s impossible to truly predict the future. The process of estimating effort can work well if the team has enough combined expertise. However, a team that is starting a project with unfamiliar technologies or a degree of uncertainty about their goals will struggle to estimate the time and effort that may be required.

Not only can it be difficult to estimate these items, but Scrum exacerbates this problem by placing a heavy emphasis on estimation. Some critics argue that Scrum focuses on the wrong idea in this regard, because the end goal of a project should always be to create high-quality software. One critic specifies that

> techniques like burn down charts and planning poker put the focus on your ability to accurately estimate task duration, not your ability to create software that is delightful to use, meets your users’ needs, keeps technical debt to a minimum, can be quickly and easily modified, [and] has a small number of bugs … unfortunately, far too many teams spend massive amounts of time and money just trying to measure their ability to guess how long work will take (Gray, 2015, para. 18).

The team’s resources might be better spent on development time.

The daily stand-up meeting is another Scrum practice which could be viewed negatively. It is usually touted as an innovative method to avoid long update meetings which waste the team’s time (See Appendix A). For instance, consider a different perspective in which a team member didn’t manage to accomplish anything significant the previous day or didn’t know what they needed to work on next. This team member could be reluctant to admit this in front of their entire project team. Alternatively, what would occur if someone oversold their accomplishments to make themselves seem like a team hero? In these scenarios the stand-up meeting adds unnecessary tension between team members.

Alternatives

To overcome these shortcomings and the aforementioned difficulties my team experienced during our project, I would like to discuss the possibility of using alternative methods in Design Studio.

The first alternative that comes to mind is Waterfall (See Appendix A). Most Waterfall methods have evolved into the Iterative and Incremental approach. Neither of these two methods are completely Agile. They use sequential development techniques which require planning to be done upfront (Poppendieck, M., & Poppendieck, T., 2003). However, it would be a waste of time to consider either of these options as a viable alternative to Design Studio's Scrum processes. For one thing, Design Studio previously used the Waterfall approach, and switched to Scrum for a good reason. Sequential development would not have been suited to the type of project I was working on because it requires all planning to occur before development. This would have led to failure in our case because our project was quite vague at the beginning and required us to change plans often. For this reason I would like to stay within the Agile realm.

Another alternative is Extreme Programming (XP) (See Appendix A). The only procedures associated with using XP that aren't already part of Design Studio's Scrum process are its encouragement of pair programming and mandated test-driven development. Pair programming would be ideal for remedying my team's problem of difficulty assigning tasks. It would encourage us to help one another and develop our understanding of code by sharing our knowledge with one another. The only foreseeable problems which could arise would be conflicts and personal differences which may occur when two team members don't get along.

Test-driven development would be helpful in its own way. It could help ensure quality programming by requiring the team to keep the end goal in mind, because tests are written before anything else. Unfortunately, extreme programming's emphasis on testing, while valuable in many cases, may not work for every team. Testing is an extremely important practice, but for Design Studio it really depends on the sponsor and the situation the team is placed in as to whether they need to implement testing or not. For example, the team may be creating a front-end for a project that has existing functionality, or simply researching MVP's, as my project did. However, if the team doesn't need to write tests, according to the creator of extreme programming, "You don't get to choose whether or not you will write tests—if you don't, you aren't extreme: end of discussion" (Beck, 2000, "What Is XP?", para. 2).

A third alternative is Lean (See Appendix A). Much like XP, Lean recommends utilizing pair programming. Lean recommends pair programming because it follows a couple of central principles of Lean, empowering the team members while also promoting learning over getting it right the first time (Poppendieck, M., & Poppendieck, T., 2003). Furthermore, Lean advocates that the downfalls of planning in Agile methods can actually be considered advantages. Agile approaches

> do not provide for design prior to the beginning of programming. … Those with a bias toward sequential development would like to see all design done prior to the start of programming. [Instead,] Agile approaches recognize that architectures evolve and mature; the practical approach is to provide for an emerging design rather than try to stop it (Poppendieck, M., & Poppendieck, T., 2003 "Chapter 5. Empower the Team, The Fuzzy Front End", para. 1).

Kanban (See Appendix A) is a simple procedure which could help to alleviate some of the difficulty planning and allocating tasks in Scrum. The good news is that the simple card system Kanban advocates is exactly like the configuration of my team's Jira board. This means we were essentially already using Kanban techniques. However, not all teams use Jira. As mentioned previously, many of them use Zenhub because this is the tool Design Studio encourages. Zenhub is adequate for most teams' needs, and has equivalent functionality to Jira, depending on how columns on its board are customized. If Zenhub were inadequate, other teams could take advantage of the index-card technique Kanban describes. This technique is simple, but effective, and would not require a large amount of extra resources for Design Studio to implement.

Conclusions

Based on my experience, I believe Design Studio's Scrum practices are on the whole adequate for its needs. The recommended practices effectively allow each team to be flexible and to develop its own practices within the framework. However, if I were to propose a couple of changes to Design Studio, it would be to heavily encourage pair programming and to increase sponsor communication.

To improve the problem we had with sponsor communication, it would have helped my project if we had a designated time to discuss technical aspects of the project. Perhaps Design Studio could add a second weekly meeting, or, if that requires too much extra time as an investment, teach the team leads to always bring up technical questions during the weekly meetings. Perhaps Design Studio could rework their focus on the Agile principle which states: "Business people and developers must work together daily throughout the project" (Beck et al., 2001a, para. 4). While this may not be possible for all projects (for example, those with remote sponsors), increased sponsor communication would help guide many teams.

Of course, if communication between the team and sponsor is increased too much, there could be a point where it becomes more detrimental than helpful. The sponsor could become overbearing with more frequent demands for the team, or become more aware of the team's daily struggles and have a negative view of how the project is going. More communication would also take away from development time. Design Studio should strive to find the correct balance. However, I think my suggestions will help without making this goal too overwhelming.

To enable us to better allocate tasks and learn from one another, Design Studio should also place a greater emphasis on pair programming. Design Studio already has a discussion of pair programming during one of the class periods. It is combined with another topic, but it takes up about half a class period. This discussion could be expanded into a full lecture. Another possibility is that an in-class hands-on activity could be added. Alternatively, Design Studio could add a pair programming assignment where each team has to have each a combination of developers collaborate together and report on the results.

If these changes were to be implemented, they would help solve the problems my team faced. For instance, these changes would have dealt with our difficulty in allocating tasks, in not having enough technical knowledge, and in the need to reevaluate our project vision.

Research Challenges

During the course of our project, my team faced a couple of research challenges that had nothing to do with our development methods or Scrum. One such challenge was that our development environment setup took a long time at the beginning of the project. Not being able to start hands-on coding for a month or so forced us to focus more on ideation during that phase of the project. We attempted to continually be productive by spending that time generating ideas for our solution while development was delayed.

Another challenge arose when our Development Manager had to drop out for personal reasons. This was particularly tough for our team because it occurred halfway through the year, and we suffered a loss of expertise about the project. To minimize our losses, we were able to promote a new Development Manager from within the existing team and to add a new developer. However, we still had to spend time and resources setting up the new developer’s development environment, as well as updating him on the technologies involved and the state of the project. For practically an entire release we had one less developer than usual. It didn’t seem to affect our progress too much, but then again it is impossible to say how much more we could have accomplished if we’d had another developer working during that time.

Glossary

Agile: A software project management philosophy which values responding to change over following a predetermined plan (See Appendix A) (Beck et. al., 2001b).

Burndown Chart: A graph depicting the number of Story Points completed in a Scrum Sprint over time. A visual representation of progress the team has made during the Sprint.

Daily Scrum: Also known as the **daily stand-up**. It is a short meeting limited to fifteen minutes during which the team members stand and answer the questions:

- What are the obstacles or impediments that are preventing me from making progress? (See Appendix A) (Rubin, 2013).
- What did I accomplish since the last daily scrum?
- What do I plan to work on by the next daily scrum?

Development Manager: Design Studio role. A student leadership position within the team itself. Responsible for guiding the more technical aspects of the project.

Extreme Programming: An Agile-like framework which preceded the Agile philosophy, and included specific implementation guidelines such as test-driven development (See Appendix A).

Incremental and Iterative: Software development processes which represent a more modern interpretation of Waterfall. They can be thought of as a mixture between Waterfall and Agile. They involve cycles of development with distinct linear stages within each cycle (See Appendix A).

Jira: A paid sprint planning tool for software development which can be accessed at: https://www.atlassian.com/software/jira

Kanban: A method of planning a software project by organizing cards into sections such as “To-Do”, “In-Progress”, and “Done”, and moving cards whenever a relevant change occurs (See Appendix A).

Lean: A set of principles that were originally applied to manufacturing, but have since been adopted and applied to software development (See Appendix A).

Minimum Viable Product (MVP): A product which produces the desired functionality with the minimum possible effort. Used in user testing, prototypes, etc.

Pair Programming: Pairs of developers work together, with one typing at the computer and the other providing comments and supervising. Allows developers to share expertise (See Appendix A).

Planning Poker: Method recommended by Scrum to allocate story points to each task. A game which enables the team to agree on the amount of points to assign (See Appendix A).

Product Manager: Design Studio role. A student leadership position within the team itself. Responsible for sponsor communication, team management, planning and organization.

Release: A deadline where a finalized unit of the project or some equivalent deliverable is presented.

Scrum: A commonly implemented set of software development rules which follow the Agile philosophy (See Appendix A).

ScrumMaster: A role recommended by Scrum (See Appendix A). “Helps everyone involved understand and embrace the Scrum values, principles, and practices” (Rubin, 2013, “Chapter 2. Scrum Framework, ScrumMaster”, para. 1).

Sponsor: Design Studio role. A representative from the participating company who is responsible for and involved in the Design Studio project.

Sprint Retrospective: Last activity in a Scrum Sprint. It allows the team to inspect and adapt their process, as opposed to the product (See Appendix A) (Rubin, 2013).

Sprint Review: Second-to-last activity in a Scrum Sprint; a conversation amongst all the relevant parties which allows the team to inspect and adapt the product that is being built (See Appendix A) (Rubin, 2013).

Sprint: A fixed period of time in which the steps of the Scrum cycle occur. Usually about two weeks (see Appendix A).

Story Points: Points allocated to tasks in Scrum which are used to estimate the task's difficulty. Not tied to any concrete measure, like time (See Appendix A).

Team Time: A Design Studio team's working hours.

Telemetry: In a software development context, telemetry refers to collecting data about the software's performance and usage.

Test-Driven Development: A method of software development wherein tests are written first, and code is developed to suit the test afterwards. Recommended by Extreme Programming (See Appendix A).

Waterfall: A software development process which has linear stages. Planning occurs before development. It was the most common process used before the advent of Agile (See Appendix A).

Zenhub: A free alternative to Jira; a sprint planning tool for software development which can be accessed at: https://www.zenhub.com/

Appendix A

Literature Review

Investigating alternatives to Scrum software development practices for Design Studio was one of the main goals of this research. To determine which set of practices would be best suited to Design Studio, I conducted a review of the literature. This research covered a large amount of material because of the large variety of software development philosophies. This section is dedicated to defining the terminology and concepts behind each practice. However, not all of the practices or terms will be discussed here. I've limited the scope to only the best alternatives I considered and to relevant concepts that I discuss in this thesis, with the aim of using this material as a reference for the previous sections of this thesis which discuss their feasibility.

Agile:

The word "Agile" is commonly used as an umbrella to cover a set of many different, yet similar, brands of software engineering practices. Because of this, it could easily be mistaken as referring to one of those methods individually. However, at its core, Agile is a philosophy rather than a strict set of rules and practices. This began as a set of principles called the Agile Manifesto, written by a group of software engineers to function as "a distillation of the ideas that had been circulating in the IT community, as developers searched for alternatives to the Waterfall methodology and its many pitfalls" (Freedman, 2017, "What is Agile?", para. 5). The Agile Manifesto is simply the following statement:

> We are uncovering better ways of developing
> software by doing it and helping others do it.
> Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right,
we value the items on the left more (Beck et. al., 2001b).

Later, the authors of the Agile Manifesto expanded upon these simple principles and created a document that described twelve Agile Principles, which were more specific and which many find to be a more useful guide than the Manifesto to implementing this practice (Freedman, 2017). These twelve principles are as follows:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly (Beck et al., 2001a).

Extreme Programming:

Many people confuse Agile and Extreme Programming. Extreme Programming actually predates the Agile Manifesto, and is one of the methodologies from which the Agile Manifesto evolved. Extreme Programming arose after the days of Waterfall and was developed by Kent Beck.

According to Beck, Extreme Programming's distinguishing qualities are:

- Its early, concrete, and continuing feedback from short cycles.
- Its incremental planning approach, which quickly comes up with an overall plan that is expected to evolve through the life of the project.
- Its ability to flexibly schedule the implementation of functionality, responding to changing business needs.
- Its reliance on automated tests written by programmers and customers to monitor the progress of development, to allow the system to evolve, and to catch defects early.

- Its reliance on oral communication, tests, and source code to communicate system structure and intent.
- Its reliance on an evolutionary design process that lasts as long as the system lasts.
- Its reliance on the close collaboration of programmers with ordinary skills.
- Its reliance on practices that work with both the short-term instincts of programmers and the long-term interests of the project. (2000, “What Is XP”, para. 1).

As you can see, this description mirrors the philosophy of Agile. Extreme Programming, however, prescribes more specific practices than Agile. For example, it emphasizes test-driven development and pair programming. A typical development cycle in Extreme Programming would include the following practices:

- Pairs of programmers program together.
- Development is driven by tests. You test first, then code. Until all the tests run, you aren't done. When all the tests run, and you can't think of any more tests that would break, you are done adding functionality.
- Pairs don't just make test cases run. They also evolve the design of the system. Changes aren't restricted to any particular area. Pairs add value to the analysis, design, implementation, and testing of the system. They add that value wherever the system needs it.
- Integration immediately follows development, including integration testing. (Beck, 2000, “Chapter 2. A Development Episode”, para. 20).

Scrum:

The Scrum framework is a commonly implemented version of Agile. Those who choose Scrum appreciate its flexibility and ability to adapt to suit each organization’s needs. Scrum “is based on a set of values, principles, and practices that provide the foundation to which [each] organization [adds] its unique implementation … for realizing the Scrum practices. The result [is] a version of Scrum that is [unique]" (Rubin, 2013, "Chapter 2. Scrum Framework, Overview", para. 1).

Interestingly, Scrum is able to achieve this flexibility while also providing a framework with a specific and clear set of practices. These practices can be divided into Roles, Activities, and Artifacts, as well as numerous specific Rules which I will not cover in this discussion (Rubin, 2013).

Roles:

The **product owner** is defined as being "the single authority responsible for deciding which features and functionality to build." In addition, he or she "maintains and communicates to all [outside] participants a clear vision of what the Scrum team is trying to achieve …[and] is responsible for the overall success of the solution" (Rubin, 2013, "Chapter 2. Scrum Framework, Product Owner", para. 1).

The **ScrumMaster** "helps everyone involved understand and embrace the Scrum values, principles, and practices" (Rubin, 2013, "Chapter 2. Scrum Framework, ScrumMaster", para. 1). He or she "is also responsible for protecting the team from outside interference … The ScrumMaster has no authority to exert control over the team, so this role is not the same as the traditional role of project manager or development manager" (Rubin, 2013, "Chapter 2. Scrum Framework, ScrumMaster", para. 2).

The **development team** "is a diverse, cross-functional collection of … people who are responsible for designing, building, and testing the desired product," also known as **developers** (Rubin, 2013, "Chapter 2. Scrum Framework, Development Team", para.

1).. They could include programmers, testers, UI designers, etc., but they are all termed "developers" by scrum. Ideally, the team is fairly small, around five to nine people (Rubin, 2013).

Activities:

Sprints are periods of time during which the following activities take place. Conceptually, sprints can be thought of in terms of iterations or cycles and should represent a fixed duration of time (Rubin, 2013).

Sprint planning is the process of preparing for the upcoming sprint by determining a sprint goal and breaking each goal into a set of tasks called the **sprint backlog** (Rubin, 2013).

The **daily scrum**, also known as a **daily stand-up**, is a short meeting limited to fifteen minutes during which the team members answer the questions:

- What did I accomplish since the last daily scrum?
- What do I plan to work on by the next daily scrum?
- What are the obstacles or impediments that are preventing me from making progress?

It is a common practice for the entire team to stand during this meeting to encourage brevity (Rubin, 2013).

The majority of the time spent in the sprint, while the team is completing tasks from the backlog, is called the **sprint execution** (Rubin, 2013).

The **sprint review** is the second-to-last activity in a sprint and is a conversation amongst all the relevant parties which is meant to allow the team to inspect and adapt the *product* that is being built (Rubin, 2013).

The **sprint retrospective** is the last activity at the end of the sprint. It allows the team to inspect and adapt their *process*, as opposed to the product (Rubin, 2013).

After the sprint cycle and before the next sprint, the process of "creating and refining product backlog items, estimating them, and prioritizing them" takes place and is called **product backlog grooming** (Rubin, 2013, "Chapter 2. Scrum Framework, Product Backlog", para. 3).

Artifacts:

The **product backlog** is a prioritized list of features. It begins as a list of requirements that are needed to meet the product owner's vision but evolves over the course of an ongoing project into a list that may contain new features, changes, improvements, etc. (Rubin, 2013).

A **sprint backlog** is the collection of tasks needed to complete a targeted feature within a sprint (Rubin, 2013).

The result of the sprint should be a **potentially shippable product increment** that meets the standards of the team according to their definition of done. The product doesn't necessarily have to be shipped, but there should be a state of confidence that the sprint's tasks are complete (Rubin, 2013).

Lean:

Lean originated as a set of principles that are applied to manufacturing, but it can also be applied to Agile software development. Mary and Tom Poppendieck are authors of a book which identified seven of these Lean principles and provided tools to implement them in a software development context (2003). These principles are as follows:

1. Eliminate waste.
    a. In a software development context, this means eliminating partially done work, extra processes, extra features, defects, and any other undesirable byproducts of the process (Poppendieck, M., & Poppendieck, T., 2003).
2. Amplify learning.
    a. Lean recommends focusing on learning effectively, rather than getting it right the first time. Take advantage of techniques such as learning cycles, iterations with refactoring, and increasing feedback (Poppendieck, M., & Poppendieck, T., 2003).
3. Decide as late as possible.
    a. In Agile software development, it is possible to create multiple options and delay making a decision on them until customer needs are better understood (Poppendieck, M., & Poppendieck, T., 2003).
    b. There is a distinction between plans and predictions, which are a good thing, and making irrevocable decisions based on speculation, which should be avoided (Poppendieck, M., & Poppendieck, T., 2003) .
4. Deliver as fast as possible.
    a. Do not rush projects, but learn to deliver rapidly to gain a competitive advantage and benefit the customer (Poppendieck, M., & Poppendieck, T., 2003).
    b. Complements the *decide as late as possible* principle.
    c. Pull systems such as Kanban are more effective than strict software development schedules in complex environments with even a slight amount of variability (Poppendieck, M., & Poppendieck, T., 2003).

5. Empower the team.
    a. Attempt to provide intrinsic motivation to the team. This involves defining a clear and compelling purpose, that is achievable, and giving the team access to the customers (Poppendieck, M., & Poppendieck, T., 2003).
    b. Encourage pair programming and design reviews because they provide expertise sharing between team members. However, design reviews should focus on learning rather than criticizing mistakes (Poppendieck, M., & Poppendieck, T., 2003).
6. Build integrity in.
    a. Strive to build a system with high perceived and conceptual integrity by having detailed information flows from customer to development team as well as among the development team itself (Poppendieck, M., & Poppendieck, T., 2003).
    b. Perceived integrity: affected by the customer's whole experience of a system. Maintain institutional knowledge about the system with a suite of automated test and good documentation (Poppendieck, M., & Poppendieck, T., 2003).
    c. Conceptual integrity: System's central concepts are cohesive and work as a whole. They have an effective architecture. Strive for simplicity, clarity, suitability for use, no repetition or extra features (Poppendieck, M., & Poppendieck, T., 2003).
7. See the whole.
    a. Utilize systems thinking. It is important to realize that

        > a system is not just the sum of its parts—it is the product of their interactions. The best parts do not necessarily make the best system; the

ability of a system to achieve its purpose depends on how well the parts work together, not just how well they perform individually (Poppendieck, M., & Poppendieck, T., 2003, "Chapter 7. See the Whole, Systems Thinking", para. 1).

b. Avoid fixing scope in detail, so you can be flexible. (Poppendieck, M., & Poppendieck, T., 2003).

Kanban:

Kanban is a "pull scheduling" system of using cards to dictate every step of a process. It was "originally patterned after restocking grocery store shelves. Kanban means sign or placard in Japanese" (Poppendieck, M., & Poppendieck, T., 2003). In a more general context, which may be applied to software engineering,

Kanban advocates that you:

- Visualize how the work flows through the system (for example, the steps that the support organization takes to resolve a support request)
- Limit the work in process (WIP) at each step to ensure that you are not doing more work than you have the capacity to do
- Measure and optimize the flow of the work through the system to make continuous improvements (Rubin, 2013, "Chapter 1. Introduction, Interrupt-Driven Work", para. 3).

As a software engineering implementation of Kanban, index cards can be created for each task and organized into three categories on a centrally visible bulletin board: "To-do," "In-progress," and "Done." This approach can enable developers to visualize exactly what to work on at a glance (Poppendieck, M., & Poppendieck, T., 2003).

The Lean methodology mentioned delivering “as fast as possible” as one of its principles. Kanban “is the enabling mechanism of [this principle]. [In the grocery store example, it was] the thing that [told] people and machines what to do from hour to hour in order to achieve optimum plant output” (Poppendieck, M., & Poppendieck, T., 2003, “Chapter 4. Deliver as Late as Possible, Manufacturing Schedules”, para. 7).

Kanban is a beneficial system because it empowers the team to regulate their own work. For example, in the

> pre-lean days, people were told what to do by managers who modified the … schedule based on their personal knowledge and decided what each workstation should do. … The interesting thing about pull scheduling is that it takes the manager out of the loop of having to tell workers what to do. The work is self-directing. The managers spend their time coaching the team (Poppendieck, M., & Poppendieck, T., 2003, “Chapter 4. Deliver as Late as Possible, Manufacturing Schedules”, para. 8).

Waterfall:

Waterfall does not follow the Agile philosophy. It was the traditional method used before the advent of Agile, and it is linear in nature. It

> is predicated on a sequential approach to change whereby development is linear, with outputs of each phase of an analysis (requirements), the design, build, and test and deploy development process cascading like a waterfall into the next phase upon completion (Davis, B., & Radford, D., 2014, “What is Waterfall?”, para. 2).

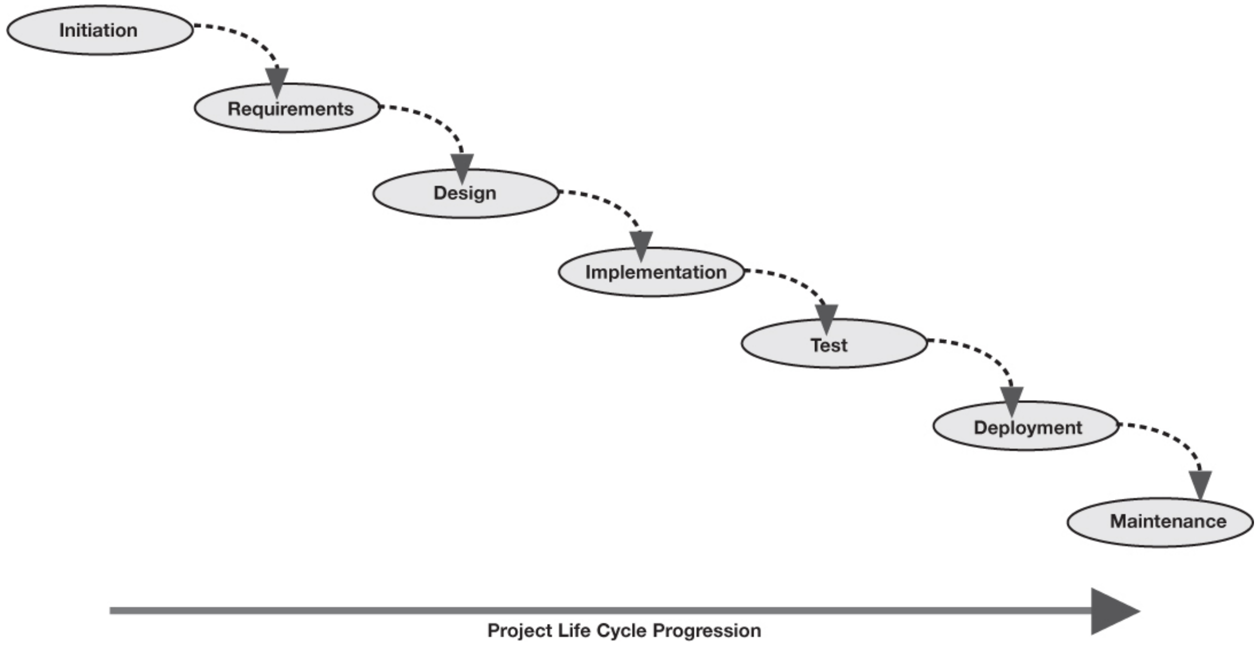


Figure 3. Stages of the Project Life Cycle in Waterfall Development. Taken from: Davis, B., & Radford, D. (2014). Going beyond the waterfall: Managing scope effectively across the project life cycle. Plantation, FL; 4: J. Ross Publishing. Retrieved from https://www.safaribooksonline.com/library/view/-/

Waterfall used to be popular "primarily because it is simple. It is easy to understand and follow, and therefore adopts a logical sequenced approach" (Davis, B., & Radford, D., 2014, "What is Waterfall?", para. 3). Most of the project life cycle time is spent in various forms of planning. Implementation only begins after every aspect of the project has been planned in advance. According to Davis and Radford, "with time spent up front defining and ultimately fixing the scope, a key principle of Waterfall is that once fixed, the development machine will progress and take that agreed scope and make it real" (2014, "Impacts of Waterfall on Scope", para. 2).

Much of the reason Waterfall has not been in use much anymore is that, due to the nature of fixing scope before actually implementing, there is no room for error or unexpected circumstances. If something unexpected does happen, it disrupts the entire planned project timeline.

Incremental/Iterative:

Iterative and incremental development can be thought of as a halfway point between Waterfall and Agile. It uses aspects of both, with each increment or iteration taking place in its own cycle, much like Agile, but following a similar pattern to Waterfall within each iteration.

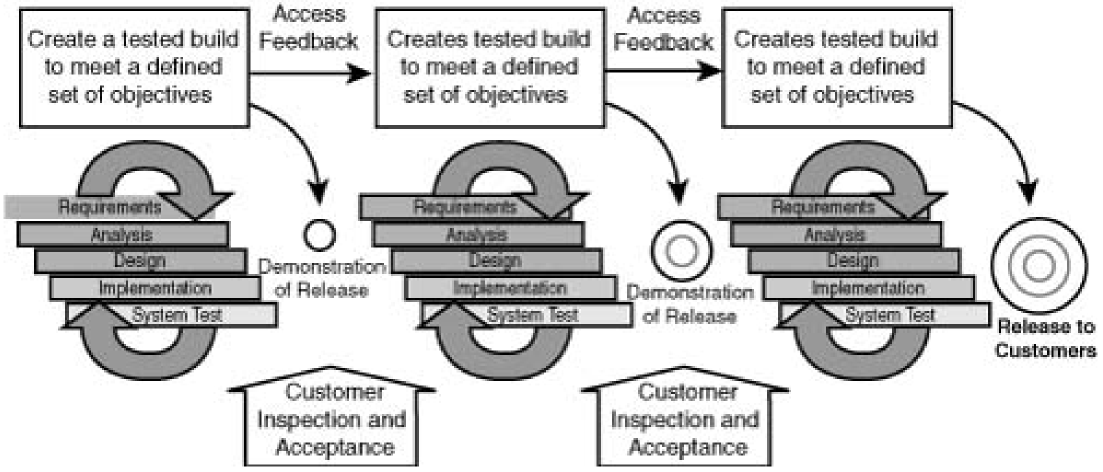

Create a tested build to meet a defined set of objectives
Access Feedback
Creates tested build to meet a defined set of objectives
Access Feedback
Creates tested build to meet a defined set of objectives
Requirements
Analysis
Design
Implementation
System Test
Demonstration of Release
Demonstration of Release
Release to Customers
Customer Inspection and Acceptance
Customer Inspection and Acceptance

Figure 4. Iterative and Incremental project development cycle from the Project Manager perspective. Taken from: Bittner, K., & Spence, I. (2006). Managing iterative software development projects. Upper Saddle River, NJ: Addison-Wesley. Retrieved from https://www.safaribooksonline.com/library/view/-/ 032126889X/?ar

Iterative and incremental development has the following characteristics:

- It involves the iterative application of a set of activities to evaluate a set of assertions, resolve a set of risks, accomplish a set of development objectives, and incrementally produce and refine an effective solution.
- It is iterative in that it involves the successive refinement of the understanding of the problem, the solution's definition, and the solution's implementation by the repetitive application of the core development activities.

- It is incremental in that each pass through the iterative cycle grows the understanding of the problem and the capability offered by the solution.
- Multiple applications of the iterative cycle are sequentially arranged to compose a project (Bittner, K., & Spence, I., 2006, “Iterating and the Scientific Method”, para. 5).

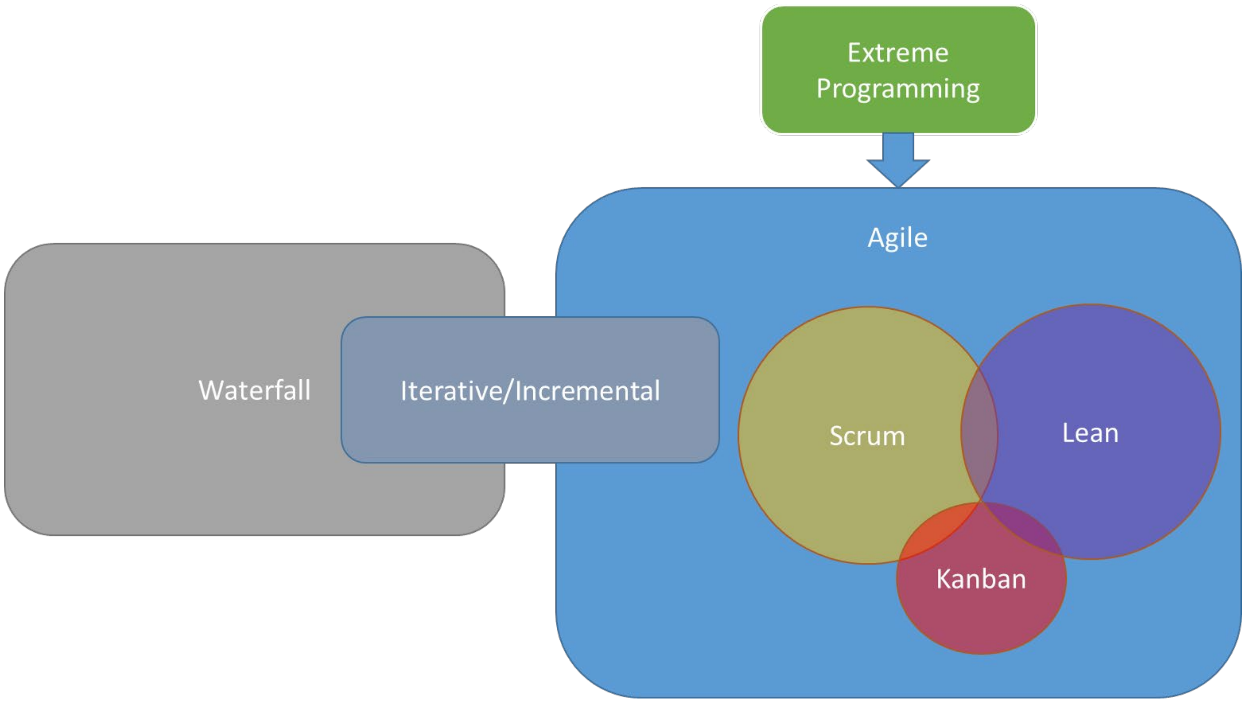


Figure 5. Graphical representation of relationships between software development methodologies described in this Appendix.

Appendix B

Software Solutions

This section provides an overview of the MVP software solutions we identified and implemented for Hudl over the course of the project:

Suggestions

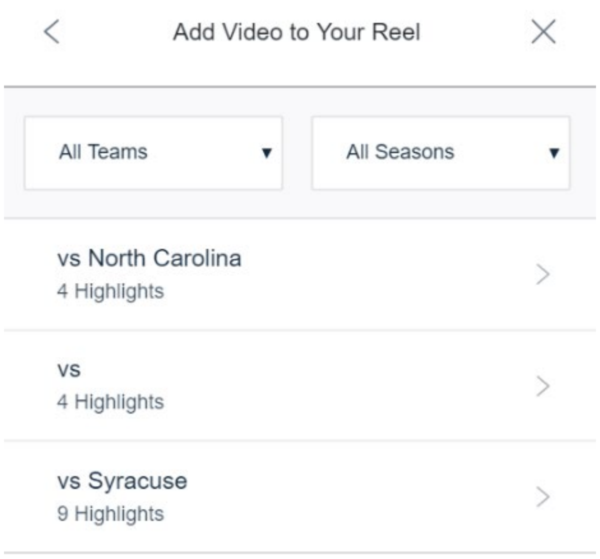


Figure 6. Hudl Highlights editor sidebar **before** implementation of the Suggestions menu.

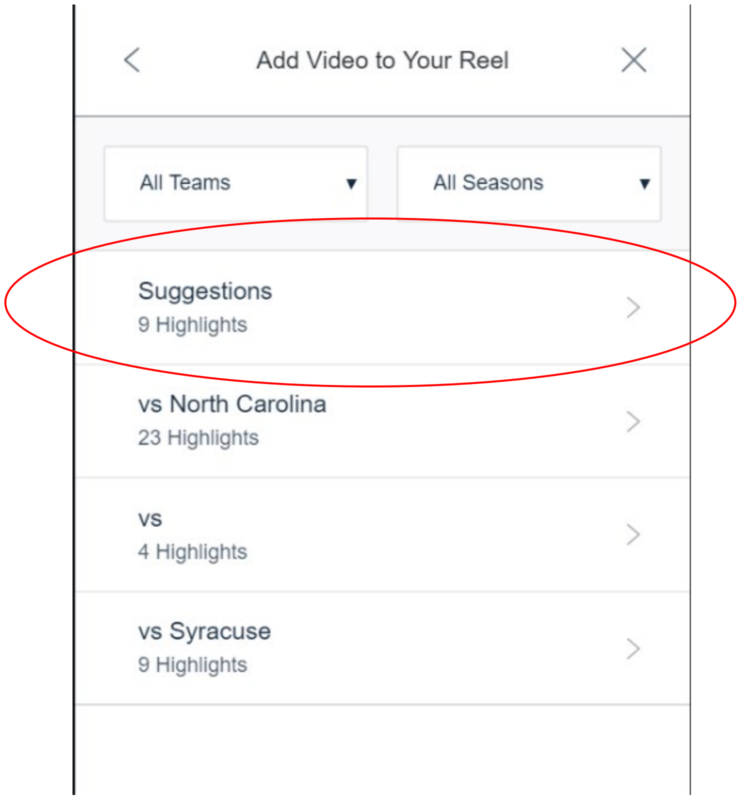
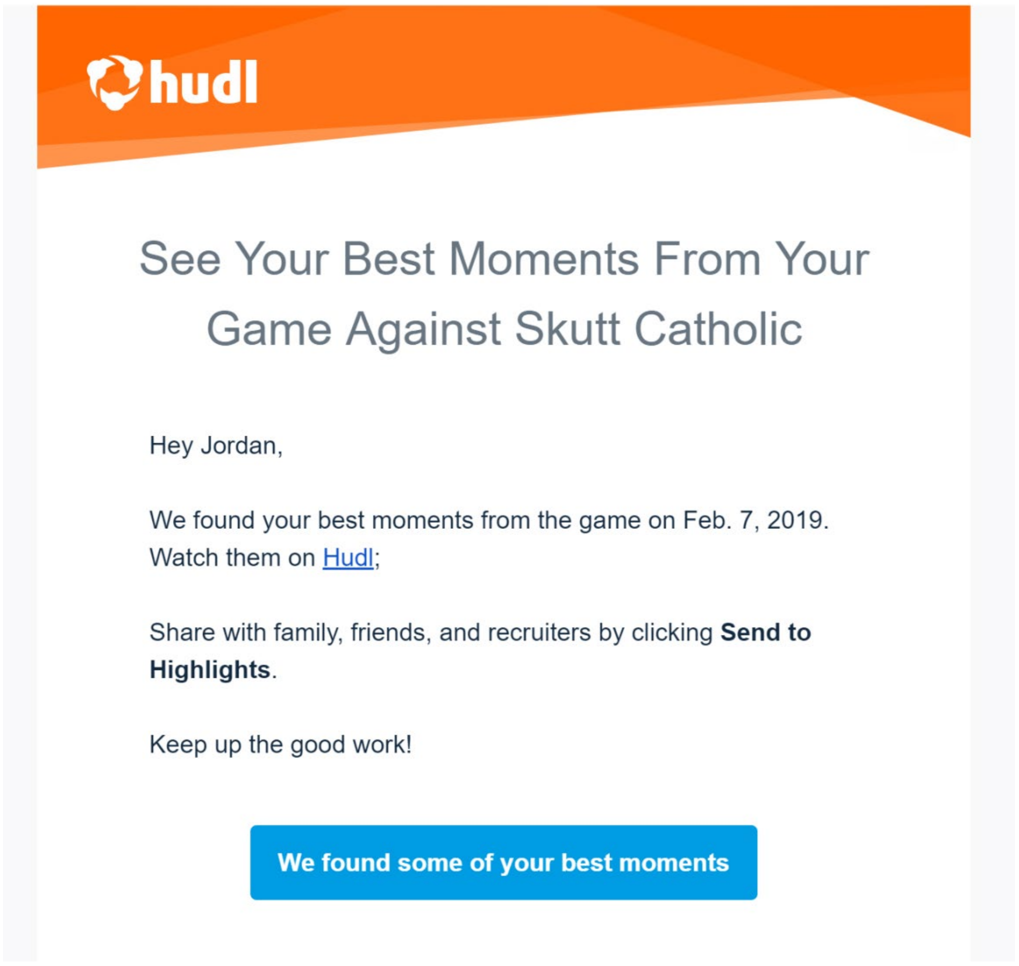


Figure 7. Hudl Highlights editor sidebar with the Suggestions option added.

- In the Highlights editor, users are able to open a menu which suggests their most recent 10 clips for them to use in their highlight.
- Results of A/B testing this solution led us to not recommend pursuing it further (See Appendix C).

Notify and Filter



See Your Best Moments From Your Game Against Skutt Catholic

Hey Jordan,

We found your best moments from the game on Feb. 7, 2019. Watch them on Hudl;

Share with family, friends, and recruiters by clicking **Send to Highlights**.

Keep up the good work!

Figure 8. Email notification sent to athletes after their gameplay video is fully tagged with game data by a coach. The link takes athletes to a pre-filtered video on Hudl with all of their best gameplay moments selected for them.

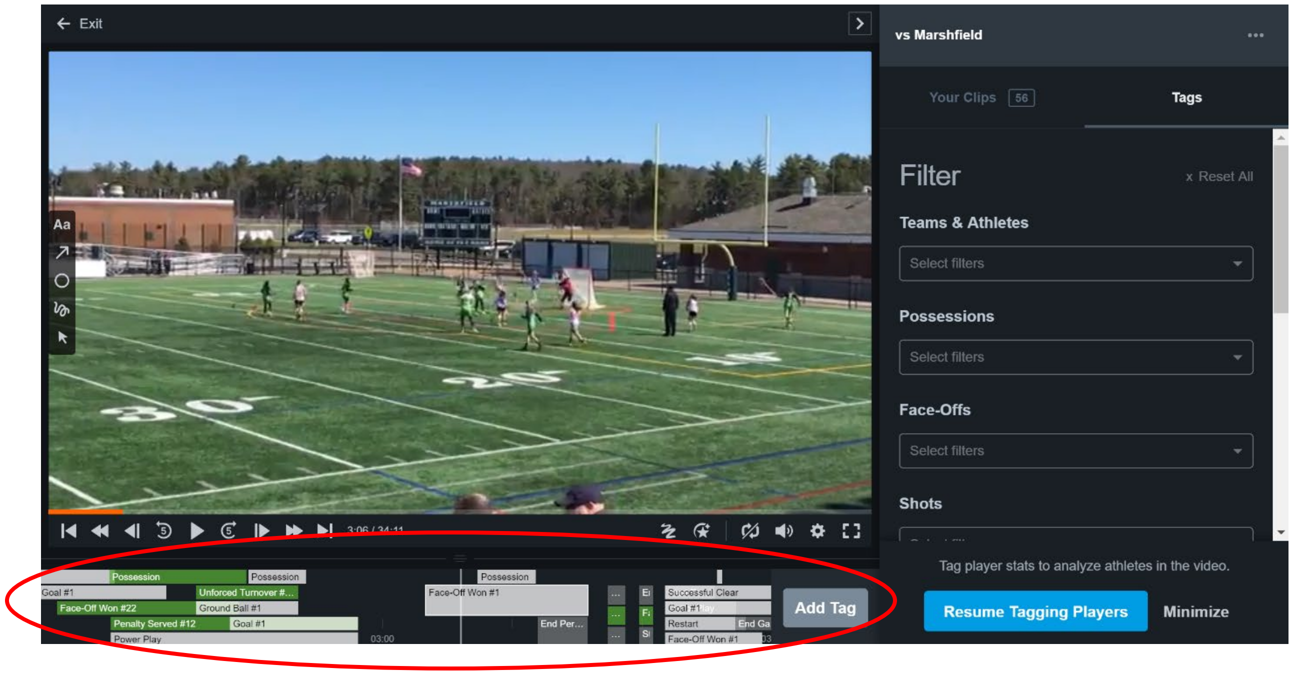


Figure 9. Hudl video player before filtering tags.

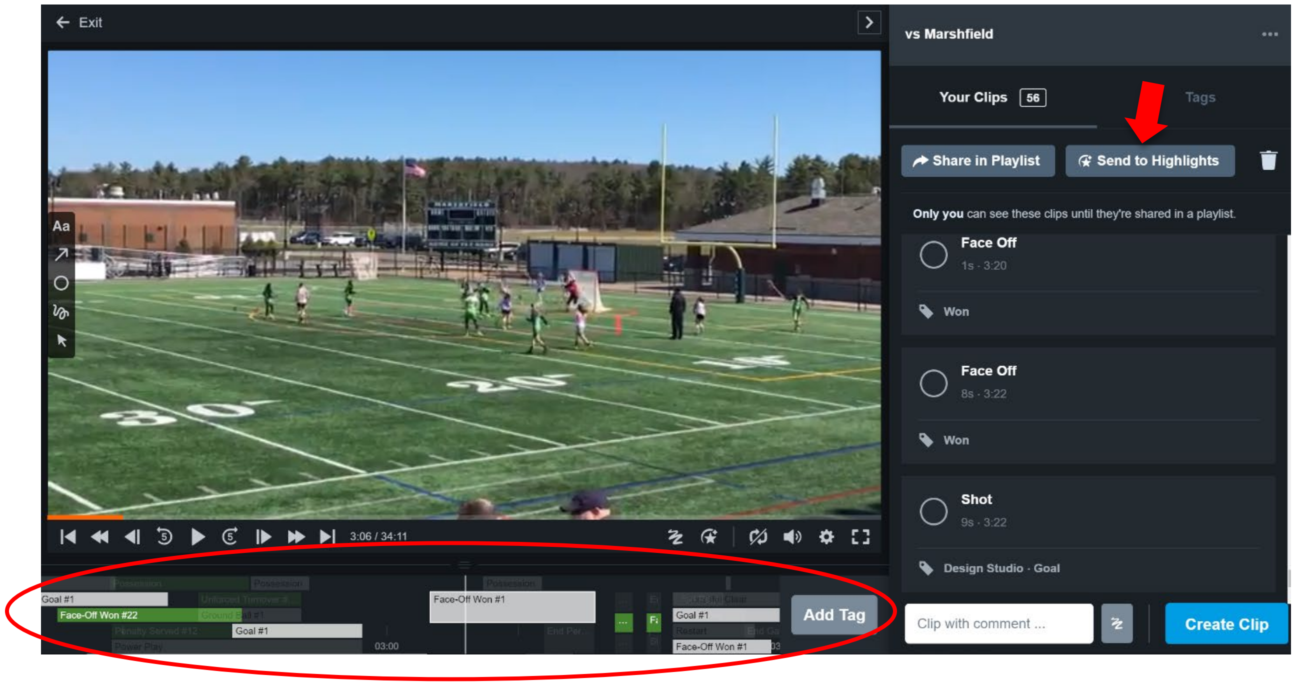


Figure 10. Hudl video player after positive tags are filtered. This is the view the email links to. The ability to “Send to Highlights” with one click of a button after these moments are selected makes creating highlights a simple process.

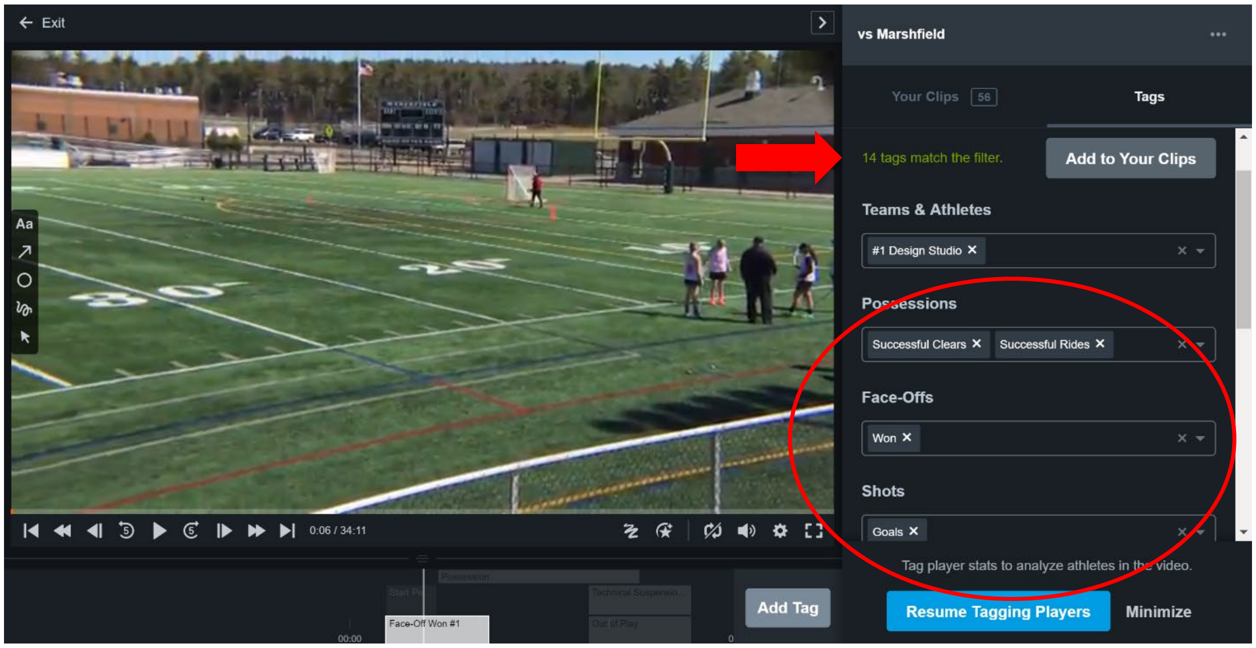


Figure 11. Hudl video player with pre-selected positive filters shown on the right. Normally an athlete would have to manually select these filters, but the email link takes them directly to a pre-selected version.

- After a video from a game is tagged by the coach, a notification appears which directs the athlete to click on a link. This link leads them to a pre-filtered video with all their best moments already selected, after which they can send it to the highlights editor with one click.
- Email notifications were implemented, and the possibility of other types of notifications was discussed.
- Testing this solution and gathering data on its usage was not complete at the time of writing this thesis.

The following are some possible solutions we identified, but chose not to pursue:

One-Click Auto-Gen

- Athletes would be able to automatically generate a highlight based on a recent video with one button.

- This solution did not make it far past the ideation stage because we realized it would be too complex and not feasible for the scope of our project.

Mobile Notify and Filter

- While this would have brought more value to our Notify and Filter solution for Hudl, the idea was presented with only two releases left in the project and only one of our developers had any iOS development experience, so it was decided not to pursue it.

Future Solutions

At the time of writing this thesis, there was one modification to a solution still in the early stages of development and planned as the next deliverable feature.

Notify and Filter with Statistics

- A minor modification to the Notify and Filter email message will be included which lists how many of each type of positive moment the athlete had in a game, so they can review their statistics.
- Hopefully this modification will get athletes excited about what they accomplished, even more than the current email's wording, and encourage them to make highlights of the game.

Appendix C

Suggestions A/B Test Results

The results of our A/B Test were evaluated using an online tool: https://neilpatel.com/ab-testing-calculator/. This tool provides results which include the percentage better or worse the experimental option is to the control, the certainty, and categorizes results on whether they are statistically significant or not, or "questionable." It seems that the tool uses a p-value threshold of < .05 for a "yes" answer, "questionable" indicates a p-value between .05 and .1, and "no" is anything greater than .1.

Table C1

Highlight Creation (Primary Metric)

	Suggestions ON	**Suggestions OFF**	**Better/ worse?**	**Certainty?**	**Statistically significant?**	**P-Value**
Week	**% Created**	**% Created**				
49	20.94%	21.09%	-1%	53% in favor of OFF	No	.47
50	28.74%	25.15%	15%	99% in favor of ON	Yes	.01
51	28.29%	26.05%	9%	90% in favor of ON	Questionable	.1
52	30.98%	31.22%	-1%	54% in favor of OFF	No	.46
1	29.37%	30.10%	-3%	65% in favor of OFF	No	.35
2*	25.12%	25.94%	-4%	60% in favor of OFF	No	.4
Total	27.93%	26.70%	5%	93% in favor of ON	Questionable	.07
*as of 1/8/19						

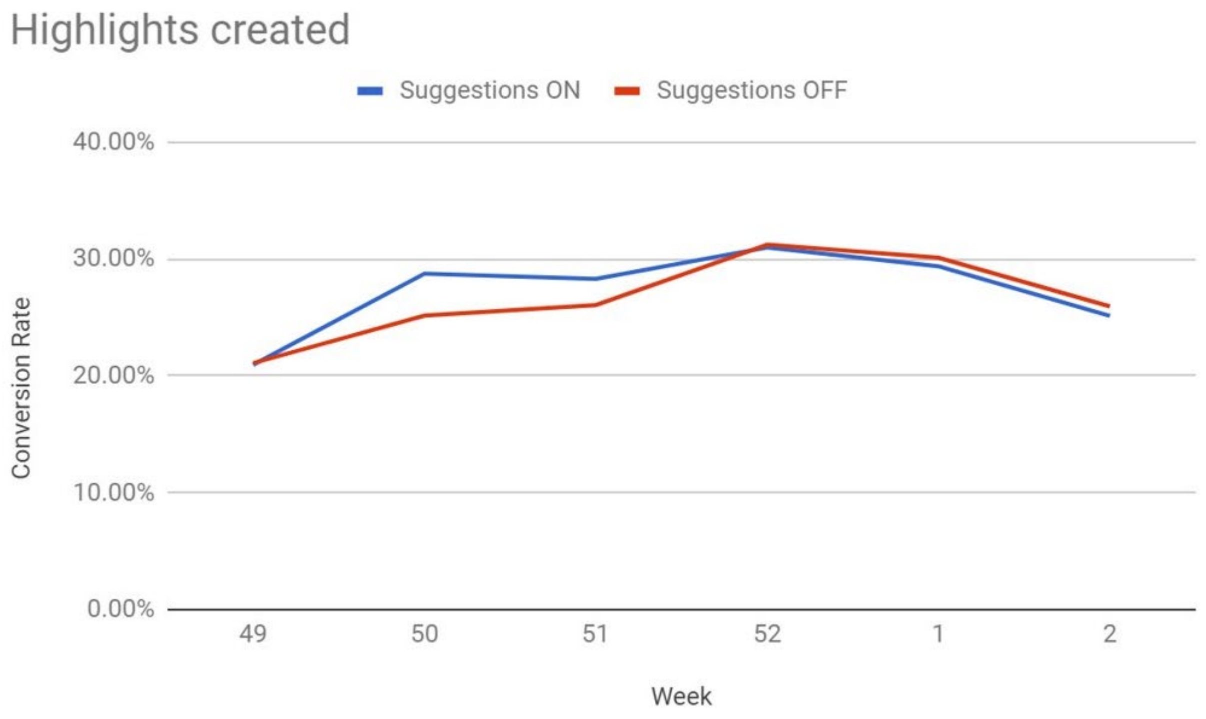


Figure 12. This graph depicts the percentage of users creating highlights after entering the highlights editor, over a test period of 6 weeks. Taken from: Wiles, B. (2019b). Suggestions AB test results. Retrieved from https://sync.hudlnet.com/display/BETS/Suggestions+AB+Test+Results

Table C2

Highlight Sharing

	Suggestions ON	**Suggestions OFF**	**Better/ worse?**	**Certainty?**	**Statistically significant?**	**P-Value**
Week	**% Shared**	**% Shared**				
49	60.00%	40.30%	49%	100% in favor of ON	Yes	< .01
50	26.94%	28.33%	-6%	68% in favor of OFF	No	.32
51	32.30%	29.40%	10%	80% in favor of ON	No	.2
52	25.20%	30.16%	-20%	88% in favor of OFF	No	.12
1	28.50%	23.81%	20%	91% in favor of ON	Questionable	.09
2*	20.75%	16.83%	24%	74% in favor of ON	No	.26
Total	30.72%	28.25%	9%	94% in favor of ON	Questionable	.06
*as of 1/8/19						

Conclusions

Based on these results, we had the following conclusions:

> Overall, the test appears to be inconclusive, with slight preference towards turning suggestions on. However, the statistical significance for both metrics overall is questionable. There are some weeks where suggestions on wins and is statistically significant, but that doesn't hold true at scale. It is possible that … users were more curious about it, but that interest declined as time went on. Many weeks saw suggestions lose or have no statistical significance. My suggestion is to no longer pursue or invest resources in this feature (Wiles, 2019b).

References

About us (n.d.). Retrieved from https://www.hudl.com/about

Antonson, M. (2018). Overview & Syllabus [PowerPoint slides]. Retrieved from https://canvas.unl.edu/courses/45000/files/folder/Presentations?preview=2558982

Beck, K. (2000). Extreme programming eXplained :Embrace change. Reading, MA: Addison-Wesley. Retrieved from https://www.safaribooksonline.com/library/view/-/0201616416/?ar

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (2001a). Principles behind the agile manifesto. Retrieved from http://agilemanifesto.org/principles.html

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (2001b). Manifesto for agile software development. Retrieved from http://agilemanifesto.org/

Bittner, K., & Spence, I. (2006). Managing iterative software development projects. Upper Saddle River, NJ: Addison-Wesley. Retrieved from https://www.safaribooksonline.com/library/view/-/ 032126889X/?ar

Davis, B., & Radford, D. (2014). Going beyond the waterfall :Managing scope effectively across the project life cycle. Plantation, FL; 4: J. Ross Publishing. Retrieved from https://www.safaribooksonline.com/library/view/-/

Freedman, R. (2017). What is agile? (First ed.). Sebastopol, CA; 4: O'Reilly Media. Retrieved from https://www.oreilly.com/library/view/-/9781492039860/?ar

Gray, A. (2015). A criticism of scrum. Retrieved from https://www.aaron-gray.com/a-criticism-of-scrum/

Hudl design studio 2018/19 | highlights are easy (n.d.). Retrieved from http://designstudio.unl.edu/Home/ProjectDescription?id=40

Moore, A. (2018). Hudl team evaluation. Unpublished manuscript. Retrieved from http://designstudio.unl.edu/Team/Evaluation?token=6e834f67-f411-4d31-bde6-f23197e5b95f

Poppendieck, M., & Poppendieck, T. (2003). Lean software development :An agile toolkit. Boston: Addison-Wesley. Retrieved from https://www.safaribooksonline.com/library/view/-/ 0321150783/?ar

Rubin, K. S. (2013). Essential scrum :A practical guide to the most popular agile process. Upper Saddle River, NJ: Addison-Wesley. Retrieved from https://www.safaribooksonline.com/library/view/-/

SaumyaSaxena1, & ritwikshanker (n.d). Software engineering | incremental process model. Retrieved from https://www.geeksforgeeks.org/software-engineering-incremental-process-model/

Schwaber, K. (2004). Agile project management with scrum. Redmond, Wash.: Microsoft Press. Retrieved from https://www.safaribooksonline.com/library/view/-/

Wiles, B. (2019a). Design studio behaviors. Unpublished manuscript. Retrieved from https://unl.box.com/s/5ns780min9v5oxihmc14uhzghdzrou13

Wiles, B. (2019b). Suggestions AB test results. Retrieved from https://sync.hudlnet.com/display/BETS/Suggestions+AB+Test+Results