Sprint 0: The Sprint That Isn't

When a team has been working together for a long time on a series of similar projects they have a lot of advantages that a newly formed team doesn't have:

- A well understood process for getting work done.
- Understanding of the types of resources needed for this type of project, and how to set them up. Often they are already set up from previous projects.
- Well established relationships and communication patterns.

Because of this, there is a significant gap in the activities of the first sprint for a "veteran team" versus one that has just been created that extends beyond just the difference in technical experience of the individual team members.

This leads to the question of whether it is fair/possible to expect a new team to handle all of the sprint tasks in the initial sprint after which they have been formed.

Some believe that there should be a special "Sprint 0" for some teams that allow them to get a grasp on the early fundamentals of a project before beginning the "real development".

Others think that having "special sprints" is an improper change to the Scrum process and that it delays teams from getting into the rhythm of regular sprints.

This an ongoing debate in the software engineering industry, and the correct answer likely depends on each particular development team.

However, at the point in the class, you are simply not ready to begin proper development on your projects. There are several parts of the process that you still need to learn about, as well as aspects of your project that you need to set up.

Goals for Sprint 0 in this class

- Identify and make progress towards eliminating the biggest technical risks of your project.
- Understand the process of User Interface Development expected by this class.
- Develop the interfaces expected to be used in Sprint 1.
- Understand the process of using sprints to incrementally deliver working software.
- Decide on the teams process for creating, testing, and delivering software.
- Set up git and development environment.

- Make sure everyone is familiar with the expected SCM process.

While we are covering these things, we will do some aspects of a typical sprint while leaving out others.

In Sprint 0:

- Modified sprint planning meeting
- “Daily” Sprint Meetings
- Modified Progress check ins

Not In Sprint 0:

- Expectation of story programming tasks completed
- Sprint Retrospective
- Demo of software

During this sprint we are going to be focused on learning the processes and tools we need, as well as preparing for the real sprints. While there is no demo for Sprint 0, this time is vitally important to your teams success, and your progress will be monitored to ensure that the new processes are being correctly applied. This will help you to form good habits during the actual development work.

Modified Sprint Planning Meeting

At the start of every sprint a planning meeting is held to discuss and define the user stories which the team plans to deliver at the end of the sprint.

For each of stories in the planned sprint backlog the following occurs:

1. The story is discussed
2. The story is broken down into tasks to be performed by the developers

During Sprint 0, we are breaking the stories down into tasks not to get those tasks done this Sprint, but to get a better understanding of where we have to beef up our skills in order to complete this project.

Essentially you are going to be planning Sprint 1 twice, once this week with very little information, and a second time in 2 weeks with the information and the skills that you have gathered over the course of Sprint 0.

Before your group begins the modified group planning meeting, make sure you have finished estimating and prioritizing your product backlog. This will play an important role in the meeting’s discussion.

Discussing The Stories

First off, the sprint planning meeting is a great time to ensure that the story has not changed in priority since it was last placed in the list. It might be that a User Story was thought to be important, and since then its priority has faded. It is best to know before detailed work begins that the story is no longer important or necessary.

If the story is still important enough to be worked on this sprint, the team discusses the story enough to be able to break it into its tasks. There does not need to be a ton of detail at this point, that can be done in subsequent discussions specific to that story. This might be the appropriate time however to do some of the following activities:

- draw a basic sketch of the interface
- discuss how it fits into the application flow
- identify any major technical challenges this story might contain

Stories at this point should be in the size of roughly 1 to 5 days work in order to be easily broken down into tasks that can be completed within the length of the sprint. If the stories are larger than that, they should be further broken down into smaller stories, not tasks.

User Stories Applied uses the following example for breaking down the user story:

“A user can search for a hotel on various fields.”

- code basic search screen
- code advanced search screen
- code results screen
- write and tune SQL to query the database for basic searches
- write and tune SQL to query the database for advanced searches
- document new functionality in help system and user’s guide

When establishing the tasks for a user story, you might want to follow the following guidelines:

- Tasks that are very difficult might need to be separated away from the rest of the story
- If a task has parts could be easily accomplished by two different people without a ton of communication necessary, then split those parts into two tasks.
- If the user story is small enough, there shouldn’t have to be a limit on the size of the task, although if a task in a user story seems to be way larger than expected, the points of the user story may need to be reevaluated.

These tasks should be added to the sprint backlog. Their title should indicate both the task and the user story it is a part of.

At this point in a normal sprint, the tasks would then be assigned to individual developers and another round of estimation would occur. However, this is not a normal sprint, and as a team you should to do the following instead:

- Identify the parts of the stories that will be the most difficult to do and identify the skills that you would need to develop in order to do them.
- Brainstorm how the functionality could be displayed on application screens.

Identifying High Risk Tasks

When you first getting into application development it might seem like everything is a high risk task. However, some tasks tend to be “riskiest” in this class:

- Stories involving complicated algorithms or AI (Almost Impossible)
- Stories involving networking (Very Hard)
- Stories involving Games (Hard)
- Stories integrating APIs (Depends on the API)
- Stories involving Databases (Relatively Easy)
- Stories involving Complicated User Interfaces (Not Appropriate At This Time)

Stories involving complicated algorithms or AI:

Over the semesters there have been a variety of apps that initially set out to be able to “figure out” how to accomplish some task automatically, such as recommending healthy meals, generate complicated music playlists, or create video transcripts.

These types of stories are not achievable in the scope of the semester.

Be aware of phrases like “The app is going to suggest...” or “The app will automatically…”, these are likely to be stories that contain complicated algorithms or AI.

If you have stories like these, think about how you could provide similar value to the user without the app magically doing it for you. Providing an easy to use interface that allows the user to make their own decisions might be the best strategy.

Networking Tasks:

The sign of having a story that involves Networking is if it includes “multiple users”.

Student’s have generally found networked applications to be a high hurdle in this class. These are possible to be successful as long as the networking is light in scope, such as sharing a database amongst multiple users.

Having updates happening on the screen in real time is much less achievable at your group's experience level.

These types of tasks are greatly helped by having group members with previous Networking experience. If you do not have any such members, you are going to have a very hard time implementing anything beyond shared database tables.

Stories involving Games:

Graphics can be very difficult to implement in mobile applications without a game engine or a graphics framework. At the moment, the course does not support Unity development, so if you want anything above the most simple shapes and interactions, you will need to find and learn a graphics framework. Students in the past have found success using LibGDX, although they came in with prior experience.

Stories Involving APIs:

If your app needs to play videos, music, display map information, get tweets, or any other kind of complicated media thing you have no idea how to do, then your app is going to need to use an API.

Many class applications in the past have successfully integrated APIs in order to offload some otherwise impossible development tasks. Some of these APIs include:

- Youtube
- Google Play
- Google Maps
- Spotify
- AWS
- Alexa

The more popular the API you are trying to use is, the more likely you will be able to successfully integrate it into your app. Popular APIs tend to have a lot better documentation, better features, and many examples online.

Even when using a popular API, Sprint 0 is the ideal time to figure out how difficult it will be to use in your app. If the API is central to your app, you should be spending time now figuring out how to use it and trying to do basic stuff in a test application over this time period. If you don't know what API you are going to use, now is the time to try and find one.

Stories involving Databases:

Whenever your app has data that shouldn't be deleted whenever the application closes, some sort of Database should probably be on your mind. Stories that involve databases might include the phrase "persist" or "The user

is able to store…”

Some applications with very simple data needs might be able to get away with storing data in XML or JSON files, which should be investigated if that’s what you decide to do.

For single user applications with more robust needs SQLite is probably your best option.

For applications that need storage independent of one device, Firebase is a good bet.

By the end of Sprint 0, it would be ideal for you to be able to do basic storage and retrieval with a storage tool that you think is appropriate for your app. The data you store for now doesn’t need to be specific to your app, it can be “Hello World!”. But this can make a huge difference in knowing what it will take to make your app work.

Stories involving complicated interfaces:

For now, stay away from developing complicated user interfaces, because it is likely that there exist simpler options that can be built up over time.

Instead focus on providing as much value to your users with as simple an interface as possible.

Now What?

You’ve identified some of the biggest sources of technical risk in your project, and possibly reduced some of its intensity.

At this point your team should be mostly aware of the amount of technical risk in the project, and everyone should be in agreement that this is a level of risk that everyone feels they can overcome. If several team members are not comfortable, this may be a sign that the scope of some of the more technically involved features may need to be reduced or some of those features need to be eliminated.

This does not mean that your project should have no areas of technical risk. Accomplishing an appropriate level of risk is one of the best ways to gain experience as a Software Engineer. There should still be some areas with technical risk remaining in the project. So how do we go about approaching and resolving these risks?

It might seem simple, go out and learn all the information about the thing you are trying to do. However, game development, networking, and databases are all their own classes, how are you going to learn everything about that subject in just a few weeks?

Start by identifying what aspects of these technologies you need, you certainly don’t need to know everything. If you don’t know enough in order to do that, research a general overview of that technology.

Once you know what you need to know, start by only learning enough to understand the basics of how it is going to work and to be able to provide an estimate on how long it will take to implement.

When you get here, you and your team are probably ready to really start learning the technology. But you still want to break this up into as small of chunks as possible.

Create what Patton in *User Story Mapping* calls a “minimum viable product experiments or MVPe for short. It’s the smallest thing I could build to learn something”

If there’s a particular thing your team has to do with a technology, try to think of the smallest thing you could build that does that task at a basic level. At this point, what you built does not need to be a part of your real app, it can just be a really small test application.

In this reading, we covered the purpose of a Sprint 0 and exected outcomes.

These are the primary goals:

- Identify and make progress towards eliminating the biggest technical risks of your project
- Understand the process of User Interface Development expected by this class
- Develop the interfaces expected to be used in Sprint 1
- Understand the process of using sprints to incrementally deliver working software
- Decide on the teams process for creating, testing, and delivering software
- Set up git and development environment
- Make sure everyone is familiar with the expected SCM process.

We have now covered the first goal, "Identify and make progress towards eliminating the biggest technical risks of your project.”

Over the course of the next readings we will also establish the process of UI development and SCM you will be expected to use throughout the sprints.