Throughout this reading we will attempt to describe the remaining processes that will come up during your actual Sprints. These include:

- Planning an actual Sprint
- Coming up with a Sprint Goal
- Designing and Implementing the User Stories
- Having Productive Sprint Meetings
- Wrapping up the Sprint Work

We have now spent the last couple of weeks experimenting in order to prepare for our actual sprints, in which we will build our intended application.

As mentioned in the Sprint 0 reading, at the start of every sprint a planning meeting is held to discuss and define the user stories being worked on over the sprint.

Sprint Planning

Note: While some of this information is the same as the Sprint 0 reading, there is important new information using what we have learned since the start of Sprint 0. Do not skip this section.

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.
3. Developers claim responsibility for at least some of these tasks.

Discussing The Stories

During the Sprint 0, we should have discovered a lot about the nature of our user stories' technical risks. It is possible that we have found that some are much easier to do than anticipated, or that others are completely impossible within the scope of this project.

At this point, each team member should be much more informed on what the main tasks of development will be, how long they will take, and how to achieve them. This will greatly improve the accuracy of our sprint planning and execution.

The Sprint Planning meeting is a great time to update these estimations and priorities with your team.

We recommend going through the user story estimates again, and making sure that these still make sense given what you have learned about your app's implementation during the Sprint 0 phase. It is likely that you have greatly over or underestimated various stories in your backlog.

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.

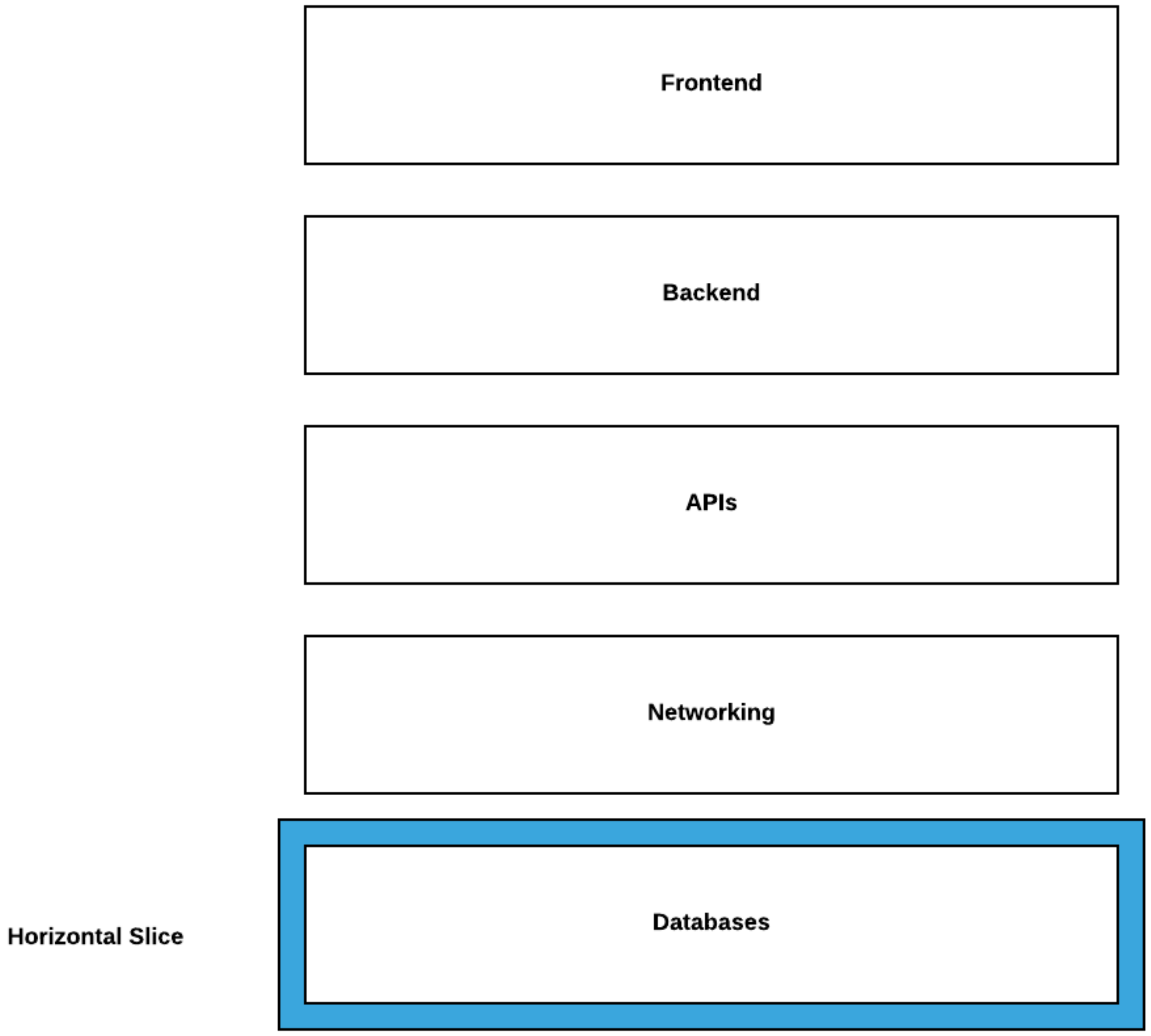
The Sprint Goal:

While ideally we are always working on the most important user stories for our apps, we also want to create an iteration that resembles a valid, releasable product.

We therefore want to define a sprint goal as a collection of related user stories that would represent some releasable version of the product.

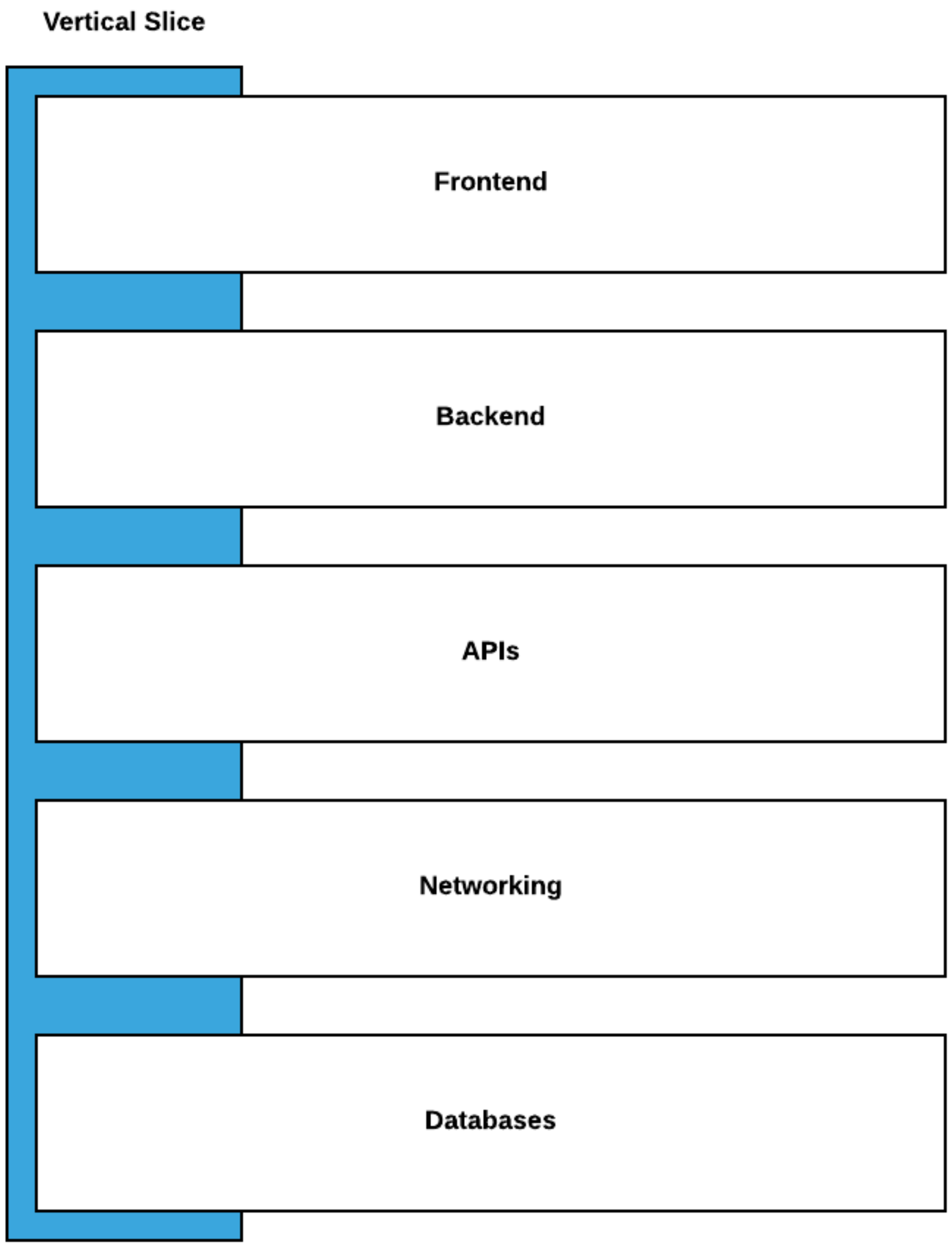
Ideally, our first sprint would consist of only several must haves, but sometimes our most important stories are unrelated to each other. When this is the case we should choose as many related must haves as possible, followed by accentuating should haves or could haves that complement the primary stories and fill in the rest of the sprint.

The biggest mistake that students make when creating a sprint goal is to aim for a horizontal rather than a vertical slice of functionality. What we mean by this is best encapsulated in two pictures



A horizontal slice would be taking one part of the software stack, such as the Networking or Database layer, and trying to implement all of this over the course of the sprint. Students will often set out to "complete all of the database" throughout Sprint 1. While all the database functionality sounds nice to have, there isn't much feedback that can be said about it, and we are only guessing about all the needs of our database since it is not connected to real backends meeting real user needs. We instead want to aim for accomplishing vertical slices of our system.

A vertical slice would be taking enough of each layer to provide some user functionality. Just enough databasing, enough networking, backend, and front end to achieve a user outcome. By accomplishing vertical slices, we are able to get user feedback, and we are making sure we are only building parts of systems that our users actually need.


Vertical Slice
Frontend
Backend
APIs
Networking
Databases

During the sprint we should aim to create as many fully functioning vertical slices as possible.

Breaking the Stories into Tasks

If a story is within the sprint goal and small enough to be broken down, the team discusses the story enough to be able to break it into its tasks.

These tasks should be more clear than those defined in the Sprint 0 planning meeting. In particular they should focus on creating:

- The View components necessary to have the user achieve that story.
- The Model components that can represent the data used throughout the Story.
- The Controller components that connect the Model and View components as well as implement the necessary backend processes that make the user story work.
- Properly integrating and testing these components.

This is an appropriate time to be drawing high to medium level Code Component views, sketching wireframes, and possibly some UI maps.

While it can be exciting to add a bunch of new screens, it is important to limit the amount of screens as much as possible, especially in the beginning.

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 that could be easily accomplished by multiple people without a ton of communication necessary, then split those parts into two or more tasks.

These tasks should be added to the sprint backlog, in a column separate from the product backlog. There should be a clear link between a task and the User Story it is a part of.

Once all the tasks for the story have been written down, it is time for the tasks to be assigned to a team member.

Assign Developers To The Tasks?

As with so many things, there is a debate in the Agile community over whether the team should leave the Sprint Planning meeting with all of the tasks already assigned to the developers or not.

The debate mostly revolves around individual versus team accountability.

It is the team's responsibility to try and complete a usable version of the stories they've decided to take on for the sprint. Regardless of an individual's contributions the team is either successful or not.

In a professional team, it doesn't particularly matter if all the tasks are assigned in the beginning or not. As team members complete their tasks, they will work on new ones.

With new teams, it can be risky to leave so much of the task allocation undefined. Members unaccustomed to the Sprint process will often only attempt to do their initial task throughout the Sprint, procrastinating until the last few days to get it done. Not only will this result in the team member losing points (In this class) throughout the Sprint, but will also damage the team's ability to accomplish their goals.

Having all of the tasks assigned before the Sprint begins can give new team members a clearer view of the full amount of work they will have to do throughout the Sprint in order for the team to accomplish its goals.

Some experts recommend that at least for the first few sprints, all of the tasks should be assigned to team members during the sprint planning meeting.

For more information regarding this debate:

https://www.planningpoker.com/blog/should-your-team-claim-tasks-in-sprint-planning/

How are Tasks Assigned?

In Agile, tasks are volunteered for by members of the development team.

There is no manager that assigns who works on what task. In order to provide developers control over their work load, all tasks are assigned by a developer volunteering to do that task.

This can be problematic in the classroom if a person refuses to accept beyond a bare minimum amount of work, so it is important for a team to set the expectations of work necessary for the whole team to achieve their goal.

In the Waterfall process, there is likely to be designers that work on UI tasks, programmers that work on backend tasks, Database admins that work on the persistent data, and so on. In an Agile project it may be tempting to fall into this pattern. However we should actively try to resist it.

Having specialists on a team who are the only person who can accomplish a certain type of task tends to just create bottlenecks on the work being done.

Instead effort should be spent by those who know more about a subject on getting everyone else up to speed.

This allows any piece of work in the Sprint backlog to be picked up by anyone.

In order to achieve this goal, individuals on a team should actively try to take on a wide variety of tasks that touch on many different aspects of development work, especially near the beginning of the project. As the sprints progress, each team member will become more and more cross functional.

After all of the stories have been discussed, the developers take a look at the tasks they have been assigned and individually estimate the amount of work that their tasks will take. It is not important for these estimates to line up with the user story estimate, although if one task is longer than the whole story estimate, the story estimate should be re-evaluated.

Once all of these have been added up, the team member needs to make an assessment of whether they will be able to complete all of these tasks throughout the course of the sprint or not. In the case that all of this cannot be completed:

- The tasks may need to be spread to other developers.
- Some stories might need to be split into more stories, with some of the new stories not in the scope of the current sprint.
- Stories might have to be dropped from the sprint altogether.

Tasks that no one will be able to work on should not stay in the sprint backlog.

Using the Stories During The Sprint

At this point your team should now have a general understanding of the big picture things they have to do during the sprint. There should be interface pieces to make, systems to create and connect, as well as new technologies to understand.

The team should now choose a user story to begin working on, preferably the most critical one to the application. The team should then discuss the high level overview of how the code components will interact in order to achieve the functionality. This overview should be added to the user story Trello card.

The team should use a branch for the specific user story that functions similarly to the development branch, and gets merged into the development branch when complete. Tasks would then be merged into the story branch upon completion.

At this point the developers are ready to begin working on their tasks. One task for each developer should be chosen, preferably a task that is the most important, and moved to the in progress column of the Trello board.

Using the high level overview as a guide, developers need to identify who is writing the components that their current component interfaces with. At this point, these involved developers should meet to discuss the medium level interface for this function and agree to how these components will interface with each other. These medium level designs should be stored on any task that is or interacts with this component.

If we are using the DIP, these interfaces should be literal interfaces that the code interacts with, while the other developers code concrete classes that implement those interfaces.

If your components are properly broken down, it should not be necessary for two people to work simultaneously on the same components. This is not only a sign your component violates the SRP, but will also result in merge conflicts.

It is the individual developers job to then create a low level design that implements that interface. This should also be stored on the task Trello card. The developer may then begin implementing the task on a branch specific to the task.

Throughout this process of design, the user stories should be further fleshed out with more acceptance tests, or can still be restructured at this point with team approval.

Reliance on Other Components

Often times students will say they cannot complete a task until another student finishes their task, derailing progress.

Using the SOLID principles, most of this is completely avoidable.

Remember how we have just defined a minimal interface for how our component interacts with others?

We can use this to create a simple dummy concrete implementation that pretends to work just enough to properly test our application. All it needs to do is return hard coded values that fulfill the return types promised by the interface,

Once the real implementation is completed, a real instance is passed to the function instead of the dummy one, and if properly coded, both classes work without needing to modify any code, because references to the dummy implementation exist only in tests, and the real classes only reference interfaces.

Task Completion

Over time, developers will begin “finishing” a task. At this point, it is important that the developers test their own code, as well as submit their code to their teammates for review.

We will provide a more detailed breakdown of code review in the future, but at the very least the code should be tested to ensure that it works, it meets the intended task scope, and does not break anything else in the system.

When code does not meet these requirements, the code should not just be fixed for the teammate. There should be clear feedback about what ways the code has not met the team’s requirements, and the developer should have the opportunity to fix their mistakes.

At some point, a developer might complete all of their tasks for a particular story, but still have tasks on other stories for the sprint. At this point they may wish to focus on testing and accepting other people's work until the group can discuss and plan an additional story.

If a developer has completed all of their assigned tasks, they should discuss with their team the way to move forward that best helps their team. They should not stop working or just start working on other people's tasks.

Scrum Daily Meetings

Throughout the sprints, the daily meetings are the main method by which the team gains a shared understanding of the project status.

In each meeting, each team member provides a brief answer to the following questions:

- What have you done since the last meeting?
- What are you going to do before the next meeting?
- What is in your way?

These will help keep your team aware of what is happening with each member, increasing understanding, cutting down duplicated work, and helping the team become aware of problems.

One thing that early Scrum users often do incorrectly is to go into way too much detail regarding someone's answer to these questions. The questions are intended to be answered with a brief overview, and with little discussion about the team responses.

If there are areas that need to be discussed in depth, this should be done after the meeting with only the members who need to know this much detail about the matter being discussed. This allows uninvolved individuals to get back to work.

What is in your way?

This question tends to be the area in which students waste the most time.

The purpose of this question is not to allow you to vent.

Instead, it is intended to bring to light solvable problems that your team (or, in industry, your ScrumMaster) can assist you with to help you complete your task.

Common things that your team can help you with include:

- Getting your development environment set up.
- Not understanding how to implement your task.
- Not understanding what your task is about.
- Not understanding how your task fits into the system.

During the meeting, one or more members should volunteer to help you resolve these problems after the meeting is over. The Scrum meeting is not the time to actually resolve the problems.

Sprint Late Game

As the sprint goes on, you will be hopefully finishing task branches that get merged into user story branches, and finishing user story branches that get merged into develop.

We want our develop branch to only contain fully implemented stories, and not any stories that are incomplete.

We want master branch to end up containing a collection of finished user stories that represents an intended iteration of our application.

It is extremely unlikely that in the first few sprints we will accomplish everything we wanted to do within the sprint.

As the sprint comes to a close, we will find that there are probably some stories still in progress, and even some stories that haven't been started at all.

The stories that haven't been started at all should be put back into the product backlog, at a place that represents their priority.

Stories that have been partially implemented should first be re-evaluated to consider what work needs to be done to finish the story. This is essential for picking back up the work in a future sprint. The task should be estimated again and placed back in the product backlog at an appropriate priority level.

In Summary:

Over the course of this reading we have discussed the following topics:

- Planning an actual Sprint
- Coming up with a Sprint Goal
- Designing and Implementing the User Stories
- Having Productive Sprint Meetings
- Wrapping up the Sprint Work

In the next reading we will watch a video that introduces some important concepts for testing our software as we create it, rather than at a dedicated step at the end. This content has too many interactive pieces to be written as a reading.