<u>Notes from someone who took this class</u>

Introduction

This is the intro.

This document is based off a compilation of notes taken during Software Studios. This is meant to give ideas that a team needs to value when starting the new semester. Keep in mind, this is made from the perspective of one student and not a professor, so take it with a grain of salt. These tips take from experiences from the team's activities and topics that professors have said in class meeting. Without further ado…

Work Delegation: Team Leads and New Members

"It all goes back to Taylor."

Make sure that the team lead is not the person doing most of the coding. This can lead to everyone else having no idea how the code works. Ideally, the team lead is supposed to focus on managing and guiding the team throughout the project, instead of being the sole pillar the project is built upon. This forces senior programmers to explain their code to less experienced teammates they are working with, resulting in more people understanding the project's code. The goal should be that if the team lead is evaporated from existence, **knocks on wood** team members can still work on the project without interruption.

If you're new to the team or new to the code you have been assigned to, don't be apprehensive to asking questions. The start of the new semester will most likely be dedicated to learning and less actual coding, so take this time to study the existing code, each component's purpose within the whole project, and any concepts that would be deemed useful to the project's construction.

Semester Goals

Shoot for the moon, but one you can hit.

There might be a lot of features for your project that the team wants to implement. For example, you might want to have a fully functioning data warehouse with a brand-new input format, functioning front-end, and full communication with the front and back end all on a working AWS server. While it is a good idea brainstorm ideas for your project, you need to keep in mind that you have only one semester to work to implement your ideas.

There may be unforeseen incidents that can obstruct the progress of a feature. For example, your AWS server may be crashing every time someone works on it, or there may be a learning curve on a concept that you thought would be easy to implement. Lack of progress on a project's feature may start to make the list of other stuff you wanted to do look daunting and may lower team morale. So, try not to bite off more than you can chew.

Get together with your team to discuss what one or two features should be the focus of this semester and determine what components need to be made make them reality. This way you team can focus all their efforts into creating excellent products. Of course, there needs to be accountability among each other to make sure everyone is doing what they can to help.

Communication is key.

We need to know what the heck is happening.

Slack messages should not only consist of the team lead's and the "mega-skilled" programmers' messages or worse yet just the team lead and "yessir" replies. Let everyone know what you are doing regarding the project. This may be in the form of stand-ups, which are daily updates each team member sends in Slack. This should not be a shaming sort of thing. You can let the team know that you weren't able to work on the project that day for reasons you provide. Stand-ups are meant to let the team know where everyone is at.

Not everything has to do with coding.

You don't need to be Bill Gates

This may be stated in class meetings, but Software Engineering is more than just coding. For Studio at least, there are presentations, documentation, GitHub branch management, and burndown chart calculations just to name a few. If you think that you don't have "adequate" coding skills to work on your team's code directly, these are a few things you can do to not only take a load off your team lead's back but also help the team avoid any hiccups in the long run.

An example of an alternative role is the designated scribe, keeping notes on what is said in class, team meetings, or code reviews. Keep in mind this does not necessarily mean that you do all the documentation for the project. Team members who write code need to be able to explain their code to the scribe for better documentation.

Code Reviews

Toughen up, buttercup.

Most code reviews that yield the best results will be ones where someone other than the team lead is being subject to it, and the team lead, professor, or "more skilled" programmer is judging the code. Often, the "less experienced" programmer will have formatting mistakes that the senior programmers or the professor observing will notice. This results in there being more to correct and note. The point of a code review is not only to make sure the code is good enough to be pushed to main, but to educate teammates on what standard their code is to be upheld to. If your code is the one being reviewed, you need to be prepared to explain/defend any choices you made in writing your code. Some examples can range from using single quotes on your string variables to the workflow of a function.

Lunatic-Labs' coding standards for different languages can be found in the "language-references" repository in Lunatic Labs.

User Stories

How they are to be written, how they are to be point attributed.

User stories are going to be the goals you are to complete in a sprint. The formula for creating a user story is as follows:

As a **person of interest**, in order to **complete said person's goal**, I need to **have a feature that the project provides**.

User Stories need to be in simple language, meaning there needs to be little to no technical jargon. Someone that has no idea about your project needs to be able to understand them.

Another thing to note for user stories are their designated points. Points are assigned to each story based on their difficulty. An example could be a documentation story being given 2 points, or an endpoint creation story being given 5 or 6 points. Be sure to touch base with your team lead before creating a story.