Describe in detail your capstone experience (at least one page):

My capstone experience was a software development internship at Asynchrony. Asynchrony is located in St. Louis and it's one of the best Agile shops in the Midwest. Besides application and mobile development, they also do Agile consulting for companies such as Monsanto, Departments of Defense, Deloitte, Mastercard and others.

The internship project consisted of converting an Agile board game into a mobile application. There were thirteen interns, six working on an Android version, six on iOS and a designer. Each team also had one tech lead whose role consisted of creating and refining user stories as well as providing general mentorship and good advice throughout the internship. We also had an internship coordinator whose role was the equivalent of a Business Analyst, and finally a customer, which in our case was the CTO of the company. The goal of the internship was to not just learn how to develop an Android application, but to learn very good software development practices that will make us better developers. The goal was not to become expert Android or iOS developers, but to learn good habits so that no matter what kind of code we might be working on, we would follow good practices and make our code extensive, reusable, maintainable and testable.

I was assigned on the Android team and during my first week one person on our team left, so until the end of the summer our tech lead paired up with one of us and was more involved in the implementation process than is usually advised to.

Typical Day

This is what a typical day looked like. We would come in at around 9:00 pm and have a stand-up at 9:30 that would last no more than three minutes. After that, we would pair with another person and pull the right-most story from the Kanban board. Some of the time this would be a user story not yet started, but other times this could also be a bug found by QA or a story that was started the day before but was not yet finished. We work with the first pair until 11:00 pm after which we had a pair switch. The second pairing of the day usually lasted until 2:30pm with an hour off during lunch (I played ping pong at lunch, but there were also people that played video and board games). At 2:30 we would have another stand-up meeting that would also last no more than 3 minutes, after which we would have our third and final pair switch for the day. Finally the day would end around 5-5:30.

Agile Practices

Asynchrony prides itself on being an Agile company, so they tried to incorporate as much as possible of good Agile practices into the internship experience. Here are few things that we did, which shows that they are really serious about Agile, and it's not just a buzzword used to sell the company to developers.

First there was a huge focus on pair-programming. You couldn't write any production code unless there was another guy paired with you. Developers did not have their own workstations, but everyone shared the same computers and could log into any development station in the company. Everywhere you went you could hear chatter and collaboration from people working together to hash out a design or implement a story.

Another core value that we practices was Test Driven Development. This was by far one of the hardest things to learn and it took us half of the internship to get to a place where we felt comfortable with TDD (not be good at it, but just feel comfortable with it). The urge to start coding is very strong (because it's so much fun) but it's only after the fact, do you see the benefits of having tests around your code and not having to worry about introducing bugs when refactoring. Pair programming helps because people hold each other accountable and it prevents against spikes gone awry.

We also kept a Continuous Integration build at all times, which was essentially a server that had the latest APK passed by QA. This was so that at any point the customer could walk in and see the progress on the application. At any point in time we could show them the latest features implemented and the software would be usable.

We also had weekly presentation on Friday to our customer, which for us was the CTO of the company. Each team presented the progress they've made during the week and would show off the new features. We all had a chance to present, which was really cool. We also had weekly "retrospectives" where we discussed issues and came up with solutions on how to fix them. The retrospectives must produce "actionable" items so that next week we could say "we did this" or "we didn't do this".

Finally, we used user stories to implement the features of the application. We created these during a bi-weekly meeting in which we listed the acceptance criteria (used by QA to test the new functionality) and voted on story size. The stories were put on a Kanban board with several columns: Backlog ( where all new user stories were), WiP, Code Review, QA Review, Bugs and QA Pass. The stories in QA pass were merged into the main apk and this is what we presented in the weekly Friday customer meetings. One process that we introduced, which I really liked, was "new story turn around". This consisted of the pair that starts on a new story, do a quick turnaround to let everyone know that they are pulling a new story and to describe some of the acceptance criteria of the story. This was really beneficial because as the code base started growing bigger, we could identify the necessary classes and methods to do the work. This lead to greater knowledge sharing and implementing stories faster because we didn't have to re-invent the wheel every time.

Tools

We used plenty of tools throughout the internship. In fact the steepest learning curve was learning how these tools worked and then using them. In beginning we had to strip down some frameworks because it was too overwhelming, but later we re-introduced them back once we felt more comfortable.

First we used Git for version control. Our branch management was as follows: We used a master branch that had code that was working and has passed code review and QA testing. We also had a dev branch that we developed on. Each pair would start their story on a branch called "bug #" and branch off dev. When done with the story they would submit a merge request, after which a team will do a code review and either will have a code review fail or allow the request and merge the branch into dev. QA would then use the dev branch to test the new functionalities.

To help with TDD we had to rely on JUnit and bunch of other frameworks such as Mockito for mocking out objects, RoboGuice for injection and dependency inversion, Robotium and MonkeyRunner for UI testing and Robolectric for testing the Android parts of our code (Views, Lifecycle events etc.).

We also used several other libraries that helped with abstracting out details of Android that we didn't care about and did some of the boilerplate code for us, so that we could only focus on the logic part. Some libraries included Otto, AndroidAnnotations, Jackson, StaggeredGridView, and Gradle.

To develop we used mini Mac to begin with, but because Android is really resource heavy we had to switch to using Ubuntu mid-way through the internship. Because we used Git, this was super easy and took us maybe 30 minutes to switch machines and go back to our regular workflow.

Learning Challenges

I think by far the hardest challenge I faced was learning how to do proper TDD. TDD is unlike anything I've encountered before and it takes a while to learn how to think in TDD terms. The urge to just write code is too strong and you have to actively go against your natural instincts, but I think it's worth it. We realized the advantages of TDD whenever we had to refactor or make design changes. I felt so much more comfortable changing our code because I knew if any of my changes introduced a new bug or not. Also the code coverage when you do TDD is much higher and tests are higher quality because testing becomes the focus and not just an afterthought; it's harder to half-ass the tests when you do TDD.

On top of learning to think in TDD terms, you also have to learn the tools that help you with TDD such as mocks and injections, which I've never even heard of before this internship. Mocking out is essential for unit testing and it takes a while to learn the APIs available to you as well as how to use them properly.

Another challenge I encountered was learning Git for the first time, though I think it's very intuitive way to think about version control and it didn't take very long to get comfortable with it. After you learn Git though, you realize that the benefits of using it are enormous, and it's hard to understand why wouldn't you use it for everything. I've started using it for anything, from programs assignments to writing papers and my resume. It's really an awesome tool that helps you so much, and it's pretty easy to learn it too.

Problem I encountered and it's solution

One particular problem that I faced was changing the code of an existing library to work within our application code base.

The story was to implement a Grid Layout from a bunch of blocks available on the screen, but according to the Design specifications it had to staggered grid, because some blocks where taller than others, so they all had to stack on top of another. The library that we used to help us with this, supported staggered grid but the blocks were meant to be static, they couldn't change. The problem was that in our game, whenever the player would finish a card, the card would have to collapse to a much smaller rectangle, and all other cards on top and bottom were supposed to collapse around it. To incorporate this piece into our application required that we really looked deeper into the library we were using and really understand the API that was public for us to use. Eventually, I fixed this bug by calling several methods consecutively, essentially emulating a destruction of the old and re-creation of a new staggered grid view with the new data. I was really proud because it was essential to our app's design to have staggered grid view, but also because the Tech Lead tried fixing this issue before and couldn't get it to work, whereas it took me and another intern about 3 hours in the afternoon. This was one of the moments where I realized that it's impossible to know everything about software development, and there will be times where I will know more or get a better solution to a problem than some of the more experienced developers out there. It was realization that to be a good software engineer you need experience, but there is also that "aha" factor that comes from getting an insight into a problem. After this I realized how much I enjoy programming and solving real world issues, but it also made me a lot more confident in my skills as a developer. It's not necessary to know every API method available to you, but as long as you can think logically about an issue and try to really understand it, you can solve anything thrown at you.

CS Courses that were useful

Courses that were really useful were CS 181 and CS260 because Android is in Java and is extremely Object Oriented. You really need to be strong in this area to fully take advantage of the power and flexibility of OOP in general and Android in particular. Topics on Polymorphism, Abstraction, Inheritance, Encapsulation, Anonymous classes, Event Listeners are all over Android and you really need master these topics in order to write good maintainable and extensive code. For the first time I saw that I had to use interfaces because I saw the benefit of it and not because it was specified in the assignment.

Two other useful classes were Internet Programming and Software Engineering. In Internet Programming we get exposed to some of the basic design terminology that every developer should know how to do i.e. margin, padding, font size etc. It was essential to understand these so that we could code up the design specified by the designers and get it as close as possible.

Finally, Software Engineering exposed me to Git and some of the ideas behind Agile. Everything we learned in class was spot on, and it was really interesting to see how we were applying some of the things we learned about in class, on a day-to-day basis in the workplace. Also in Software Engineering I first learned how to use JUnit and how to write tests, which was very important because doing TDD, you're going to write hundreds of tests no matter what, and becoming comfortable with JUnit is essential.

Non-CS courses that were useful

I don't know if there are any specific classes that were useful, but I do think that liberal arts education exposes us to more diverse topics and make us aware that there is an interesting world outside that doesn't really care about computers as much as we do. Taking those classes instilled in me a curiosity to learn more about the world in general and to appreciate the experiences that individual people bring to the team. I feel like I can always learn something new no matter whom I talk to. I think liberal arts classes have made me a more rounded individual, and I can interact with people from many diverse backgrounds because of it. I can always find something in common with another person and form a friendship around it.

Positives

I think this is one of the best internship that CS students at Truman can do. Because it was a development internship, I wrote tons of code and I learned a lot as a result. Writing code daily really helped me develop my Java skills and learn programming on a deeper level. Whereas in college we get exposed to many topics and know a little bit about many things, here we were exposed to one topic but had a chance to really understand it and get good at it.

Also I was really lucky because our Tech Lead really cared about good code, held us to a very high standards and was not afraid to code review fail us for bad code. He was very pro-active in working with us, showing us good patterns and helping us overcome the bad ones. He was also really patient and I think we on the Android team learned so much more than the iOS team because we had such a pro-active Tech Lead.

More practically, I also learned some cutting edge technologies that helped me become a better developer and write better code. Learning TDD, using Git, writing OO code, working in an Agile team doing pair programming, these are all cutting edge practices that the earlier I learn them the more value I get out of them, because that's where in industry is moving.

Negatives

Not everything was rosy though, and I also found few places for improvement or where I didn't necessarily agree with how Asynchrony did things. I don't like to follow TDD religiously, but I want it in a more flexible flavor. There were few times where we had to change our design and expose more data than we should've just so that we could test the classes. Our final design ended up being more rigid which kind of defeats the purpose of TDD and there were cases where I would've been fine without white-box testing a class if it gave us a better design and cleaner code.

Another practice where I'm skeptical is pair programming. I see it as a double edged sword. If two people are paired and they have roughly the same skill level, then pair programming is awesome and it does marvels. But if one person is much weaker than the other, than there really is no pair programming and it becomes one person teaching the other how to do stuff. This slows down development and it's wasting resources. So I'd be very careful if I had to choose pair programming for a team because finding good pairs is as much as art as is science and could backfire.

Indicate if there is anything you now realize that you should have known or studied before starting your capstone experience

One nice thing is that when you go into an internship, the company assumes that you don't know much, which in most cases is true. The company assumes you don't know much, but are able to learn new things quickly and pick up new technologies.

But to be specific, I would've loved to be exposed to JUnit much earlier than Software Engineering. Writing test is inevitable no matter what kind of development you do, so ideally students would've been exposed to JUnit in CS181 and CS260 so that they at least know how to setup the test suite and write basic tests.

Also Git should be introduced in school, because if anything, it'll make life much easier and I'll never have to email code to myself ever again. Also I'll never again have to worry about accidently deleting my code and having to start from scratch again.

Explain whether the experience turned out the way that you anticipated that it would

Overall, this internship was a life changing experience, because if anything, it made me realize how much I enjoyed writing code. It was the first job where I was actually looking forward to going to work every day and was excited to start the day. I learned a lot, I made few good friends and I had tons of fun starting from scratch and building this application that would've been used by other people in the company.

But it's also thanks to this internship that I have a job going into my final semester. Because we did pair-programming every day, I have never been worried about a technical interview ever since. I just see it as another opportunity to solve an interesting puzzle with another person and I feel completely comfortable with writing code while someone's looking over my shoulder.