**John Madison's 6 month review summary:**

|  |  |
| --- | --- |
| **1. OpenShift:** |  |
| Before I came here, I knew nothing about network/database programming. Since then I have deployed 3 servers on OpenShift. 2 of them use MySQL, Hibernate, and Jersey.  **Learning OpenShift was the most infuriating thing I have ever done.** Probably because it involved usage of many technologies that were all new to me. |

|  |  |
| --- | --- |
| 2. My Gauntlet : UI & Backend: | |
| Before I was on the official guantlet project. I did my own version of it. I made a lot of mistakes. A did a lot of stupid things. But I got some great experience.  I was able to: 1. Create working Admin Login page. 2. Create working Candidate Login page. 3. Create working Add candidate page. 4. Create working Exam Dispatcher.  ***5. All backgrounds are proceedurally generated and animate into place when page loads.*** | |
|  |  |
|  |  |
|  |  |

|  |
| --- |
| **3. Java Reflection and Annotations:** |
| I made annotations that check the validity of my Java Code at run time and throw errors if there are problems. For instance, I have an annotation called @Verbatim that I use on constants.  If my constant is: TIRE\_PRESSURE Then it must equal: "tire\_pressure" |

|  |
| --- |
| **4. JSP page auto complete for rest services:** |
| Basically, you write your angular HTML page inside a .jsp page. Then all needed services are called using **minimally intrusive** .JSP expressions. Advantages:  1. You have auto-complete for all of the back end services as you are making the front end.  2. When the back-end refactors, the front-end is refactored with it.  3. If for some reason you do call a service that does not exist, the runtime build of the page will  break and notify you that you were calling a service that did not exist. |

|  |
| --- |
| **5. Thread-Safe (concurrency safe) static utility:** |
| **TransUtil.enterTransaction();** //do operations on database **TransUtil.exitTransaction()** Made it so all database transaction logic must happen between those two calls. Reasoning:  1. Self-documenting code. 2. Helps locate errors easily.   Example:  1. Forget one of the calls: Throws a bracket balancing error.  2. Forget all of the calls : Throws an error saying you tried to use database logic calls  while NOT inside of a transaction state. |

|  |
| --- |
| **5. Require.js:** |
| Learned it to make Serpinski Triangle banners.  It was a pain to learn. And took a while. |

|  |
| --- |
| **6. Food drive site:** |
| 1. Finished the site.  2. Make awesome snow flake maker.  **All in under a week. Because I had been teaching myself:**  1. **Angular**.js //awesome.  2. **Require**.js //complex javascript pretty much requires you use it.  3. **OpenShift** //didn't need to rely on someone else giving me dev server space. |

|  |
| --- |
| **6. AHK (Auto Hot Key): Failed Experiment:** |
| ALT+M And ALT JKLI Basically, I wanted to be more productive, and thought if I could use keyboard shortcuts to jump the mouse to different quadrants of the screen, and well as have a keyboard mouse-move mode... I could be faster... Didn't work out. |

|  |
| --- |
| **7. Wildfly:** |
| When I was doing the ticket monster tutorial I was able to configure my local host so anyone else in the building could access my server.   |  | | --- | | **[?wildflylan]** | |

|  |
| --- |
| **8. PHP MyAdmin + TomCat:** |
| Learned how to use PHP MyAdmin and deploy war files. |

|  |
| --- |
| **9. SSH, WINSCP, PUTTY:** |
| Had to learn how to SSH into a terminal for openShift account. I didn't know what SSH was before I came here.   |  | | --- | | **>putty [ppk]** | |

|  |
| --- |
| **10: Gradel + Maven+Java:** |
| Learned how to use them. Learned how to use them on the command line as well. Used notepad++ to make a new Maven Project, and then built the project on the command line. Benifit: **I am not dependent on the Eclipse or any other IDE if something goes wrong.**   |  | | --- | | [mvnpom] [?javacmd][?gradlecmd][?mavencmd] | |

|  |
| --- |
| **11. Learned how to use NetBeans and Intellij:** |
| Because I realize I hate Eclipse. |

|  |
| --- |
| **12. Misc Productivity Enhancers:** |
| All of my programs are linked to shortcut keys. I have multiple desktops organized by what they are used for. I use a pretty badass environment variable editor. >rapid WINDOWS+ALT+[F1,F2,F3,F4] |

line

|  |
| --- |
| **13. AHK: (AUTO HOT KEY) *(This is what I think I could do a talk on)*** |
| Cheat Sheets:  In order to fully grasp how awesome this is, you'll need to stop by for a demo. Basically, anything I want is a few keystrokes away.  Project workflow:   |  | | --- | | [?gtui] [/gtui] [grun.bat] [/admin] |   Pasting: [paste] //paste simulating typing. [pone] // paste as one line.  Angular Demo:   |  | | --- | | [html] <sap] <ctrl] |   Imports: !Angular] Or highlight + "!!" to expand import. |
| Why this is so awesome: 1. I am fast. 2. I have my own personal google.  All the answers are out there. The competitive edge is now: "How fast can you find the answers?" |

|  |
| --- |
| 14. AngularJS. |
| I used AngularJS + JSP while doing my version of the gauntlet. I am now using AngularJS with Express.js and Node.js on the official gauntlet. I think Node.js is a pretty cool technology. And I would like to eventually learn how to set up a Heroku Node.js server. |

|  |
| --- |
| 15. Git |
| I now have more experience in GIT. |

|  |
| --- |
| **New Technologies Learned, List:** |
| OpenShift, Java: Hibernate, Threads, Concurrency, Synchronization, Database connections, Jersey Servlets JavaScript: Canvas, Angular, Require.js Maven + Gradle (on command line + eclipse) Servers: Wildfly, Tomcat SSH, WinSCP, Putty AutoHotKey |

|  |
| --- |
| **Opprotunities for improvement:** |
| **GIT**... Learn GIT better. I know technically you "can't screw up anything in GIT". But if you are not careful about your workflow, you end up in "Merge Conflict Hell". Knowing GIT well helps prevent programmers from being counter productive. From my understanding, the reason there are no "Minimum wage" programmers is because certain lacks of knowledge are actually anti-productive. Rather than not contributing anything, a programmer who is not skilled enough can do real damage. I am constantly afraid of doing real damage to the project. Gulp on the UI side helps! |

|  |
| --- |
| **Future Career Goals:** |
| Anything **graphics** related probably. Maybe stuff involving **D3.js**? I think I could be useful working with **designers**. All the backgrounds on my UI's are animated and proceeduraly generated. I think if I were to work with the designers, we could pull off things that would be incredibly impressive. And impressing clients is how you get the contracts.   Optimizing graphics code for Mobile applications would be fun. No one likes a sluggish app.  Lower level? No one is doing C++ or C... But if it ever comes up, I want on it. |

|  |
| --- |
|  |
| **NOTE: Below here is heavily opinionated. I am aware of the Dunning–Kruger effect. So regardless of my opinion, I will do things the way I am told.** |
|  |

|  |
| --- |
| **Things you would like your manager to know:** |
| I had so many problems with how the project was being ran... That I wanted nothing to do with the backend... I saw a train wreck from day 1, when \_\_\_\_ thought it was a good idea to make ALL of the entities. I didn't want on that train. It was the beginning of a **NON-scrum** campaign as far as I was concerned.  I wanted to slowly build the project up. Start with one table with one column. And one UI. Have that one table and one UI talk to each other. Work up the back-end + front-end at the same time in small baby steps. I even made a **7 page design document** on all the baby steps needed to get up to basic token dispatching. On **Nov 2nd, 2015** I heard that the official gauntlet was NOT able to dispatch a test. **My version was already able to dispatch a test and grade it.**  I am not saying I am a better coder. Just that, the incrimental scrum approach I took made me more productive. I did a lot of stupid things. I didn't understand how to do join tables with Hibernate. So I made my own join-table management code. It's actually pretty neat. You fill out "**order slips**" for operations you want done on different tables. The order slips can list other order slips as **dependencies**. Then the order slips are all processed. I did this after writing a lot of horrible code. I realized I needed some way to specify only what I wanted done, without having to worry about what order it was done in. I think hibernate is great. But it's kind of like being given a magical chain-saw-shotgun when you don't even know how to use an axe yet.  This is not an insult to you Drew. I think you knew how it needed to be done. But with your expertise means you get pulled into a lot of different directions. I remember your conversation with \_\_\_\_ where he was asking about grading. And you said it was too soon for that. And I raised my hand and said, "How about we just stubb it with a function that returns a random number?" And you said, "**Yes, and I will choose the random number.... Zero**".  I wanted to remain cooperative, but still get something meaningful done.  I wish more people working here would read books on scrum. **Coding is not hard**. Putting all the pieces together and building something up, that is the challenge.  I may have... delayed my participation in the official gauntlet in order to do my self-study because I didn't want to be part of a waterfalled (non scrum) campaign. When I was hired. I was told this company was all about Agile methodology. And I really do embrace agile methodology. Not everyone in this company does. As scrum got incorporated into the project... A lot of damage had already been done by then.  While we are on that, we have too many back-end people. Any good book on agile will mention the lines of communications present among different group sizes:  My delay in joining the Gauntlet put me in a bit of an awkward situation. Because it probably appeared to Keith that I was doing absolutely no work. So in addition to learning a lot, I kept a pretty detailed personal record of what I had been doing every day.  I tried to remain cooperative in a decoupled fashion. And realized that if I was super productive with my version of the gauntlet, and did the UI stuff, that I would be able to prove myself worthy to work on the UI side of things.  **I enjoy where I am on the project now.**  Also, Gulp is pretty awesome. I don't know how to use it... But being able to prevent bad commits from being made is very helpful. |

|  |
| --- |
| Gauntlet Back-End Problems: |
| **1. Test questions:**  Who cares. It does not affect the functionality. In fact, for developement, simple obvious questions like: Q:"What color is the sky?" A:"Blue". Make more sense to me than Java specific questions.  **2. Database Migration:** Worrying about database migration before a working product has been made is kind of like buying a safe to store all the stuff that you might have in the future. Now you need FlyWay scripts and this other layer of complexity that isn't necessary yet.  **3. Lack of comments:** I read through the Gauntlet a few times. I couldn't understand it. And since it isn't built up in working baby steps, I never had anything to actually RUN so I could understand it.  I think my idea of comments is radically different than others. **Comments**: Declare what you want to do. **Code**: The ***attempt*** to do that thing in the comments. If code were truely self documenting, the bugs by definition, could NOT exist. Because: Code always does EXACTLY what you tell it to do. Code needs to be written in a way that takes into account that we are human and we make mistakes.  Tests are good. They definitely help the problem of making sure we know what a function should do. But I'd rather read a 2 line summary comment in a JavaDoc then hunt down the test code that may or may not exist, and then read that test code, just to figure out what a function does.  I'd be okay with a comment-less code-base if it had heavy documentation to go along with it.  While the lack of a descriptive variable name can signify a bad name. The lack of a descriptive comment can signify someone doesn't actually have full understanding  of what the code is supposed to do.  **4. Lack of concise variable names.**  See rant on "no problem domain". |

|  |
| --- |
| **Rant: No problem domain:** |
| **"I like cake".** Pretend "cake" is a silly made-up variable name. No one has invented cake yet. There is no word for it.  **"I like heat treated mixtures composed of sugar and flour".**  Pretend "flour" is a made up word. No one has a word for flour yet. **"I like heat treated mixtures composed of sugar and pulvarized wheat grass."**  Pretend "sugar" is a made up word. **"I like heat treated mixtures composed of crystalline surcrose derived from tropical grass and pulvarized wheat grass."**  Now turn that into a variable name, limiting ourselves to the words that exist. cake --> **heatTreatedCrystalSurcroseWheatMixture** We have a very specific and concise idea that is being thrown around in the code WITHOUT having an equally concise and specific variable name. At some point in time, someone just needs to be like, "Hey, lets all agree that Cake means '**heat treated mixtures composed of crystalline surcrose derived from tropical grass and pulvarized wheat grass' "** One problem I see in the gauntlet is that "answer" and "question" mean different things in different contexts.  For example, to disambiguify this, in my own version I call an an answer that can be selected on an exam by the candidate as a "quip". Which differentiates it from the answers within the answer table.  But the problem is, we seem to want to write code that can be immediately understood. Without the reading of ANY DOCUMENTATION. Without the constraint of "read the f\*ing manual" before proceeding, we are not at liberty to make up concise variable names for concepts.  Someone had to invent the word "read", "persist", and "serialize". And those inventions make code more readable. "serializeExam()" is more readable than "convertExamInstanceToTextRepresentation()" |