COMP 3110 Fall 2020

Take-Home Midterm Examination

University of Windsor

Instructions:

This take-home exam is handed out on November 3rd, 2:30 pm and due back no later than 2:30am November 4th, 2020. The exam is individual based and includes short questions. There is no make-up or extension to this exam due to its flexible nature. This exam is worth 10% of your final marks.

Type all your answers. Submit on blackboard as PDF document. Use a font size of 12pts, single spaced, with 1" margins for your paper. Use letter format, 8.5"x11" (21.59cm x 27.94cm) size paper.

Do your own work! Cheating is not tolerated. Your answers must be **unique, and sufficiently different**. Answers that are found to be similar or mere duplicate of each other will receive a mark of zero.

You must sign the following agreement and submit this page as the title page for your submission:

"I hereby certify that the solutions submitted for this exam are solely my own work."

Print Name: Jose Guillen Santos Date: 11/3/2020

Student ID: 104985300

Signature: [illegible]

The mid-exam includes general software engineering skills: the type of questions you may encounter in a real job interview. Please support your answer with examples or diagrams whenever possible. Barebones version of the answer will not be acceptable.
Answer **any 8** of the following questions. Limit each answer to half a page (minimum of 1/3 page per question and do not exceed 4 typewritten pages in total - marks will be deducted if you exceed). [1.25 points each question, equal weight].
If anyone do all 10 questions, first eight only will be graded.
Don’t use any example from online or other resources. Use your own example or experience to get full points.

1. What are your favorite programming languages? Why did you pick these over others?
2. Describe your ideal development environment?
3. What values do daily builds, automated testing, and peer reviews add to a project? What disadvantages are there?
4. Explain the major difference between iterative and incremental development processes based on example.
5. Given Time, Cost, Client satisfaction and Best Practices, how will you prioritize them for a project you are working on? Explain why.
6. What is Refactoring? Have you used it and why is it important? Name three common refactoring.
7. You have just been assigned to a project in a new technology how would you get started?
8. Given that you are required to design and implement a CMS (Content Management System); describe the structure and contents of a design document, or a set of design documents for this CMS.
9. What is a version control system? Have you used it and why is it important?
10. Explain the concept of separation and independence of software architecture and its pros and cons.

1. What are your favorite programming languages? Why did you pick these over others?

My favorite programming languages are Python and Java. The main reason that I chose these languages was because I am proficient in both, which means that my development time and debugging time are much shorter compared to other programming languages. This allows me to be more efficient in accomplishing my goals in my projects. It's my increased efficiency with the languages that make Python and Java my favorite programming languages. However, my proficiency is not the only advantage of using these languages over others. Both languages are well documented and very popular among other developers. Consequently, this makes collaborating on a Python or Java project much easier than most languages. This combined with the fact that both languages are easy to learn makes group projects with these languages one of the most ideal experiences in my opinion. Ultimately, it’s the combination of these languages maximizing my efficiency, and their ability to enhance my collaboration experience, that these two languages are my favorite.

2. Describe your ideal development environment?

My ideal development environment is one that is flexible to my needs and the needs of my team, but without being overly laissez-faire. Structure and routine are important to a team for them to be efficient, which is why I relish the opportunity to work in a development environment that has a clear structure. Therefore, my ideal development environment has clear goals and deadlines, but is flexible and agile enough in order to support different ways of thinking and problem solving. My ideal environment has short iterations, quick meetings, and promotes high velocity by minimizing unnecessary bureaucratic processes. This is my preference because this will help my team stay on task and focused, and remove any unnecessary distractions and slowdowns. This keeps our efficiency high, and therefore we can continue to progress with our goals and the goals of our organization. Moreover, my ideal development environment places an emphasis on refactoring and mitigating problems before they become out of control. In other words, my ideal environment does not contain any “boiling frogs” that do not realize that there are any problems until they reach a “boiling point” where it is too late to fix anything. Chiefly, my ideal development environment is structured, agile, and emphasizes forward thinking and refactoring.

3. What values do daily builds, automated testing, and peer reviews add to a project? What disadvantages are there?

Daily builds add value because you are constantly updating your product. Hence, you are implementing this business technique known as “kaizen” which refers to small, consistent, and gradual improvement. So, rather than have large improvements every few months, you have smaller gradual improvements every day. This makes your organization flexible and adaptable in the long run, and therefore increases the organization’s longevity. Moreover, it allows you to catch bugs and issues in builds early on in development which prevents them from becoming larger issues. A disadvantage of daily builds would be the burnout your teams could potentially face if gone unchecked. So, it’s important to constantly check in on your teams and make sure that the tasks they've been given are not overwhelming. Automated testing adds value by reducing human errors and ensuring certain consistency throughout all testing since they are not subject to human error. However, a drawback to automated testing would be the initial costs to implement automated testing since some automation tools can be very expensive. Finally, peer reviews add value by minimizing errors and bugs that get pushed to the live build. The more errors you can prevent from happening, the more fruitful your development will be because you are saving time in the long run by not having to fix these large mistakes later. However, a disadvantage would be the additional time spent by my team members to conduct these peer reviews which could draw them away from doing their own tasks, but this time is well spent because preventing mistakes is one of the best uses of company time. Ultimately, daily builds, automated testing, and peer reviews all have pros and cons, but the pros usually outweigh the cons especially when you're implementing these practices in large organizations.

4. Explain the major difference between iterative and incremental development processes based on example.

The major difference between iterative and incremental development processes is that the former focuses on developing in cycles where each iteration you improve on the last by refining the work done on the previous iteration and it's not a linear development model. Now, the latter, which is an incremental development process, focuses on developing in a linear manner in which you cut the work into smaller pieces called increments and each increment adds a small bit of functionality every time. For example, a team with an iterative approach to building a fence would build a fence by first quickly putting up a fence, then ensuring its straight, and then would finally refine the wood and paint it. Now, an iterative team would first refine the wood, then paint it, make sure the measurements are correct, and then finally they would put up the fence. Ultimately, the difference between the iterative and incremental development processes is that the latter is more linear and builds directly on top of their past work, while the former focuses on refining their past work in cycles so they don't have to worry about having exact requirements.

5. Given Time, Cost, Client satisfaction and Best Practices, how will you prioritize them for a project you are working on? Explain why.

Given Time, Cost, Client Satisfaction, and Best Practices, I would prioritize them in the following manner: Best Practices, Client Satisfaction, Cost, and Time. Best practices should be prioritized above all else because they are the tried and tested techniques which have guaranteed successful and ethical projects. If we were to value anything over best practices, we run the risk of violating an ethical guideline in order to make the client satisfied, to ensure we stayed under budget, or to ensure the team stayed within the timebox. Therefore, it may be unethical to prioritize anything above best practices, so it should be our first priority. Client satisfaction should be the second priority because they are the ones who will either be using the product or be affected by its outcome. Therefore, from an ethical perspective, it would be irresponsible to put cost or time above best practices because you would be valuing money and your time over your client’s best interest which is an inherent ethical violation. Finally, we should value cost over time because we are able to delay products as long as we can ensure that the finished product will have a high quality. However, if we run out of funding, there’s not much one can do to delay bankruptcy. So, while our time is valuable and we should aim to try to push developments as fast as possible, we can not make any real progress without managing our costs. Ultimately, we should prioritize these values in this manner to ensure our project is ethical and successful.

6. What is Refactoring? Have you used it and why is it important? Name three common refactorings.

Very abstractly, refactoring is the process of improving code without affecting the codes functionality. This improvement could come from restructuring the code, renaming methods and variables, replacing inline code with method calls, or improving the code’s readability. In any case, the purpose is to make the code cleaner, tidier, and easier to work with in order to make it easier to implement functionality later. Therefore, refactoring is important because it helps increase your team’s velocity by making the codebase easier to work with. Furthermore, “clean” code is much easier to debug, understand, and manipulate so this will also increase your team’s productivity in these areas. Increased productivity means more goals are being accomplished which means that the organization is thriving. So, it all starts with refactoring because it's much easier to be efficient and productive when the tools you use are well maintained. Lastly, I have refactored code before and three common refactorings (which I have done) are red-green refactoring, litter-pickup refactoring, and comprehension refactoring. Red-green refactoring means that I have written enough code to pass test cases, then I have refactored that code to make it as clean as possible. Litter-pickup refactoring means that I was working on a project and found messy code, and stopped to refactor it before continuing to add additional functionality. Comprehension refactoring means that I have refactored code that didn’t make sense after reading it a few times, so I refactored it to improve readability.

7. You have just been assigned to a project in a new technology how would you get started?

The best way to get started learning a new technology is to read and understand as much of the documentation as possible. It's important to get the gist of the technology’s strengths, limitations, and specifically why my team chose to use this technology in order to achieve our goals. After this I would like to spend time perusing our repository to get an understanding of any stylistic preferences when using this tool, but also to grow my understanding of how this tool can be used. I would also make it a point to identify the person in our team who is the bonafide expert of this technology and ask him about the common mistakes that new learners make when using this tool in order to minimize the mistakes I make and lower the probability that I push a bug to production. By doing this, I would ensure a base level understanding of the new technology as well secure an understanding of the quality of work expected from me by the team. After this I would have to rely on my deep background in computer science and problem solving in order to handle any issues that would arise from learning a new technology. Overall, this methodology would ensure that I minimize any decreases to my team’s velocity and set me up to be a contributing member of my team in the long run.

9. What is a version control system? Have you used it and why is it important?

A version control system (VCS) is a system or software which tracks changes in files. Specifically, it is used in software development most notably to track changes and modifications to code. Usually, VCSs have a repository which can be thought of as a database of changes. Essentially, these track snapshots of your project over a period of time. They will also usually contain a copy of your current work so that people can edit the file and then commit their changes to a repository. So, it's clear to see why VCSs are so important and play such a vital role in software development. Being able to track changes to a large project is very important because it allows team members to branch off and work on separate pieces of the project and then merge these ideas together at the end without worrying about not having the latest version of the codebase. Moreover, it allows teams to revert back to old versions if an unexpected bug arises in their latest version, so it ensures a greater level of security. So, it’s because of the safety that version control provides, and the greater collaboration that it enables that it’s important to software development. These are also the reasons I have used VCSs like Git for my own schoolwork. Being able to track changes and work on projects from different machines is very functional and useful for a student such as myself. So, I make sure to use version control systems whenever possible in order to back up my work and to create easy collaboration opportunities.