**Question 1:**

**This is a software Testability Checklist**

**1. Operability – the better it works the more efficiently it can be tested**

**2. Observability – what you see is what you test**

**3. Controllability – the better software can be controlled the more testing can be automated and optimized**

**4. Decomposability – by controlling the scope of testing, the more quickly problems can be isolated and retested intelligently**

**5. Simplicity – the less there is to test, the more quickly we can test**

**6. Stability – the fewer the changes, the fewer the disruptions to testing**

**7. Understandability – the more information known, the smarter the testing**

**Choose two and explain why they have higher merit than the others, remember to provide enough depth so that you convey that you understand the depth of the question.**

From the software testability checklist Understandability has the highest merit. In software testing this is important because the more a tester understands about the project, the better they can go through and ensure the proper requirements have been fulfilled. Since testing is an integral part of software development, understanding the code and its requirements should weigh pretty high on that scale. It makes little sense to have a tester test something that they do not understand or comprehend. Ultimately, having more insight and knowledge leads to higher quality tests. Software development is all about user satisfaction and it is up to the tester to verify that the product meets the client’s demands. Having a tester who truly understands the needs of the clients and the programs technical ability, leads to better testing and happier clients.

Second on that list would be Observability, this is the concept of the tester only being allow to test what they can physically see. This is done to mimic testing from a user’s point of view. This is important because it helps developers ensure that all the clients’ requests can be observed and accounted for. The more observable information is, the deeper a tester can go into finding issues and bugs. If something is being tested but the outcome isn’t easily observed then it becomes an issue from a testing standpoint. It is recommended requirements be observable so they can be tested and fixed before the final product is revealed. If the final outcome doesn’t have observable outcomes the client will begin to question if the program really worked as intended.

**Question 2:**

In OO Testing,

What is the difference between surface and deep structure testing and how does it benefit the overall testing of OO designs?

When should they be used when why is the environment important?

What kind of project would require each, give three examples of each?

Object-Oriented testing allows testers and software engineers to test classes, objects, functions, and over all code structure in an OO architecture. This allows for a more robust understanding of tests performed in an OOP application.

Surface structure testing allows testers to test the application from the user’s point of view and typically operates more on the front end of the application. Surface structure testing involves manipulating objects and performing tasks exhausting all user choices. In order to do this testers need to understand the processes in which the user goes through to complete tasks. Testers will start at the surface of these tasks and then exhaust resources inwards as the process begins to pan out. Essentially surface structure testing’s is testing performed from the top down in the point of view of the user and then trying all combinations possible. This is beneficial to the overall OO testing because it allows testers to better understand requirements from a user’s point of view. Surface structure testing would be used in projects that weigh heavy importance on user interfaces. Some examples would be a drag and drop map application with a nice graphical interface. Another could be a game with a main character walking a storyboard. Lastly, another example of a program you would test surface structure testing would be some kind of banking application, where surface structure would be tested from initial log in, all possible transactions, then logout. These are all examples of programs with user interfaces that can be tested in a surface structure manner.

Deep structure testing can be thought of opposite of surface structure testing in regards to it is testing done from more of a inwards out approach behind the scenes from the user, and is typically performed on the back end. Deep structure can only be understood by reviewing the underlying code. Deep structure testing allows for testing of behaviors, dependencies, and communications between objects and classes. This benefits OO testing because it allows testers to better understand back end process and relationship that are not easily observable to the client or on the front end. A tester would implement deep structure testing when relationships between objects or process are to be better understood and tested.

An example of this would be if you have a program with a user object and an admin object. Deep structure testing would test the relationship between the two and see how much functionality a user would have compared to an admin. Another example would be a dog class and a cat class that both implement the same animal interface. Deep structure testing would test how both dog and cat implement the animal class differently. Last another example of deep structure testing would be to see how all inherited classes interact with each other. This could be conveyed as a child class inheriting a parent class inheriting a grandparent class and how they affect each other is tested in deep structure testing.

**Question 3:**

In Formal modeling and verification what is the Z Specification language?

Why is it used and what are the drawbacks of formal modeling, Give three examples of software projects (applications) that would require Formal modeling.

The Z specification language is one of the most widely used and regarded for formal modeling and verification. This language allow for modularity through schemas and allows processes to be visualized and later built. It is used to design and map out processes to later be built regardless of what programming language is chosen. This allows for a single uniform design to be adopted and then applied later on down the line. These graphical implementations are used among colleagues to verify business processes and set a baseline design. Some draw backs of formal modeling is that it is pretty technical in the fact that it is not common knowledge for most people. This can be time consuming and expensive to an organization to have to pay for someone with the technical prowess or to have someone trained in formal modeling. One example of an application that would use formal modeling would be game theory where complex algorithmic process would need to be proved out in a formal specification language. Another example would be in GPS imaging, where complex processes would be need to be mapped out regarding the multiple data entry points coming into the system. Lastly, another example of a program that would use formal specification would be any project regarding science or physics such that the complex equations would need to be proved out before being implemented in a native programming language.

**Question 4:**

In Measurement Process Activities, the following list applies:

1. *Formulation* – derivation of software measures and metrics appropriate for software representation being considered

2. *Collection* – mechanism used to accumulate the date used to derive the software metrics

3. *Analysis* – computation of metrics

4. *Interpretation* – evaluation of metrics that results in gaining insight into quality of the work product

Of the above four items, list the order of focus,, highest to lower which items requires more work than others, Give a detailed explanation of why you think this is so.

When it comes to which items require more work than others, interpretation takes the most work. This is followed by formulation, analysis, and collection in that order.

To begin interpretation takes the most work and can be pretty cumbersome. Gaining insight into the quality of the work in the product is an important task but can be quite tedious. Measurement results can have a plethora of information and it can be challenging to extract useful data. The challenge is team members need to truly understand the measurement results and how their individual roles affect the development process. Arguably, interpretation takes the most work because it challenges one to think outside the box and always be innovating. Interpretation constantly challenges how processes can be approved by making decisions based on the data’s interpreted value. That is why great thought and effort should be taken into account for interpretation.

Just under interpretation is formulation. Formulation should also take a great amount of work because a solid focus needs to be had on what questions need to be answered along with what tools to answer them with. Understanding how to derive the proper metrics for you project is crucial in gaining the correct insight to later be interpreted. This can take up resources because teams need to work together to agree on what metrics need to be measured and how.

Analysis is right under formulation because be it true that analysis can be tedious in that it is the computation of metrics. This is arguably less intensive because it is more logic based and straight forward rather than thinking outside the box and abstractly like in interpretation and formulation.

Lastly, the least amount of work is collection. Collection is also more logic based because the means of collecting the metrics were already laid out in the formulation stage. One simply just needs to apply the formulated logic already blue printed out for them to apply a means of collecting metrics and then let the data be collected to be later analyzed and interpreted.

**Question 5:**

What are the major differences between Process and Project Metrics and why are they important to software engineering?

Software metrics can be classified as process or project metrics. These metrics are measurable parts of software. The major differences are that project metrics gauge the quality of the project and quantifies concerns to be evaluated. These concerns include topics like cost of the project, estimation of resources, the schedule, and how productive the project is. This is different than process metrics because they gauge the quality of the development process itself not the project. Process metrics measure how effective a given process is and whether that process is helping deliver the product in the most efficient way. An example would be if the process to request new software is efficient or not and how can it be improved. They are both important to software engineering because roject metrics insure that the project is of highest quality with least amount of resources while process metrics verify that the process to build code becomes more effective over time.

**Question 6:**

How does Software Re-engineering fit into the agile development cycle, explain in detail, how to combine re-engineering process model and the agile process model

Software re-engineering fits in the refactoring part of the agile development life cycle. Re-engineering is about changing the structure and design of software that is already functional, for better maintainability. Refactoring in the agile process allows for changes to the product that produce the same behaviors as before. Since both do not change the behavior of the outcome we can see where re-engineering would fit in the agile process of refactoring. One can combine the re-engineering and agile process models by performing re-engineering in the refactoring phase of the agile development life cycle. An example of this would be an already finished application hosted on local servers and a refactoring ticket was created to have that same application hosted on a cloud environment. This would be an example of re-engineering in an Agile development process. Although, re-engineering would mean that a faulty agile development cycle was used on the previous legacy software.

**Question 7:**

If you are in an agile development cycle how would you incorporate the use of UML, does it have a place and how strongly should it be implemented?

User Markup Language can be used in an agile development life cycle. This is because UML can be used to help define if its user requirements are being addressed or not. UML is a good way to effectively communicate processes to stake holders before development begins. Having the ability to visualize these processes before development starts has a huge place in agile development. In agile development it is important to have communication between the developers and the stakeholders and having a UML express that communication is an added benefit. If UML is used throughout the entire process it gives the client a frequent opportunity to see the product. This helps verify that the product is not going off course and that requirements are being met. Since stakeholders are not typically non-tech savvy, UML helps convey these products in layman’s terms to them. UML should be implemented strongly in the agile development process because it helps communicate processes to stakeholders that would typically not be able to comprehend their technical counterparts, this is used to verify how process are being incorporated and verifies requirements are being met.

**Question 8:**

Regarding Clear-Box, State-Box and Black-Box quality management approaches,

Explain the uses of each?

Give an example of each?

Why you would use 1 over the other 2 in each case?

Clear box testing is when a tester has full knowledge of the applications underlying architecture and can fully test all behaviors and relationships between the front and back end of the program. An example of this would be logging into an application as an administrator vs a normal user. When a user logged in they would want to perform a black box test on the back end to verify the user is an administrator regardless of what is conveyed on the front end. Testers would use clear box testing in this case because State-Box and Black-Box testing are not concerned with the behind the scenes back end-code. State-Box testing relies on the transition of different states throughout the application. An example would be the initial login page, then testing the successfully logged in home page, then testing the next functionality, until finally logging out and testing that functionality as well. Testers would use State-Box testing when a focus on business flow and the processes of the application is needed. Tests would use this over Clear-Box testing because they are not concerned with how the processes our getting achieved between the front and backend, but rather if the flow of the processes can be exploited. They are not using Black-Box testing because they are more concerned with the flow of the processes rather than the results from initial input.

Black-Box testing is when a tester begins a procedure without any prior information or knowledge about the underlying architecture of the system. They perceive and test the application exactly as a new user would in the market. They base their results of off the outputs from the tested inputs, oblivious to what’s going on behind the scenes in the code. An example of this would be any user input scenario where a tester would put in some initial input and get a result back such as a login or an order form. Testers would prefer block-box testing over clear-box and state-box testing when they are not concerned on the underlying architecture or the flow of processes, but rather are interested in how the client interfaces with the application.