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.

Understandability is the notion that when more information is known about software, the more effective testing it can be. Since testing is one of the most important aspects of software development, understandability should be ranked very high on the software testability checklist. Testing is essentially where all of the bugs and defects are found in software so that they can be addressed before the product reaches the hands of the customer or client. Developers and testers are always looking for smarter and more efficient ways to complete their work in a timely manner, so smart testing is very important to the overall development process. Smarter development and testing can lead to more projects being completed, which would in turn bring more capital to the team and the company overall. Testers should understand both how the developed software works and the initial requirements so that they can properly test whether the built software satisfies those requirements. In the end, software comes down to user satisfaction with the product that they receive so it is important that testers understand exactly what the client wants. Essentially, the more the tester understands, the more they can go through the software to ensure all requirements are fulfilled.

Observability is the notion that what the tester is physically able to see is what they will test. This is also of utmost importance because the tester is able to see exactly what the users of the software will see. Not only is this important for testers, but developers should ensure that all client requirements are observable so that they can be properly tested before reaching the client. The more observable a piece of software is, the further and deeper the tester can go into it to identify any underlying bugs and defects that could cause the software to function incorrectly. When more defects can be identified, more issues can be brought to the attention of developers who can then fix these issues to provide a more stable build of the software. It could be argued that poor observability can lead to a poor user interface. What is observable and what the user can see is what allows the user to navigate through the software. User interfaces that are implemented well are what make software simple to use, which in turn makes the lives of its users much easier. Essentially, it is very important for all requirements to be observable so that they can be tested, fixed, and used when the final product is launched.

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?

OO (Object-Oriented) testing is the testing of individual classes and objects function as defined and function properly with the overall code structure. Object-oriented testing is conducted by both software engineers (as more objects and classes are created, they must ensure that other instances have not been negatively affected) as well as testers. Testers test the overall code to ensure that the application or software works according to specifications and requirements without any significant defects. Object-oriented testing is comprised of both surface and deep structure testing in order to ensure that all elements of the software are observed for defects. This allows for more systematic testing of object-oriented software.

Surface structure testing refers to the testing of the externally observable aspects or surface/front-end part of an object-oriented application or piece of software. This involves the structures and interface that are immediately visible to the user of the software. Surface structure testing involves manipulating objects and performing user tasks using every possible user choice. This type of testing involves understanding the process that the user or client needs to go through to complete the task or tasks that were defined in the initial requirements. The tester essentially puts themselves in the shoes of the client and uses the software or application as they would in order to see if there are no defects with any choice that the user could possibly make. The tester uses a testing checklist with each object to ensure that they are all functional at the surface/front-end level.

Deep structure testing refers to the testing of the internal, technical details of an object-oriented application or piece of software. This generally involves the back-end/deep part of the software that the user or client is not able to physically see. This is the structure that can only be understood by observing the underlying code, design, and implementation of the application. Deep structure testing involves the testing of the relationships between each object. More specifically, these tests check the dependencies, behaviors, and communication mechanisms between the classes and objects and how the entire system is affected by these relationships. Object-oriented programming is the use of different objects that work together with one another, and these relationships can become quite complex as more objects are added to the code, so it is absolutely essential to test all of these relationships during development. If deep structure testing begins after development is complete, it can be a very tedious and time-consuming process. Essentially, surface structure testing is used to ensure that no issues arise on the front-end/user interface side of the application, while deep structure testing corrects any issues on the front-end and observes the relationships between each object.

Projects that would need to use surface structure testing are those that are very reliant on the user interface to complete tasks. Almost every piece of software today is reliant on a robust and intuitive user interface. One example would be an address book application where users input contacts into a smartphone or other smart device. This is fully reliant on the user interface to complete the task of creating and editing contacts. This would need to be tested in order to ensure that contacts can be created properly; otherwise the application would be rendered useless. Another example of surface structure testing would be a calendar application. Many objects are used to perform tasks such as adding events to a certain date, setting reminders, displaying notifications and many more. Testers would need to ensure that the user interface allows the user to perform these tasks without error, or again this application would be rendered useless. Lastly, another example of this type of testing would be an ATM or ATM simulation applications. All of the different transaction types would make up different objects and the interface would need to be tested to ensure that deposits and withdrawals work correctly.

Projects with intricate relationships between classes would need to implement heavy deep structure testing. The previously stated examples could also be used here because all object-oriented applications use object relationships to function. These relationships are what make the programs object-oriented. Both surface structure and deep structure testing must be implemented to ensure that an object-oriented application is functioning to its maximum potential.

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 required Formal

modeling.

In formal modeling and verification, the Z specification language is the most widely used formal specification language. The language is based on typed set theory and upon schemas, which serve as basic building blocks, allow modularity, and provide graphical representations for better understanding. Because of these graphical representations, systems designed with a formal specification have better designs and implementations upon completion. However, formal modeling does come with some drawbacks, which include its time consuming and expensive nature. Formal modeling is highly technical, and therefore it can sometimes be quite difficult to use this model as a communication mechanism between team members who do not have highly technical backgrounds. Due to this issue, extensive training is usually required for those who do not have technical experience, which requires additional time and money. Once these team members have the required training, this model can be implemented in the development process. An example of a software project/application that would need to use formal modeling would be in insurance claim processing. A formal model would help to ensure that the proper sequence of processing is maintained, and that particular events will always lead to the same correct outcome. Formal modeling essentially helps to ensure that failures do not occur in large distributed systems where errors would be catastrophic. Another example of formal modeling use would be in NASA systems. Whether these systems are in spacecrafts or not, they need to be absolute and without failure so it is important that formal modeling is used in their creation. Fault monitors need to be put in place to inform users of any outstanding issues. Lastly, any banking system that handles the funds of the general public need to have proper formal modeling so that no issues arise with people not receiving their money. Similar to insurance claim processing, banking systems should always ensure that particular events always lead to the correct outcome. For example, if a user of an ATM does not input the correct PIN, there should always be a message showing that the PIN was invalid so that the user is not able to access funds that are not theirs.

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.

The order of focus for the measurement process would be interpretation, formulation, analysis, and collection. Collection is the step in which data about products, processes, and projects are gathered so that software metrics can later be derived. In other words, collection includes the mechanism that is used to gather the data needed for the metric derivation. This is where it is decided how the measurement will be determined. Formulation includes the derivation of metrics and measures that are appropriate for the software that is being considered. In other words, formulation involves finding what exactly needs to be measured based on the software being used or created. Analysis of these derivations is conducted to identify any errors or inefficiencies in the data so that any necessary improvements can be made both to the product itself and as well as to the measurement process. Lastly, the highest focused; interpretation is the evaluation of metrics to gain insight on the quality of the software and the software development process. The developer or project team will analyze their measurement results during this step. This is done to ensure that the product will be of its highest possible quality after completion.

Interpretation probably requires the most work and the most meticulous work. Interpretation is where the developers, along with the project team including testers, analyze the measurement results. This can be time-consuming, as each member of the team needs to understand the measurement results and how each individual is affecting the development process. Interpretation is also the measurement step where it is determined as to how the results can improve the team’s development process. Every team member should have a clearly defined role in the development process and individual roles can always be improved upon to improve the process as a whole. Interpretation allows project teams to see where they might not be as strong and they can then decide the best course of action for improving their work as one cohesive unit. Interpretation is essentially used to improve performance and efficiency in developers as well as other team members, and should be implemented for companies to grow. When performance and efficiency are improved, more projects can be completed within a timely manner, which leads to higher customer satisfaction and increased company capital.

Question 5:

What are the major differences between Process and Project Metrics and

why are they important to software engineering?

To begin, a metric is a measureable attribute of any piece of software. Software metrics are classified as either process metrics or project metrics. Process metrics are used to measure the efficiency and effectiveness of the software development process itself, while project metrics are used to quantify defects, costs, schedule, productivity, and the estimation of project resources and deliverables. Both are necessary to ensure that there are no resources including time and money are being squandered during the software engineering process. Project metrics allow developers to assess the status of an ongoing project, track potential risks, uncover problem areas before they go “critical,” (meaning catastrophic and far more difficult to contain and handle) adjust work flow or tasks, and evaluate the project team’s ability to control quality of software work products. On the other hand, process metrics allow developers to check the cost of quality, which includes review, testing, verification review, verification testing, quality assurance, measurement, and training. These attributes all need to be measured to ensure that the development process is delivering the highest quality product to the client in most timely and efficient manner possible. If the cost of quality is high and the efficiency of the process is low, then the process needs to be re-evaluated and fixed before proceeding to other projects. Essentially, both process and project metrics are absolutely necessary to software engineering because one improves the development process efficiency and delivers a quality product (process) and the other ensures that no defects are present in the final product (project).

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 is the process of examining and altering a system to reconstitute it in a new form in order to improves its maintainability. This process encompasses many sub-processes such as reverse engineering, restructuring, forward engineering, and retargeting. This is done because it is sometimes more cost effective to re-engineer existing software rather than creating software from scratch when components are already available. Rather than changing functionality, software re-engineering is typically concerned with improving performance, scalability, security, quality, usability, and consistency. However, form an agile point of view, software re-engineering can be problematic. The agile development cycle is usually used to create new products and complete them in a quick and efficient manner where every team member has an assigned role. Since no new functionality is being created during the software re-engineering process, it does not usually make sense to implement the agile development cycle in the process. Re-engineering is concerned with addressing changes after team members have learned new skills and want to implement them within existing software. Although this would be cost effective when it comes to legacy systems which always need to be re-engineered to keep up with changing technology over many years, it would not be cost effective to re-engineer software that is still fully functional. This is not to say that the software would not be improved, but from an agile standpoint where there are clearly defined requirements, architecture, and design attributes, re-engineering a technically completed product does not make sense. There are no new requirements or architecture needs, as only performance and quality are being improved upon in re-engineering. Additionally, if a system that is not a legacy system genuinely needs to be re-engineered, then it shows how ineffective the initial agile process was so it does not make sense to implement the same process during the re-engineering phase. Lastly, this poses a redundant testing situation. Re-engineering will cause new bugs and defects to arise, so the testing in the first engineering phase would be rendered useless and a decent amount of time and resources would be considered wasted. Essentially, the testing for the application would need to start from the very beginning again. For these reasons, it usually does not make sense to combine the software re-engineering process with the agile development process unless it is a very useful and expensive legacy system that is being considered. In this case it would cost far more time and money to create a new system when there is already a relatively stable system to build from.

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?

The agile development cycle is a process for creating products where iterations are created and released in short bursts so that risks and defects are minimized while expanding the capabilities of the product over time. Every team member has a clearly defined role in this process, and this helps the process to be efficient with both time and resources. UML (Unified Modeling Language) is used to provide a standard way to visualize the design of a system before starting its creation. This type of visualization definitely has a place in the agile development process, especially during the early stages when requirements are being considered by both the customer/client and the development team. It can sometimes be difficult for clients and development teams to be on the same page when it comes to the final product that needs to be created, and therefore it helps if there is a tool to help visualize the end product for both parties. Developers can show clients how they visualize the product through different models and changes can be addressed early in the process before any sort of development begins. Through modeling, the client has frequent opportunity to see the product at almost every stage, as UML is used throughout the entire process. Software development really comes down to client/customer satisfaction, as they are the ones that the software is being developed for. It helps to have models to show so that the envisioned product does not go askew throughout each stage of the development process. Additionally, since clients are usually not technologically savvy, it helps to have models to show them their product in a layman’s term fashion. Essentially, the use of UML helps to ensure that requirements and architecture for the client’s product are ironed out from the beginning and it even helps to change requirements when necessary. Clients are able to see exactly what the product that they are receiving looks and feels like throughout the development process and easily inquire about any changes that need to be made so that it is more aligned with their vision. For these reasons, UML should be implemented very strongly in the agile development process.

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?

The clear-box (also known as transparent-box or glass box testing) quality management approach is where functionality and internal code are focused on. This is the type of testing where functionality bugs/defects are identified, code is optimized, and code coverage is increased. This approach is used when complete transparency of underlying code is necessary, hence its names referring to see-through objects. This testing helps to rid systems of any latent defects that could cause systems to run inefficiently or even crash. The previous example regarding the address book can be applied here. In this case the tester would need to observe the underlying code and ensure the class and object relationships are working properly to ensure that the overall application will function according to requirements and specifications. Essentially this approach should be used to ensure that the back-end code is cooperating with the front-end to produce a working user interface.

The state-box quality management approach is used when there is a set, finite number of transition states available. A certain input can be made and the system will proceed to another state as the output. In this approach, the user is not concerned with underlying code and functionality. An example of the state-box approach would be an ATM system, where inserting an ATM/debit card serves as an input, which triggers the next state where the user’s PIN is requested. A valid PIN will trigger the next state, while an invalid PIN will cause the system to retain the current state. States will continue to change until the user’s request for a deposit or withdrawal is complete. This approach is essentially used when users do not need to be concerned with underlying code and need to go through a series of steps or states to fulfill a certain request.

The black-box (also known as the behavioral approach) quality management approach is used when details of the functionality of the given application are needed without any knowledge of the underlying code that is implemented. Instead of using underlying code for testing purposes, an input is given to the system and the final output is tested. In contrast to the clear-box approach, the black-box approach checks to ensure that the front-end of the application is working according to standard. There should be no defects present in the user interface or any front-end part that the user is able to see. The same example of the address book could be used here as well, but instead of testing the underlying classes and objects, the user interface is tested to ensure all front-end functions are in order. The tester would try all possible user inputs to ensure functions such as adding or editing a contact are working properly. Essentially this approach is used to ensure the front-end is functioning properly and if it is not, then the tester may need to look at the application at a deeper level.