NAME: M OMAR MASOOD

ID: 14305

SUBJECT: Object Oriented Software Engineering

DEPARTMENT: BS(SE) 5th Semester

Submitted to: Mam Sanaa Jeehan

Question #1 (10 Marks)

In Software Engineering, there is not a single answer to the question "What should be done first, Coding or Modeling?". Elaborate different scenarios in which all the answers to this questions are justified.

ANSWER NO 1:

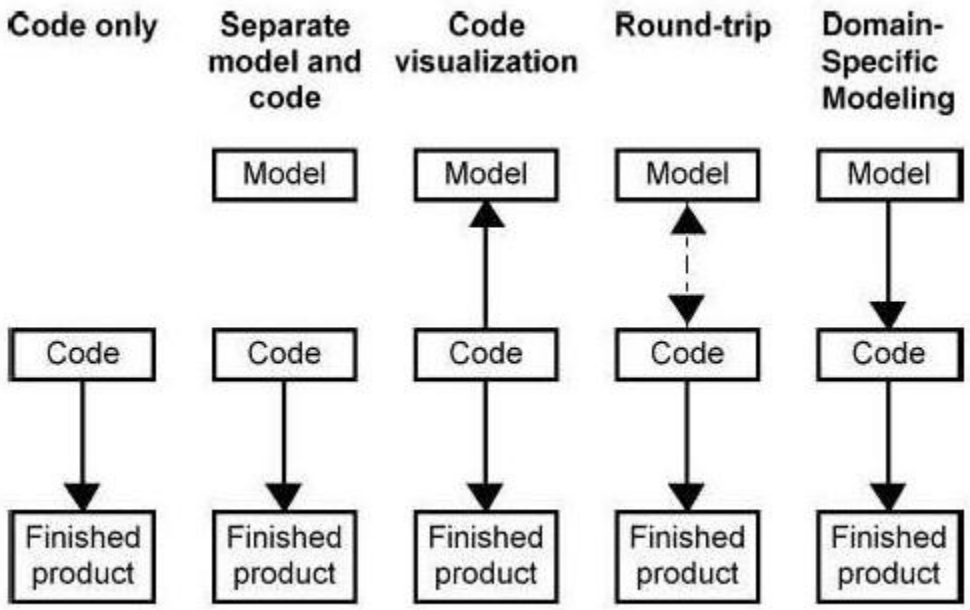
Developers generally differentiate between modeling and coding. Models are used for designing systems, understanding them better, specifying required functionality, and creating documentation. Code is then written to implement the designs. Debugging, testing, and maintenance are done on the code level too.

Most software developers, however, also create models. Pure coding concepts are, in most cases, too far from the requirements and from the actual problem domain. Models are used to raise the level of abstraction and hide the implementation

details. In a traditional development process, models are, however, kept totally separate from the code as there is no automated transformation available from those models to code. Instead developers read the models and interpret them while coding the application and producing executable software.

During implementation, models are no longer updated and are often discarded once the coding is done. This is simply because the cost of keeping the models up-to-date is greater than the benefits we get from the models. The cost of maintaining the same information in two places, code and models, is high because it is a manual process, tedious, and error prone.

Following image shows the Aligning code and models



During implementation, models are no longer updated and are often discarded once the coding is done. This is simply because the cost of keeping the models up-to-date is greater than the benefits we get from the models. The cost of maintaining the same information in two places, code and models, is high because it is a manual process, tedious, and error prone.

Models can also be used in reverse engineering: trying to understand the software after it is designed and built. While creating model-based documentation afterwards is understandable, code visualization can also be useful when trying to understand what a program does or importing libraries or other constructs from code to be used as elements in models. Such models, however, are typically not used for implementing, debugging, or testing the software as we have the code.

One approach to solve this problem is to use just a single source, usually the code, and show part of it in the models. A classical example is to use only part of the expressive power of class diagrams. That part is where the class diagram maps exactly to the class code.

In model-driven development, we use models as the primary artifacts in the development process: we have source models instead of source code. Throughout this book, we argue that whenever possible this approach should be applied because it raises the level of abstraction and hides complexity. Truly model-driven development uses automated transformations in a manner similar to the way a pure coding approach uses compilers.

Once models are created, target code can be generated and then compiled or interpreted for execution. From a modeler's perspective, generated code is complete and it does not need to be modified after generation. This means, however, that the "intelligence" is not just in the models but in the code generator and underlying framework.

Question #2 (10 Marks)

When carrying out Testing of a Software, a number of techniques are used. Why are they so many in number? Name a few popular Testing Techniques in Software Engineering and state the importance of each one.

ANSWER NO 2:

Techniques:

The goal of utilizing numerous testing methodologies in your development process is to make sure your software can successfully operate in multiple environments and across different platforms. These can typically be broken down between functional and non-functional testing. Functional testing involves testing the application against the business requirements. It incorporates all test types designed to guarantee each part of a piece of software behaves as expected by using

uses cases provided by the design team or business analyst. These testing methods are usually conducted in order and include:

- Unit testing
- Integration testing
- System testing
- Acceptance testing

Non-functional testing methods incorporate all test types focused on the operational aspects of a piece of software. These include:

- Performance testing
- Security testing
- Usability testing
- Compatibility testing

Why are in Numbers?

The key to releasing high quality software that can be easily adopted by your end users is to build a robust testing framework that implements both functional and non-functional software testing methodologies that's why there are in numbers.

Importance

Unit Testing.

Unit testing is the first level of testing and is often performed by the developers themselves. It is the process of ensuring individual components of a piece of software at the code level are functional and work as they were designed to. Developers in a test-driven environment will typically write and run the tests prior to the software or feature being passed over to the test team. Unit testing can be conducted manually, but automating the process will speed up delivery cycles and expand test coverage. Unit testing will also make debugging easier because finding issues earlier means they take less time to fix than if they were discovered later in the testing process. Test Left is a tool that allows advanced testers and developers to shift left with the fastest test automation tool embedded in any IDE.

Integration Testing

After each unit is thoroughly tested, it is integrated with other units to create modules or components that are designed to perform specific tasks or activities. These are then tested as group through integration testing to ensure whole segments of an application behave as expected (i.e., the interactions between units are seamless). These tests are often framed by user scenarios, such as logging into an application or opening files. Integrated tests can be conducted by either developers or independent testers and are usually comprised of a combination of automated functional and manual tests.

System Testing

System testing is a black box testing method used to evaluate the completed and integrated system, as a whole, to ensure it meets specified requirements. The functionality of the software is tested from end-to-end and is typically conducted

by a separate testing team than the development team before the product is pushed into production.

Acceptance Testing

Acceptance testing is the last phase of functional testing and is used to assess whether or not the final piece of software is ready for delivery. It involves ensuring that the product is in compliance with all of the original business criteria and that it meets the end user's needs. This requires the product be tested both internally and externally, meaning you'll need to get it into the hands of your end users for beta testing along with those of your QA team. Beta testing is key to getting real feedback from potential customers and can address any final usability concerns.

Security Testing

With the rise of cloud-based testing platforms and cyber attacks, there is a growing concern and need for the security of data being used and stored in software. Security testing is a non-functional software testing technique used to determine if the information and data in a system is protected. The goal is to purposefully find loopholes and security risks in the system that could result in unauthorized access to or the loss of information by probing the application for weaknesses. There are multiple types of this testing method, each of which aimed at verifying six basic principles of security:

1. Integrity

2. Confidentiality
3. Authentication
4. Authorization
5. Availability
6. Non-repudiation

Usability Testing

Usability testing is a testing method that measures an application's ease-of-use from the end-user perspective and is often performed during the system or acceptance testing stages. The goal is to determine whether or not the visible design and aesthetics of an application meet the intended workflow for various processes, such as logging into an application. Usability testing is a great way for teams to review separate functions, or the system as a whole, is intuitive to use.

Compatibility Testing

Compatibility testing is used to gauge how an application or piece of software will work in different environments. It is used to check that your product is compatible with multiple operating systems, platforms, browsers, or resolution configurations. The goal is to ensure that your software's functionality is consistently supported across any environment you expect your end users to be using.

Testing With TestComplete

TestComplete is our robust automated GUI testing tool that excels in compatibility and integration testing. It helps QA teams create and run tests across desktop, mobile, and web applications – enabling testing professionals to speed up delivery cycles and improve software quality. Testcomplete comes with built-in support for various test environments, integrations to performance testing tools, as well as support for developer friendly SCMs, allowing you to seamlessness integrate it into your development process. Using TestComplete will enable you to build a robust testing framework that utilizes the broad spectrum of available software testing methodologies.