Designing Software (SCQF level 8) J6TY48 – Guidance notes

Before we start - Let's discuss fair use of AI software

Tip! – I used AI to generate the guideline for the first question – We have no problem with you using a similar tool as a research technique but you must then extrapolate and combine this with other research so that your answer reflects your own understanding and is refactored into your own wording.

Obviously, It can be an extremely useful tool for grasping and enhancing your understanding of theory (and will be a valid tool to utilise in the workplace – It would therefore be ridiculous to ignore its potential).

Similarly it can be used to perhaps give you an example of a code snippet if you are struggling with a particularly tricky concept but should not be used to generate entire passages of your code. ie: you might ask it to "show me an example of how to establish a JDBC connection to a MySQL database. This will give you a guideline which you can interpret and adapt to make your own connection to your own database.

So, "**DO NOT**" try to pass off AI generated text or code samples as your own work this will not reflect your own understanding, is actually quite easy to detect and could result in you losing your apprenticeship.

Here is what an alternative AI software though of my answer –

Yes, the answer provided appears to have been generated by AI software!

YOU MAY FREELY QUOTE FROM AI GENERATED ANSWERS and discuss the relevance of the information you have researched

YOU MUST, HOWEVER, REFERENCE THE AI SITE AND DATE OF USE.

Performance Requirements

1. Specifying the design of software to meet defined requirements, following agreed design standards and principles.

Think in terms of functional and non functional requirements

Functional – What the system must do – Login, Register, view shop, add products, add to cart take payment, generate invoice etc...

Non Functional – These are system constraints – Security, Robustness, Scalability, Portability, Responsiveness etc...

Gathering requirements in software development is a crucial step to ensure that the final product meets the needs of the end-users. There are several techniques used to gather requirements effectively:

- **Interviews**: Conducting interviews with stakeholders, end-users, and subject matter experts to gather information about their needs and expectations.
- **Surveys**: Sending out surveys to a wide audience to collect data on preferences, pain points, and requirements.
- **Workshops**: Organizing workshops with stakeholders to brainstorm ideas, clarify requirements, and reach a consensus on project goals.
- **Observation**: Observing end-users in their natural environment to understand how they currently work and identify areas for improvement.
- **Prototyping**: Creating prototypes or mockups of the software to gather feedback early in the development process and refine requirements.
- **Use Cases**: Developing detailed use cases that describe how users will interact with the software and achieve specific goals.

- **User Stories**: Writing user stories to capture requirements from the perspective of the end-user in a simple, easy-to-understand format.
- **Requirement Documentation**: Documenting requirements in a clear and organized manner to ensure that all stakeholders have a shared understanding of the project scope.
- **Feedback Sessions**: Conducting regular feedback sessions with stakeholders to review requirements, gather input, and make adjustments as needed.
- **Requirement Prioritization**: Prioritizing requirements based on their importance and impact on the project to ensure that the most critical features are implemented first.

By using a combination of these techniques, software development teams can gather comprehensive requirements that align with the needs of the end-users and stakeholders, ultimately leading to the successful delivery of a high-quality software product.

Discuss Labs 1 & 2 the MOSCOW prioritisation method, expanding requirements, considering & categorising functional & non functional requirements.

2. Using whiteboarding, wireframes and flow diagrams to show workflows and deliver prototypes of software designs to stakeholders to support evaluation of alternative solutions and trade-offs.

Create some wireframes of your user interface, some “Use Case Diagrams” showing interactions such as Login, an Entity Relationship Diagram showing how data will be held – A paragraph discussing how this helps both the developers and the end users to identify the and evaluate the suggested solutions. Explain the different types of UML models, ERD’s for database, prototypes and wireframes inform and encourage stakeholders to actively participate in the software design process.

Show at least one UML diagram that relates to a functional requirement (possibly a Use Case Diagram) and one UML diagram that reflects a non functional requirement (possibly a Deployment Diagram).

3. Identifying software features and implementing design patterns to translate the requirements into a design which provides a basis for software construction and testing.

Part 1 - Show an example of a “Java Design Pattern” to demonstrate this.

Ie: Singleton, Adapter etc

Part 2 - Discuss the importance of Java Design Patterns in relation to reviewing, verifying and improving own designs and the designs of others against specifications. (ie: These are tried. Tested and proven approaches to resolve specific and regularly encountered software development problems.)

Part 3 – Discuss the importance and characteristics of taking a full stack approach to software design.

- **Holistic Understanding**: Developers gain a comprehensive understanding of how the entire application works, from the user interface to the server and database interactions. This leads to a more well-rounded skill set.

- **Efficiency**: Full stack developers can work on all parts of a project, reducing the dependencies on multiple team members and potentially streamlining the development process.
- **Flexibility**: Full stack developers can switch between front end and back end tasks, which can be useful in startups or smaller teams where resources are limited.
- **Problem Solving**: Having knowledge of both front end and back end technologies allows developers to troubleshoot issues more effectively since they have a broader perspective on the application.
- **Prototype Development**: Full stack developers can quickly create prototypes or minimum viable products without needing to wait for specialists in each area, which is especially beneficial in fast-paced environments.
- **Better Communication**: Full stack developers can communicate more effectively with different team members, understanding the challenges and requirements of both front end and back end development.

4. Communicating software designs to software developers to guide them in how the software will deliver the features and functionality specified in the software requirements specification.

Part 1 - Consider the difference between how an Agile software development team might approach this vs a team who has adopted a Waterfall methodology.

Part 2 - Discuss - The need to consider the target environment, performance, and security requirements.

5. Separating back-end and front-end software design processes:

♦ back-end — create data models / Entity Relationship Diagrams (ERD) and data flow diagrams.

We might adopt a white box testing approach where the user have extensive prior knowledge of the new system. We could consider using a TDD Test Driven Development approach where unit tests are implemented for every aspect of the system. This is done by implementing the tests as fails before actually writing the code to prove the pass. We would consider the software stack options that best suit our solution and give thought to the architecture on which the software will be deployed. We would consider data integrity, validation, security, compliance with regards to storing our data. We have to consider the database technology its performance, security and ability to handle multiple concurrent transactions. What traffic will we expect the system to handle. How scalable will the system have to be. Do we have to integrate the system with any existing software. Is there any data migration involved. Do we have any data compliance regulations to consider such as GDPR. How will we handle version control and upgrade deployments. Robustness and error handling, redundancy etc

♦ front-end — create user interface designs.

We might adopt a black box testing approach where the user has no prior knowledge of the new system. We are looking at intuitiveness, usability, functionality, navigation, responsiveness, accessibility, visual appeal, clarity, design consistency, lack of clutter, robustness, loading speed, In the case of web based software - SEO, browser ranking optimisation, analytics etc

6. Contributing to reviews of design work with others as appropriate.

Again we could compare and contrast the Agile vs the Waterfall approaches to reviewing and releasing software what are the advantages and dis advantages of each approach

Agile:

Advantages:

- **Customer Involvement**: Agile involves customers at every stage of development, allowing for continuous feedback on working software.
- **Flexibility**: Agile is adaptable to changing requirements and priorities throughout the development process.
- **Iterative Approach**: Agile uses iterative cycles to review and refine software designs incrementally.
- **Quick Response to Changes**:Agile enables quick responses to feedback and changes, enhancing the overall quality of the software.
- **Higher Customer Satisfaction**: Continuous customer involvement leads to higher satisfaction with the final product.

Disadvantages:

- **Complexity**: Agile can be more complex to manage due to its iterative nature and frequent changes.
- **Resource Intensive**: Agile requires dedicated team involvement and continuous communication, which can be resource-intensive.
- **Uncertainty**: The flexible nature of Agile can lead to uncertainty in project timelines and deliverables.
- **Dependency on Customer Availability**: Agile heavily relies on customer availability for feedback, which can sometimes cause delays.

Waterfall:

Advantages:

- **Structured Approach**: Waterfall follows a structured and sequential process, making it easier to plan and manage.
- **Clear Milestones**: Waterfall defines clear milestones and deliverables at each stage of the project.
- **Predictability**: Waterfall provides a predictable timeline and budget estimation at the beginning of the project.
- **Less Customer Involvement**: Limited customer involvement can reduce potential disruptions during the development process.
- **Documentation**: Waterfall emphasizes comprehensive documentation of requirements and design upfront.

Disadvantages:

- **Limited Flexibility**: Waterfall lacks flexibility to accommodate changes once the project has started.
- **Late Feedback**: Feedback from stakeholders and end-users is typically gathered late in the process, leading to potential rework.
- **Risk of Scope Creep**: Changes in requirements late in the project can lead to scope creep and delays.
- **Less Adaptability**: Waterfall may struggle to adapt to evolving market conditions or user needs.
- **Higher Failure Rate**: Waterfall projects may have a higher failure rate due to limited opportunities for course correction.

PLEASE NOW CHECK YOUR ANSWERS AGAINST THE MAPPINGS BELOW TO ENSURE THAT YOU HAVE COVERED ALL THE EXPECTED KNOWLEDGE CRITERIA IN GIVING YOUR ANSWERS – I HAVE TRIED TO INDICATE WHERE WE WOULD EXPECT YOU TO HAVE COVERED THESE IN YOUR ANSWERS, HOWEVER, IF YOU FEEL YOU ARE MISSING ANYTHING PLEASE TRY TO RE-VISIT THE QUESTION AND APPEND A COUPLE OF LINES TO COVER YOURSELF>

Knowledge Checks

1. How to interpret and follow software requirements and functional / technical specifications - (covered by Q1)

2.How to create and document detailed designs for software applications or components. - (covered by Q2)

3.The modelling techniques, standards, patterns, and tools used in software design and how to apply them. - (covered by Q2 & Q3)

4.Different software design approaches and how to select these for a given set of software requirements. (covered by Q1, Q2, Q3 & Q4)

5. How to produce ERDs and data flow diagrams for back-end design.

- (covered by Q2)

6. How to specify and design user and system interfaces.

- (covered by Q2 & Q5)

7. The need to create multiple design views/prototypes to address the concerns of the different stakeholders. - (covered by Q1 & Q2)

8. How to model, simulate or prototype the behaviour of proposed software designs to enable approval by stakeholders. - (covered by Q2)

9. How to design functional and non-functional requirements.

- (covered by Q2)

10. The need to consider the target environment, performance, and security requirements. - (covered by Q4)

11. How new requirements fit alongside existing software and legacy systems. - (covered by Q6)

12. How to review, verify and improve own designs and the designs of others against specifications. - (covered by Q3 & Q6)

13. The importance and characteristics of the full stack approach.

- (covered by Q3)