Sugestão de respostas para a avaliação

Questão 01 [1,5] - Possível resposta:
Os três princípios fundamentais da Engenharia de Software que devem ser considerados ao desenvolver o sistema de informação para a organização de sobreviventes no universo de The Last of Us são:

- Princípio da modularidade: O sistema de informação deve ser desenvolvido em módulos independentes e coesos, para que as mudanças em um módulo não afetem os outros. Isso é importante para garantir a manutenibilidade e a escalabilidade do sistema em um ambiente em que recursos são limitados e as mudanças precisam ser feitas rapidamente.
- Princípio da segurança: O sistema de informação deve ser projetado com medidas de segurança robustas, como criptografia de dados e autenticação de usuários, para garantir que as informações confidenciais da organização de sobreviventes não caiam em mãos erradas. Isso é especialmente importante em um mundo pós-apocalíptico, onde a segurança é uma preocupação constante.
- Princípio da documentação: O sistema de informação deve ser documentado de forma clara e concisa, para que os desenvolvedores possam entender como o sistema foi projetado e implementado. Isso é importante para garantir a manutenibilidade do sistema e para permitir que outros desenvolvedores possam trabalhar no sistema em caso de necessidade.

Esses princípios são importantes para garantir a eficácia e a segurança do sistema de informação em um ambiente pós-apocalíptico porque, em um mundo em que recursos são limitados e a sobrevivência é uma prioridade, é crucial que o sistema seja fácil de manter, seguro e bem documentado. A modularidade garante que o sistema possa ser alterado com rapidez e eficiência, sem afetar outras partes do sistema. A segurança garante que as informações confidenciais da organização não sejam comprometidas, e a documentação garante que o sistema possa ser mantido e atualizado por outros desenvolvedores, caso seja necessário.

Questão 02 [2,5] - Possível resposta:
Abaixo, segue um exemplo de possível resposta que apresenta cinco histórias do usuário para a nova funcionalidade:

I. Como um membro da organização de sobreviventes, eu quero ser capaz de localizar pontos de água potável no mapa para poder me orientar e coletar recursos com mais eficiência.
II. Como um líder de equipe, eu quero ser capaz de atribuir tarefas específicas a cada membro da equipe, para que possamos nos dividir melhor e concluir as tarefas mais rapidamente.
III. Como um sobrevivente em busca de recursos, eu quero ser capaz de marcar locais no mapa onde já encontrei itens valiosos, para que eu possa voltar a esses locais mais tarde.
IV. Como um sobrevivente em busca de recursos, eu quero ser capaz de avaliar a qualidade dos itens encontrados (por exemplo, alimentos que ainda estão frescos ou armas que ainda têm munição), para que eu possa decidir o que levar comigo.
V. Como um líder de equipe, eu quero ser capaz de visualizar as tarefas atribuídas a cada membro da equipe e o status de cada tarefa (por exemplo, em andamento, concluída ou atrasada), para que eu possa gerenciar melhor o trabalho da equipe.

Essas cinco histórias do usuário abordam diferentes necessidades dos usuários para a nova funcionalidade e fornecem uma descrição clara e concisa dos requisitos que devem ser atendidos. O uso de Metodologias Ágeis e Teste A/B ajudará a validar essas histórias do usuário e garantir que a nova funcionalidade atenda às expectativas dos usuários.

O teste A/B é uma técnica de experimentação que envolve a criação de duas versões diferentes de uma mesma funcionalidade. Uma versão é o grupo de controle (ou versão A), que representa a funcionalidade atual ou a versão anterior, enquanto a outra versão é o grupo de teste (ou versão B), que apresenta as alterações propostas para a nova funcionalidade.

Os alunos poderiam responder que primeiro definiriam as métricas que serão usadas para avaliar a eficácia da funcionalidade, como taxa de conversão, tempo de resposta, entre outros. Em seguida, aplicariam a funcionalidade do grupo de controle para um grupo de usuários e a funcionalidade do grupo de teste para outro grupo de usuários.

Após a coleta dos dados, os alunos poderiam analisar os resultados e verificar qual das duas versões obteve melhor desempenho em relação às métricas definidas. Se a versão do grupo de teste apresentar um desempenho melhor do que a versão do grupo de controle, os alunos poderiam concluir que a nova funcionalidade atendeu às expectativas dos usuários e implementá-la no sistema.

Caso contrário, os alunos poderiam retornar à etapa de Engenharia de Requisitos, revisar as histórias do usuário e fazer ajustes na funcionalidade até que os resultados do teste A/B indiquem uma melhoria significativa em relação à versão anterior.

Questão 03 [3,0] - Possível resposta:
A resposta esperada para essa questão pode variar dependendo da interpretação dos alunos sobre a organização de sobreviventes no universo de The Last of Us e a tecnologia disponível no mundo pós-apocalíptico retratado no cenário fictício descrito. No entanto, de forma geral, os alunos podem responder que uma Arquitetura de Software é uma descrição abstrata do sistema, que representa a sua estrutura, comportamento e interações entre os seus componentes.

Os principais conceitos que compõem uma Arquitetura de Software incluem:

- Componentes: partes do sistema que desempenham uma função específica.
- Conectores: elementos que permitem a comunicação e interação entre os componentes.
- Configuração: a forma como os componentes e conectores são organizados em relação à estrutura do sistema.
- Restrições: requisitos não funcionais que devem ser considerados durante o projeto da Arquitetura, como desempenho, segurança, escalabilidade, entre outros.

Os elementos que contribuem para a qualidade do software incluem:

- Manutenibilidade: capacidade do sistema de ser modificado facilmente para atender a novas necessidades ou corrigir erros.
- Testabilidade: capacidade do sistema de ser testado de forma eficaz para garantir que ele funcione corretamente.
- Escalabilidade: capacidade do sistema de se adaptar a um aumento de demanda sem comprometer o seu desempenho.
- Confiabilidade: capacidade do sistema de funcionar corretamente em todas as condições de uso.

Os alunos podem ainda citar outros elementos que considerem importantes, como portabilidade, usabilidade, entre outros. Eles devem explicar como a escolha da Arquitetura de Software adequada pode influenciar na qualidade do software e, consequentemente, no seu sucesso no mercado.

Os alunos devem explicar como a escolha de uma Arquitetura de Software adequada poderia contribuir para a sobrevivência dos personagens, permitindo que eles se comuniquem, gerenciem recursos e tomem decisões mais eficientes em um mundo pós-apocalíptico.

Questão 04 [1,5]

Questão 05 [1,5]

Questão Extra [2,0] - Possível resposta

A resposta esperada para essa questão pode variar dependendo da interpretação dos alunos sobre a organização de sobreviventes no universo de The Last of Us e os sistemas de informação utilizados pelas pessoas no cenário descrito. No entanto, de forma geral, os alunos podem responder que os princípios SOLID podem ser aplicados em diversos aspectos dos sistemas de informação utilizados pelas pessoas para torná-los mais robustos e flexíveis em face dos desafios enfrentados no universo do cenário descrito.

Alguns exemplos de como os princípios SOLID poderiam ser aplicados incluem:

- Princípio da Responsabilidade Única (SRP): cada componente do sistema deve ter apenas uma responsabilidade, tornando mais fácil a identificação e correção de problemas no código e permitindo que os personagens adicionem novas funcionalidades sem afetar outras partes do sistema.
- Princípio do Aberto/Fechado (OCP): o código deve estar aberto para extensões, mas fechado para modificações, permitindo que os personagens adicionem novas funcionalidades sem modificar o código existente.
- Princípio da Substituição de Liskov (LSP): as classes devem poder ser substituídas por suas subclasses sem que o comportamento do sistema seja afetado, permitindo que os personagens utilizem diferentes classes ou objetos para atender às necessidades específicas de cada situação.
- Princípio da Segregação de Interface (ISP): as interfaces devem ser segregadas para que cada componente dependa apenas das interfaces que precisa, permitindo que os personagens criem novas interfaces específicas para cada necessidade.
- Princípio da Inversão de Dependência (DIP): os componentes de alto nível do sistema devem depender apenas de abstrações e não de implementações específicas, permitindo que os personagens alterem ou substituam as implementações sem afetar o comportamento do sistema.

Os alunos devem explicar como os princípios SOLID poderiam ser aplicados em diferentes aspectos dos sistemas de informação utilizados pelas pessoano cenário descrito, incluindo exemplos específicos de como cada princípio poderia ser aplicado e como isso contribuiria para tornar o sistema mais robusto e flexível em face dos desafios enfrentados no universo do cenário descrito.