***Aluno: Wesley Erik Sardi e Davi Prudente Ferreira***

* 1. C

1.2)

Escolhemos a alternativa C, pois ela segue a sequência do processo de Test Driven Development (TDD), no qual o teste é escrito antes do código. Essa abordagem de desenvolvimento serve para garantir a qualidade do código e a colaboração entre os membros da equipe, permitindo que as funcionalidades sejam implementadas de forma mais segura e com testes automatizados, guiando o desenvolvedor e fazendo com que o código utilize o teste como base de desenvolvimento.

2.1)

Supondo que possuamos uma Epic que seja: "Desenvolver um sistema de reserva de hotel". Algumas possíveis User Stories que podem ser extraídas dessa Epic são:

1. Como um usuário, eu quero poder realizar uma busca por hotéis disponíveis nas datas que desejo, para poder escolher onde me hospedar.
2. Como um usuário, eu quero poder visualizar os detalhes do hotel, incluindo fotos, descrição, comodidades, para poder avaliar e tomar uma decisão informada sobre a reserva.
3. Como um usuário, eu quero poder realizar a reserva de um quarto de hotel, selecionando as datas e o tipo de quarto desejado, para poder garantir a minha hospedagem.

Cada User Story representa uma funcionalidade ou requisito específico do sistema de reserva de hotel, e pode ser desenvolvida de forma independente, permitindo que os desenvolvedores realizem adições de features e entregas frequentes para o cliente.

2.2)

Os critérios de aceitação são condições específicas que definem como a User Story deve ser implementada para que seja considerada completa e aceita. Eles devem ser claros, mensuráveis e verificáveis para facilitar a validação. Alguns exemplos de critérios de aceitação podem incluir funcionalidades específicas, limites de tempo ou desempenho, comportamentos desejados ou evitar comportamentos indesejados, entre outros.

3.1)

Um teste unitário é uma técnica de testes de software que tem como objetivo validar a funcionalidade de uma unidade isolada do programa, como um método ou uma classe. Ele ajuda a identificar problemas no código de forma rápida e precisa, garantindo que cada unidade de código funcione corretamente e atenda aos requisitos definidos. Os testes unitários são escritos pelos desenvolvedores e podem ser automatizados, o que facilita sua execução repetidamente ao longo do ciclo de desenvolvimento. Além disso, eles proporcionam benefícios adicionais, como aumentar a confiança na estabilidade do código e facilitar a manutenção e a refatoração.

3.2)

Mocking em testes unitários refere-se à criação de objetos falsos (chamados de "mocks") que simulam o comportamento de objetos reais com os quais uma unidade de código em teste interage. Esses objetos falsos são usados para isolar a unidade de código em teste de suas dependências externas, garantindo que o teste seja focado apenas nessa unidade. Em vez de usar objetos reais, que podem ter comportamento complexo ou impactar o ambiente externo, os mocks permitem controlar as respostas simuladas e os estados dos objetos falsos, tornando os testes mais previsíveis e de fácil manutenção. Isso também ajuda a reduzir a complexidade dos testes e a melhorar a velocidade de execução, pois não é necessário configurar completamente o ambiente externo para cada teste.

4.1)

Refatoração é o processo de alterar o código-fonte de um software sem alterar sua funcionalidade externa, com o objetivo de melhorar sua estrutura interna e torná-lo mais legível, compreensível e manutenível. Isso envolve a reorganização do código, a eliminação de duplicações, a simplificação de trechos complexos, a melhoria da nomenclatura e a aplicação de padrões de projeto. A refatoração não adiciona novas funcionalidades, mas busca otimizar o código existente.

No XP (Extreme programming), a refatoração é uma prática fundamental. Os programadores são encorajados a refatorar constantemente o código durante o desenvolvimento, mantendo-o limpo e de alta qualidade. A refatoração é considerada parte do processo de desenvolvimento e é realizada de forma contínua, à medida que o código é escrito.

A refatoração no XP ajuda a manter o software flexível, adaptável e fácil de evoluir. Ela permite a melhoria constante do código enquanto novos recursos são adicionados. Além disso, a refatoração também cria um ambiente de trabalho mais colaborativo, facilitando a revisão do código por outros membros da equipe e a detecção de possíveis problemas e melhorias.

4.2)

A programação em pares é uma prática em que dois programadores trabalham juntos em um mesmo código, compartilhando uma estação de trabalho. Essa abordagem tem como objetivo melhorar a qualidade do código e aumentar a produtividade.

Os benefícios da programação em pares incluem:

1. Maior qualidade do código: duas pessoas revisam o código em tempo real, o que ajuda a identificar e corrigir problemas e erros mais rapidamente.
2. Maior habilidade e conhecimento compartilhado: os programadores aprendem uns com os outros enquanto trabalham juntos, o que resulta em um aumento de habilidades e conhecimento em toda a equipe.
3. Melhor entendimento do problema: a colaboração entre os programadores ajuda a entender melhor os requisitos e o problema a ser resolvido.
4. Menos bugs: com duas pessoas verificando o código, a probabilidade de erros diminui.
5. Melhor solução: a combinação de ideias e perspectivas diferentes ajuda a encontrar melhores soluções para os problemas.
6. Aumento da produtividade: o trabalho conjunto permite que os programadores se apoiem mutuamente, resultando em uma maior velocidade de desenvolvimento.

4.3)

O TDD no XP tem como objetivo garantir que a implementação do código esteja correta e atenda aos requisitos do sistema. Ele também ajuda a manter a qualidade e a estabilidade do código, uma vez que todos os testes automatizados devem sempre passar. No XP, o TDD é aplicado da seguinte forma:

1. O desenvolvedor escreve um teste automatizado que descreve uma funcionalidade desejada.
2. Em seguida, o código de produção correspondente é desenvolvido, implementando a funcionalidade.
3. O código é continuamente refatorado para melhorar sua qualidade, mantendo todos os testes automatizados passando o tempo todo.
4. Esse ciclo é repetido para cada nova funcionalidade ou modificação do sistema.

5.1)

Nós utilizaríamos as seguintes métricas:

1. Velocity (Velocidade): Já que ela realiza a medição da quantidade de trabalho que a equipe é capaz de entregar em cada iteração.
2. Lead time (Tempo de Execução): Pois mede o tempo decorrido desde o início de uma solicitação até sua conclusão. Quanto menor o lead time, mais eficiente é o processo. Isso indica que as histórias de usuário ou as tarefas são entregues rapidamente.
3. Customer Satisfaction (Satisfação do Cliente): Visto que ela avalia a satisfação do cliente em relação aos resultados alcançados e ao valor entregue pelo sistema.
4. Quality (Qualidade): Porque é um ponto onde é avaliado a qualidade do software produzido pela equipe. Métricas como taxa de defeitos encontrados, cobertura de testes, número de problemas relatados pelos usuários podem ser utilizadas para realizar a avaliação do produto desenvolvido.
5. Team Happiness (Satisfação da Equipe): Já que mede o nível de satisfação e engajamento da equipe.

5.2)

User Stories e Epics são técnicas utilizadas no desenvolvimento de software ágil para facilitar a comunicação entre desenvolvedores e stakeholders não técnicos. User Stories são descrições curtas de funcionalidades na perspectiva do usuário, enquanto as Epics representam requisitos de grande escala que são divididos em User Stories menores. Essas técnicas ajudam a traduzir as necessidades dos stakeholders em termos compreensíveis, promovendo uma comunicação efetiva e alinhamento entre as equipes envolvidas.