Fundação CECIERJ - **Vice Presidência de Educação Superior a Distância**

Curso de Tecnologia em Sistemas de Computação
Disciplina: Engenharia de Software
AP3 1° semestre de 2017.

Nome –

Assinatura –

Observações:

1. Prova sem consulta e sem uso de máquina de calcular.
2. Use caneta para preencher o seu nome e assinar nas folhas de questões e nas folhas de respostas.
3. Você pode usar lápis para responder as questões.
4. Ao final da prova devolva as folhas de questões e as de respostas.
5. Todas as respostas devem ser transcritas nas folhas de respostas. As respostas nas folhas de questões não serão corrigidas.

---

1) Porque o uso de bons nomes para variáveis e rotinas facilita o entendimento de um programa? (valor: 2,0 pontos; máximo: 10 linhas)

O uso de nomes adequados e claros para as variáveis e rotinas contribui para aumentar a legibilidade e compreensibilidade do código, o que, por sua vez facilita seu entendimento e com isso traz vantagens para sua evolução e manutenção. O código-fonte de um programa de computador é um artefato bastante complexo e difícil de entender. Assim, todos os esforços devem ser feitos para que estes artefatos sejam mais legíveis, de modo a reduzir as chances de um desenvolvedor introduzir um erro no sistema por desconhecimento sobre como realizar uma alteração no código. Comentários, boa organização, simplicidade, boa nomenclatura para os componentes do programa (como classes, métodos e variáveis) são ferramentas que auxiliam na clareza dos programas. Bom nome para rotina ajuda o desenvolvedor a entender seu objetivo, enquanto um bom nome para as variáveis e parâmetros facilita o entendimento da informação ali armazenada no contexto da rotina. Por exemplo, considerando o domínio de problema relacionado a sistemas financeiros, a leitura e compreensão do saldo de uma conta bancária seria facilitada caso as variáveis que compõem a fórmula de cálculo apresentem nomes realísticos e adequados, tais como saldo, retirada, depósito ao invés de x1, x2 e x3.

2) Explique a relação existente entre os processos da área de conhecimento de tempo e da área de conhecimento de escopo em um projeto de desenvolvimento de software. (valor: 2,0 pontos; máximo: 10 linhas).

O escopo determina o trabalho que deve ser realizado no contexto de um projeto. Em projetos de software, os processos de gerenciamento de escopo determinam as funcionalidades que serão consideradas na construção do produto, ou seja, define as tarefas necessárias para que o projeto seja realizado com sucesso. Estas funcionalidades devem ser construídas de acordo com um cronograma, que é desenvolvido a partir de uma lista de atividades e de suas dependências, identificadas nos processos da área de conhecimento de tempo. Assim, o principal resultado da área de conhecimento de tempo, ou seja, o cronograma do projeto, é determinante para identificarmos a ordem com que as tarefas serão executadas e permitindo identificar quando os recursos devem estar disponíveis para o projeto.

3) Uma ferramenta CASE é utilizada para apoiar as atividades de desenvolvimento de software. Entretanto, sua escolha e uso dependem de algumas características do processo de desenvolvimento e do produto a ser construído. Indique uma característica associada ao processo e uma associada ao produto que afetam a escolha das ferramentas, explicando seu significado. Em complemento, cite dois exemplos de ferramentas CASE e explique como, quando e para quê elas podem ser usadas em conjunto no processo de desenvolvimento de software. (valor: 2,0 pontos; máximo: 10 linhas).

Características que devem ser observadas: Escopo coberto pela ferramenta, Ambiente de Hardware, Ambiente Operacional, Tipo de sistema de software desenvolvido, Natureza do trabalho, Facilidade de Aumento de Escopo, Geração de código, Geração de Casos de Teste, Facilidades de personalização a metodologias, Documentação produzida, Interface com o Usuário, Apoio ao aprendizado, Segurança.

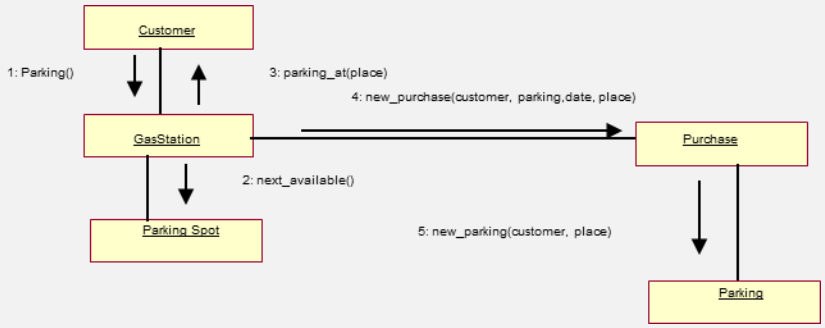
Exemplos de ferramentas CASE se relacionam a ambientes de apoio a codificação (e.g. ECLIPSE), diagramadores de modelos UML (e.g. BOUML, Enterprise Architect, StarUML, dentre outras são representantes deste grupo). Elas podem ser usadas nas fases de codificação e especificação de requisitos e projeto (ver exemplos adicionais no material do curso).

4) Explique para quê serve um diagrama de colaboração e quando devemos considerar sua construção num projeto de software? Apresente como um diagrama de colaboração é normalmente representado em UML, indicando seus elementos constituintes e as possíveis relações. (valor: 2,0 pontos; máximo: 10 linhas).

Um diagrama de colaboração é útil para mostrar a existência de múltiplas instancias de objetos, suas relações de dependência e de que forma (sentido e ordem) as

mensagens são trocadas entre eles. Ele representa uma visão alternativa e complementar ao diagrama de sequencias.

Um exemplo de Diagrama de Colaboração contendo os objetos (retângulo), relação de dependência entre eles, as mensagens (setas indicando o sentido da comunicação e rótulos indicando a ordem de comunicação)



5) O que diferencia atividades de manutenção do software de atividades de evolução? Que atividades são normalmente executadas pelos engenheiros de software para prolongar o ciclo de vida do software? Indique 3 (três) fatores que podem afetar o esforço destas atividades, explicando de que forma eles podem afetar as atividades de manutenção e evolução do software. Qual destas atividades todo software está sujeito a sofrer? (valor: 2,0 pontos; máximo: 15 linhas).

Qualquer trabalho realizado para alterar o sistema depois que ele já se encontra em operação é considerado manutenção. Entretanto, atividades de evolução intencionam realizar a modernização e adaptação do software a novos requisitos e condições de ambiente.

As atividades executadas podem ser de:
Correções: Corrige um defeito – i.e. uma discrepância entre o comportamento requerido para um produto/aplicação e o comportamento observado
Melhorias: Implementam uma mudança para o sistema que modifica seu comportamento ou implementação. Melhorias podem ser: Troca de requisitos (Manutenção Perfectiva), Adiciona um novo requisito ao sistema (Manutenção Adaptativa) e Troca a implementação mas não o requisito (Manutenção Preventiva)

Fatores: Tipo de aplicação, Novidade do sistema, Rotatividade e disponibilidade do pessoal de manutenção, Duração da vida útil do sistema, Dependência de um ambiente que se modifica, Característica de hardware, Qualidade do projeto, Qualidade do código, Qualidade da documentação, Qualidade dos testes

Responsabilidades da Equipe: entender o sistema, localizar informação na documentação do sistema, manter a documentação do sistema atualizada, estender as

funções existentes para acomodar novos requisitos ou modificações nos requisitos, acrescentar novas funções para o sistema, encontrar a fonte de falhas ou problemas no sistema, localizar e corrigir faltas, responder questões sobre a forma como o sistema funciona, reestruturar o projeto e codificação dos componentes, reescrever o projeto e código dos componentes, apagar os projetos e códigos de componentes que não são mais úteis, gerenciar trocas para o sistema

O tipo de manutenção que normalmente tem recebido mais atenção das equipes é a perfectiva.