Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação

Disciplina de Arquitetura e Projeto de Sistemas

Gabarito da AD2 – 1° semestre de 2017

Nome:

Polo:

Matrícula:

Observações:

1. Prova com consulta.

LER ATENTAMENTE AS INSTRUÇÕES A SEGUIR:

1. As respostas devem ser enviadas em um **único arquivo em formato exclusivamente .PDF, não compactado**. Além disso, o conteúdo deste arquivo deve **seguir exatamente o *template* das respostas**, caso exista. Caso não atenda a estes pontos, a **AD não será corrigida**. ADs enviadas no MODO RASCUNHO também **não serão corrigidas**. ADs MANUSCRITAS ou ESCANEADAS também **não serão corrigidas**.
2. Como a avaliação à distância é individual, caso sejam constatadas semelhanças entre provas de alunos distintos, **será atribuída a nota ZERO** a TODAS as provas envolvidas. As soluções para as questões podem ser buscadas por grupos de alunos, mas a redação final de cada prova tem que ser distinta.
3. Além disso, às questões desta AD respondidas de maneira muito semelhantes às respostas oriundas dos gabaritos já publicados de ADs e APs de períodos anteriores, **será atribuída a nota ZERO**, incluindo também cópias diretas, indiretas (semelhanças/paráfrases) ou sem sentido de tópicos dos slides das aulas. A AD é uma atividade de pesquisa (trabalho da disciplina) e deve ser elaborada como tal, não se atendo somente ao conteúdo dos slides das aulas.
4. Apenas ADs **enviadas pela plataforma** e **dentro do prazo estabelecido** serão corrigidas.
5. Por fim, a pesquisa na Internet e em livros é estimulada, devendo ser referenciada na AD, mas as respostas devem ser construídas com as palavras do próprio aluno e atender diretamente ao que pede à questão, evitando respostas prolixas ou extensas. Às respostas copiadas ou semelhantes a soluções da Internet ou de livros, e/ou que não atendem (fora do escopo) ou excedem demasiadamente ao que pede a questão, **será atribuída a nota ZERO**.

Questão 1 [1 ponto]

Cite algum padrão GRASP que contribui para a pertinência das classes em um único domínio (recordar as aulas de Princípios de Projeto OO). Justifique.

Questão 2 [2 pontos]

Em um *sistema de ponto de venda*, deseja-se criar uma classe que representa o catálogo unificado dos produtos disponíveis em todas as lojas da rede. Para isso, esse catálogo deve ter um ponto global de acesso e ter uma única instância.

a) Qual padrão de projeto deve ser utilizado nessa situação? Justifique. **[1 ponto]**
b) Desenhe o diagrama de classes que represente a solução com a adoção desse padrão de projeto. **[1 ponto]**

Questão 3 [3 pontos]

Em relação à Reutilização de Software, responda:

a) Diante das fases envolvidas no processo de reutilização como um todo, detalhe como a fase de criação e a fase de utilização estão relacionadas. **[1 pontos]**
b) Uma vez que o repositório de componentes é um elemento chave para a reutilização, descreva três de suas funcionalidades que são essenciais para a relação discutida em (a). Dica: pesquisar os processos Gerência de Reutilização e Desenvolvimento para Reutilização nos guias de implementação do Modelo MPS para Software (http://www.softex.br/mpsbr/_guias/default.asp). **[2 pontos]**

Questão 4 [2 pontos]

Em relação a Componentes, responda:

a) Explique o conceito de componente de forma a incluir pelo menos cinco de suas propriedades fundamentais. Além disso, faça um desenho que exemplifique estas propriedades, seja exemplificando-as por meio de um exemplo de componente, seja montando um modelo ou analogia para descrevê-las. **[1 pontos]**
b) No que se refere ao escopo do projeto de um componente, o que é mais vantajoso ou interessante: um componente com muitas funcionalidades, ou com poucas funcionalidades? Argumente a sua resposta. **[1 ponto]**

Questão 5 [1 ponto]

O que são interfaces? Quais os seus tipos? Explique-os e exemplifique-os considerando um componente de cálculo de conversão de moedas.

<u>Questão 6</u> [1 ponto]

O que torna arquitetura de software um tópico relevante hoje em dia? Procure identificar os benefícios de se considerar a arquitetura de software durante o desenvolvimento de um sistema.

Gabarito

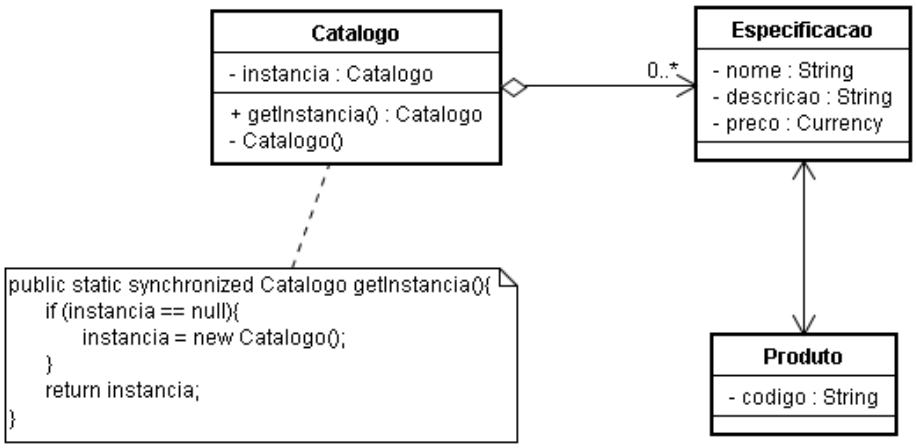
Questão 1

Padrão *Pure Fabrication* (Invenção Pura). Este padrão indica a criação de classes artificiais sempre que todas as opções ferem o princípio de alta coesão. Ferir o princípio de alta coesão pode significar que a classe está pertencendo a mais de um domínio ao mesmo tempo.

Questão 2:

a) O padrão de projeto *Singleton*. Sempre que é necessário ter somente uma única instância para uma classe e permitir que essa instância tenha um ponto global de acesso no sistema, o padrão *Singleton* é o indicado.

b)



Questão 3:

a) Estão relacionadas por meio da fase de gerência, uma vez que esta é responsável por coletar, avaliar, descrever e organizar artefatos reutilizáveis para garantir sua disponibilização aos processos de criação e utilização. A fase de gerência normalmente envolve a coleta de métricas e a administração de um repositório para manter e evoluir os ativos reutilizáveis, i.e., (i) permitir a identificação de novas demandas de criação de ativos, seja a partir do zero, por composição ou reutilização de ativos previamente armazenados, ou ainda requisições de evolução ou adaptação diante do uso daqueles existentes; e (ii) dar suporte ao ciclo de vida de um ativo reutilizável, o que envolve aquisição, aceitação, classificação, catalogação e certificação.

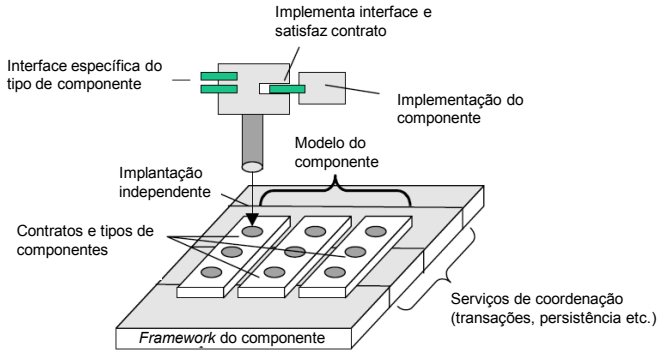
b) Primeiro, permitir a busca e recuperação de ativos reutilizáveis, i.e., explorar a documentação fornecida no ato da catalogação/publicação dos ativos para facilitar a sua localização pelos membros dos projetos de uma organização. Segundo, manter um mapa de reutilização, i.e., um mecanismo que permite que os produtores de ativos conheçam os seus consumidores e as suas necessidades, criando um canal de contato entre as partes; este canal viabiliza a notificação de atualizações, demandas

de modificações, correção de defeitos e novas funcionalidades, favorecendo o amadurecimento do próprio ativo. Por fim, gerenciar as versões dos ativos, i.e., permitir que toda liberação de versões de ativos reutilizáveis faça parte do repositório, de maneira a aplicar a esta nova liberação os critérios estabelecidos para a certificação de ativos mencionada em (a).

Questão 4:

a) Componente é uma unidade de software independente, que encapsula dentro de si seu projeto e implementação, oferecendo serviços por meio de interfaces bem definidas. A seguir, são explicadas as cinco propriedades fundamentais destacadas na ordem em que foram sublinhadas:
   i. **Identificação**: componentes devem ser facilmente identificados, ou seja, devem estar armazenados em um único lugar, ao invés de espalhados e misturados com outros artefatos de software ou documentação;
   ii. **Autocontido**: característica dos componentes de poderem ser reutilizáveis sem a necessidade de incluir ou depender de outros componentes. Caso haja alguma dependência, todo o conjunto deve ser abstraído como um componente reutilizável;
   iii. **Documentação**: A existência de documentação é indispensável para a reutilização. O tipo de componente e a sua complexidade indicarão a conveniência do tipo de documentação;
   iv. **Funcionalidade**: componentes têm uma funcionalidade clara e específica que realizam e/ou descrevem. Assim, componentes podem realizar funções ou podem ser simplesmente descrições de funcionalidades (e.g., artefatos do ciclo de vida);
   v. **Interface**: componentes devem possuir interfaces claras, que indicam como estes podem ser reutilizados e conectados a outros componentes, devendo-se ocultar os detalhes que não são necessários para a reutilização per si (encapsulamento).

Um desenho que exemplifique estas propriedades (i.e., elementos) do contexto de concepção e desenvolvimento de um componente pode ser observado abaixo[1]:



[1] **Fonte**: BACHMANN, F., BASS, L., BUHMAN, C., COMELLA-DORDA, S., LONG, F., ROBERT, J., SEACORD, R., WALLNAU, K., 2000, Technical Concepts of Component-Based Software Engineering, 2. ed., Technical Report CMU/SEI-2000-TR-008, Software Engineering Institute.

b) Dependendo do contexto do componente e/ou do projeto e da experiência do desenvolvedor e/ou organização, por exemplo, diversas vantagens (e respectivas desvantagens justificadas sobre cada uma dessas vantagens) para cada uma das escolhas mencionadas poderiam ser apontadas. No entanto, analisar o escopo de um componente requer uma análise de contexto e experiência. Um componente de conversão de moeda para um sistema web de uma companhia aérea é bem delimitado em seu escopo, isto é, possui poucas funcionalidades (ou uma, no caso) e é pequeno em termos da medida de uma métrica como LoC (linhas de código). Por outro lado, um framework para desenvolvimento web como Struts ou JSF, com diversas funcionalidades e grande em termos da medida da métrica LoC, no contexto de um sistema nacional para integrar subsistemas de saúde, educação, receita, eleitoral etc. com milhões de usuários e com vários requisitos não funcionais como disponibilidade e confiabilidade, também é bem delimitado em seu escopo e, diante de outros componentes deste sistema (onde um subsistema pode ser um componente), pode parecer oferecer poucas funcionalidades e até ser pequeno em termos de própria medida mencionada, no contexto em que está inserido. Adicionalmente, um framework destes, no contexto do sistema web de uma companhia aérea, pode parecer grande em termos da medida em questão, e com várias funcionalidades. Assim, o vantajoso, ou interessante, é que o componente atenda ao conceito mostrado em seu conceito em (a), apresentando um escopo bem definido em seu projeto e construção.

Questão 5:

Interfaces consistem em um conjunto de assinaturas de operações que podem ser invocadas por um cliente (desde que cada operação tenha sua semântica especificada) e visam determinar como o componente pode ser reutilizado e interconectado com outros componentes.

As interfaces podem ser de negócio (providas ou requeridas) ou de configuração. As interfaces providas (*provided interfaces*) servem para fornecer serviços de um determinado componente para outros componentes de necessitam desse serviço. As interfaces requeridas (*required interfaces*) representam os serviços que um componente necessita para efetuar sua função (cumprir seu contrato) e são provenientes de outros componentes. Por fim, as interfaces de configuração (*configuration interfaces*) visam possibilitar a variabilidade do componente segundo alguns parâmetros pré-estabelecidos, que devem ser ajustados para que a funcionalidade desejada sobre o componente seja alcançada.

Como exemplo, um componente de cálculo de conversão de moedas apresentaria: como interface provida, uma operação que resultasse no valor calculado após a conversão; como interface requerida, operações que fornecem uma tabela de conversão de valores padrão; e como interface de configuração, os diferentes tipos de moedas suportados pelo componente.

Questão 6:

Podemos citar uma série de justificativas sobre a relevância da arquitetura de software no cenário atual da Engenharia de Software. Uma das principais está no fato de que produtos de software são entidades que se encontram em constante estado de

mudança, sobretudo no contexto de desenvolvimento atual, que contempla novas formas de interação como desenvolvimento distribuído de software. Dessa forma, com a necessidade de evolução, os produtos de software se tornam predispostos a defeitos, atrasos na entrega e custos acima do esperado. Ou seja, seu escopo, complexidade e manutenção são cada vez mais significativos e exigem que profissionais da área se comuniquem por meio de componentes de software, uma vez que percebemos que o sucesso de um sistema computacional depende muito mais das características do software produzido do que do hardware sobre o qual irá rodar. Somando-se a isso, técnicas de abstração como decomposição modular, linguagens de programação de alto nível e tipos de dados abstratos já não são mais suficientes para lidar com os requisitos envolvidos nos domínios de aplicação atuais. Por outro lado, a arquitetura de software visa incorporar a disciplina de reutilização e agregar suas vantagens, evitando retrabalho ou trabalho desnecessário, além de permitir aos engenheiros de software tomarem decisões sobre alternativas de projeto. Enfim, o foco recente em arquitetura de software se deve, sobretudo, à economia e à reutilização.

Dentre os benefícios de se considerar a arquitetura de software durante o desenvolvimento de um sistema, estão: (i) favorecer a gerência da complexidade; (ii) facilitar a comunicação entre os *stakeholders* envolvidos no desenvolvimento de software (desenvolvedores, clientes, gerentes, etc.); (iii) criar possibilidades de reutilização; (iv) viabilizar a evolução de sistemas; (v) reconhecer estruturas comuns entre sistemas; (vi) manter o controle da produção de software, já que os detalhes relativos a esforços e desenvolvimento e impacto de mudanças tornam-se mais claros; e (vii) permitir novas oportunidades para análise (ex. consistência, atributos de qualidade, atendimento a estilos arquiteturais).