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

Curso de Tecnologia em Sistemas de Computação
Disciplina de Arquitetura e Projeto de Sistemas II
Gabarito – AD2 2° semestre de 2009.

Nome –

Observações importantes:

1- Prova com consulta.

2- ADs enviadas pelo correio devem ser postadas cinco dias antes da data final de entrega estabelecida no calendário de entrega de ADs.

3- 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. ALÉM DISSO, às questões desta AD respondidas de maneira muito semelhantes às respostas oriundas dos gabaritos já publicados de ADs de períodos anteriores, será atribuída a nota ZERO, incluindo também cópias diretas e sem sentido de tópicos dos slides das aulas.

Questão 1 [4 pontos]

Em relação à Reutilização de Software, responda:

a) [1 ponto] Defina a Reutilização de Software e explique quais foram as motivações que alavancaram o surgimento e o desenvolvimento desta disciplina.
b) [1 ponto] Discuta cada um dos benefícios apregoados pela Reutilização de Software sob a perspectiva das dificuldades inerentes à esta disciplina, considerando aspectos técnicos e não técnicos (se possível, considerá-los na forma de aspectos sociotécnicos).
c) [1 ponto] Da área de Qualidade de Software, sabe-se que o modelo MPS, no contexto do programa de Melhoria de Processo do Software Brasileiro (MPS.BR), se baseia nos conceitos de maturidade e capacidade de processo para a avaliação e melhoria da qualidade e produtividade de produtos de software e serviços

correlatos. Uma vez que uma das metas da Reutilização é agregar qualidade a produtos de software, pesquise quais são os dois processos do modelo MPS que estão diretamente relacionados com esta disciplina e discorra sobre cada um deles (DICA: verificar o Guia Geral, que contém a descrição do modelo MPS[1]).

d) [1 ponto] É conhecido na literatura técnica e em relatos de experiência da indústria que diversos progressos em Reutilização de Software foram conseguidos nas últimas duas décadas, tais como sistemas de bibliotecas, técnicas de classificação, criação e distribuição de componentes e ambientes de apoio ao desenvolvimento *para* e *com* Reutilização. Diante disso, selecione um trabalho de pesquisa que trata de aspectos não técnicos ***ou*** um tema de pesquisa atual na área de Reutilização de Software e apresente o seu conceito e o seu estado atual, bem como desenvolva textualmente seus impactos positivos e/ou negativos para a área.

Resposta:

a) Reutilização de Software é o processo de incorporar em um novo produto artefatos de software preexistentes ou, de uma forma geral, o conhecimento sobre o desenvolvimento destes artefatos. Para entender as motivações que levaram ao surgimento e desenvolvimento desta disciplina, basta recordar que, em situações cotidianas, quando um desenvolvedor lê um código implementado por outro desenvolvedor, ele pode notar algum estilo diferente que é melhor do que aquele que costumava utilizar; da mesma forma, quando se constrói um diagrama ou modelo de *design*, pode ser útil reutilizar alguma notação ou padrão criado por alguma pessoa, para que a equipe do projeto (bem como os futuros mantenedores e desenvolvedores) possa entender e colaborar entre si. Com o surgimento da denominação Engenharia de Software, em 1968, as primeiras idéias sobre a disciplina de Reutilização de Software apareceram, quando Doug McIlroy mostrou a sua motivação por software que funcionasse como "circuitos integrados" e que pudessem ser fabricados em massa, verificando os tipos de variabilidade e os outros tipos que poderiam ser padronizados – sua visão era baseada na indústria de componentes eletrônicos, os quais podiam ser selecionados em catálogos de diferentes fabricantes, apresentando propriedades configuráveis e diferentes níveis de confiabilidade. Apesar desse e de outros conceitos sobre Reutilização de Software aparecerem no final da década de 1960, pouca atenção por parte da indústria e das universidades foi concedida até 1980, quando a complexidade dos sistemas começou a se tornar visível e as empresas tiveram que buscar métodos mais eficientes de desenvolvimento de software para refletir as necessidades do mercado. Nesse momento, a Reutilização de Software era uma prática sem tanta padronização e deixava a cargo dos desenvolvedores toda a responsabilidade de fornecer e recuperar possíveis artefatos reutilizáveis na carteira de projetos da organização, além de fornecer pouco apoio e motivação para isso. Sem dúvida, isso era inadequado, pois as pessoas perdiam tempo em procurar e entender o funcionamento de cada artefato candidato (e suas partes) e, ainda assim, corriam o risco de introduzir novos defeitos no sistema. Por fim, o

[1] Maiores informações sobre o modelo MPS e os processos no Modelo de Referência MR-MPS estão disponíveis em http://www.softex.br/mpsbr/_guias/default.asp.

surgimento das tecnologias orientadas a objetos constituiu um marco essencial na Reutilização de Software e, atualmente, esta disciplina conta com um acervo variado e amplo de técnicas, métodos, ferramentas e processos relacionados, além de um grande grupo de adeptos e simpatizantes.

b) Apesar da lista de benefícios esperados com a adoção da Reutilização de Software, existe, em contrapartida, uma série de barreiras ou aspectos sensíveis intimamente relacionados. Dentre os benefícios apregoados, considerando os aspectos técnicos, estão: (i) melhores índices de produtividade; (ii) produtos de melhor qualidade, mais confiáveis, consistentes e padronizados; (iii) redução dos custos e tempo envolvidos no desenvolvimento de software; e (iv) maior flexibilidade na estrutura do software produzido, facilitando a sua manutenção e evolução. Por outro lado, existem algumas dificuldades para se alcançar estes benefícios: (i) a identificação, a recuperação e a modificação de artefatos reutilizáveis constituem um tripé fundamental para se vislumbrar uma redução de custos e tempo e/ou ganho de produtividade; (ii) a compreensão dos artefatos recuperados implica diretamente na facilidade (ou não) de manutenção e evolução de produtos de software (manutenibilidade); (iii) a garantia da qualidade dos artefatos reutilizáveis impacta o projeto e desenvolvimento de produtos de software de qualidade; e (iv) a composição de aplicações a partir de componentes afeta propriedades do produto de software no que tange a confiabilidade, a consistência e a padronozação. Além disso, outros problemas são provenientes dos chamados aspectos não técnicos da disciplina de Reutilização de Software, que devem ser igualmente considerados por esta disciplina. Por exemplo, o uso de recompensas para cada artefato reutilizado pode ajudar na implantação de um programa de reutilização (i.e., necessidade da criação de incentivos à reutilização). Isso, de certa forma, cria elementos para uma abordagem sociotécnica, na qual aspectos técnicos e não técnicos corroboram para co-modifcarem um ao outro simultaneamente e rumo a um contexto que os reflita claramente: conforme o exemplo mencionado, o programa de reutilização implantado deverá contemplar estratégias muitas vezes intangíveis para motivar os *stakeholders* envolvidos (desenvolvedores, analistas, testadores etc.), bem como considerar questões financeiras (investimento) e temporais (retorno esperado). Dessa forma, por mais que haja uma *baseline* para controle ou um arcabouço de processo passível de instanciação, o processo de implantação de um programa de reutilização, e a sua execução *a posteriori*, afeta e é afetado pelo conjunto de *stakeholders* de diferentes formas em função dos elementos de contexto (porte da organização, política, país, economia, sociedade etc.). Outro ponto importante é o fato dos desenvolvedores terem boa vontade para construir artefatos reutilizáveis e, ao mesmo tempo, possuírem certa dificuldade em aceitar e confiar no trabalho realizado por "terceiros" (*síndrome do não-inventado-aqui*). Nesse ponto, emergem as chamadas barreiras psicológicas, além de outras, tais como barreiras legais e econômicas.

c) Os dois processos do modelo MPS que estão diretamente relacionados com a disciplina de Reutilização de Software são o Processo *Gerência de Reutilização*

(GRU) e o Processo *Desenvolvimento para Reutilização* (DRU). O GRU tem o propósito de gerenciar o ciclo de vida dos ativos reutilizáveis (qualquer artefato relacionado a software que esteja preparado, i.e., empacotado de maneira própria a ser reutilizado pelos processos da organização). Assim, o GRU visa definir procedimentos tanto administrativos quanto técnicos para a utilização de ativos reutilizáveis em uma organização, estabelecendo e controlando uma biblioteca para o armazenamento e recuperação destes ativos. O DRU, por sua vez, tem o propósito de identificar oportunidades de reutilização sistemática de ativos na organização e, se possível, estabelecer um programa de reutilização para desenvolver ativos a partir de engenharia de domínios de aplicação. Dessa forma, o DRU visa aplicar técnicas de engenharia de domínio para definir o escopo, especificar a estrutura e construir ativos reutilizáveis para uma classe de sistemas, subsistemas e aplicações. Por serem produzidos a partir da engenharia de domínio, esses ativos são denominados ativos de domínio.

d) Em Reutilização de Software, trabalhos recentes tratam de aspectos não técnicos, tais como aspectos gerenciais, econômicos, culturais e legais, e, dentre alguns temas de pesquisa atuais, destacam-se engenharia de domínio/linha de produtos, reutilização de processos e desenvolvimento baseado em componentes. Deve-se recordar que, para que a reutilização alcance os níveis desejados de produtividade e qualidade na construção de software, é necessário que ela seja feita de maneira sistemática, considerando todas as fases do desenvolvimento, desde a análise até a implementação. Tal necessidade dá lugar às técnicas de reutilização, tais como Linha de Produtos de Software (LPS), *Frameworks* e Desenvolvimento Baseado em Componentes (DBC). O paradigma de LPS tem como proposta a construção sistemática de software baseada em uma família de produtos, guiando as organizações tanto na construção de novas aplicações, a partir de artefatos reutilizáveis (desenvolvimento *com* reutilização), quanto na construção desses artefatos (desenvolvimento *para* reutilização). Além disso, permite que as organizações explorem as características comuns e variáveis dos seus produtos de software como forma de alcançar economia em sua produção. O conceito de famílias de produtos foi introduzido por Parnas na década de 1970 e, inicialmente, levava em consideração apenas a variabilidade de requisitos não funcionais entre os produtos. O conceito de Engenharia de Domínio, que, posteriormente, deu origem ao termo LPS, só foi completamente introduzido no início da década de 1990, tendo como suas primeiras contribuições a descrição do método Draco (de autoria de Neighbors, publicado em 1989) e do método FODA (*Feature-Oriented Domain Analysis*, de autoria de Kang, publicado em 1990). Por sua vez, uma linha de produtos de software é um conjunto de sistemas que usam software intensivamente, compartilhando um conjunto de características comuns e gerenciadas, que satisfazem as necessidades de um segmento em particular de mercado ou missão, e que são desenvolvidos a partir de um conjunto comum de ativos principais e de uma forma preestabelecida. Os ativos, também chamados de núcleo de artefatos (*core assets*), são a essência da linha de produtos e correspondem a um conjunto de elementos customizáveis, utilizados na construção das aplicações produzidas. O núcleo inclui artefatos como, por

exemplo: modelo do domínio, arquitetura e componentes reutilizáveis que serão integrados à arquitetura para gerar os produtos. Dentre esses elementos, a arquitetura é o elemento chave. Além disso, uma LPS atua sobre um segmento particular ou missão, também chamado de domínio, que se refere à área de conhecimento ou especialização em que ela atua. O termo domínio também é utilizado para referenciar uma coleção de aplicações existentes ou futuras que compartilham características comuns. Dentre os benefícios de LPS, destaca-se o seu alto nível de reutilização, reduzindo o *time-to-market* e requerendo um menor número de pessoas para a produção de software.[2]

Questão 2 [3 pontos]

O padrão *Strategy* visa definir uma família de algoritmos de forma encapsulada, impactando positivamente a atividade de manutenção de software.

a) [1 ponto] Descreva um exemplo, **diferente daquele visto em aula**, de uma situação onde esse padrão é útil.
b) [1 ponto] Quais são os benefícios alcançados com o uso desse padrão?
c) [1 ponto] Desenhe um modelo de classes exibindo como o padrão pode ser utilizado na situação descrita.

Resposta:

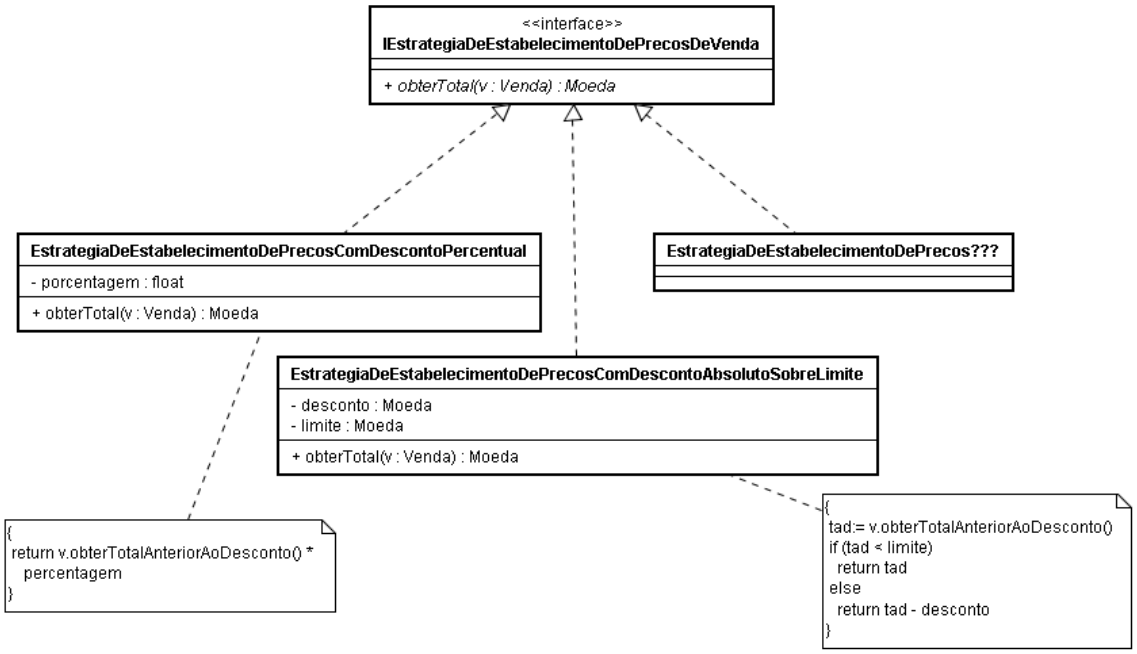
a) A estratégia de estabelecimento de preços (que também pode ser chamada de regra, política ou algoritmo) para uma venda pode variar. Durante um período, ela pode ser um desconto de 10% em todas as vendas, posteriormente, pode ser um desconto de R$ 10,00 se o total da venda for maior que R$ 200,00, além de muitas outras variações. Dessa forma, como o comportamento do estabelecimento de preços varia de acordo com a estratégia, pode-se criar várias classes `EstrategiaDeEstabelecimentoDePrecosDeVenda`, cada uma com um método polimórfico `obterTotal()`. Cada método deste recebe o objeto `Venda` como parâmetro, de modo que o objeto referente à estratégia de estabelecimento de preços possa encontrar o preço anterior ao desconto a partir de `Venda` e depois aplicar a regra de desconto. Assim, a implementação de cada método `obterTotal()` será diferente. Como exemplo, pode-se citar `EstrategiaDeEstabelecimentoDePrecosComDescontoPercentual`, que visa dar um percentual de desconto.

b) Inevitavelmente, um dos benefícios mais importantes obtidos com o uso deste padrão é favorecer à manutenibilidade de um produto de software (mais especificamente a duas subcaracterísticas de manutenibilidade, apresentadas na norma ISO/IEC 9126, *adaptabilidade* e *modificabilidade*). Além disso, este padrão visa à reutilização, à organização e à flexibilidade dos algoritmos

[2] Fonte: Fernandes, P. **UbiFEX: Uma Abordagem para Modelagem de Características de Linha de Produtos de Software Sensíveis ao Contexto**. 2009. Dissertação (Engenharia de Sistemas e Computação), COPPE/UFRJ. Disponível em http://reuse.cos.ufrj.br/files/publicacoes/mestrado/Mes_PaulaCibele.pdf.

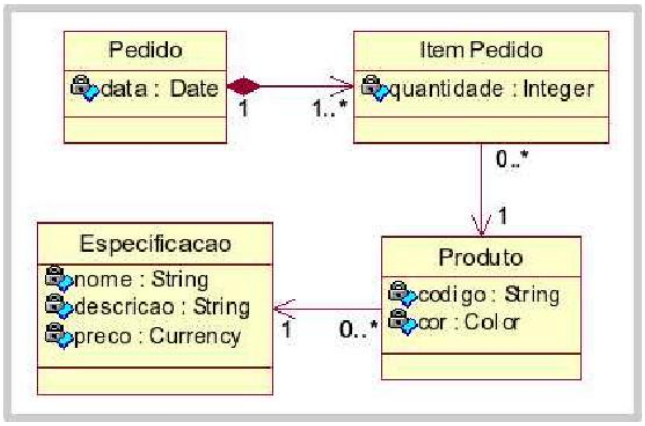
implementados em um produto de software, permitindo projetar software para que facilite a manutenção e a evolução de regras de negócio em tempo de execução.

c) Modelo referente à descrição em (a):



Questão 3 [1 ponto]

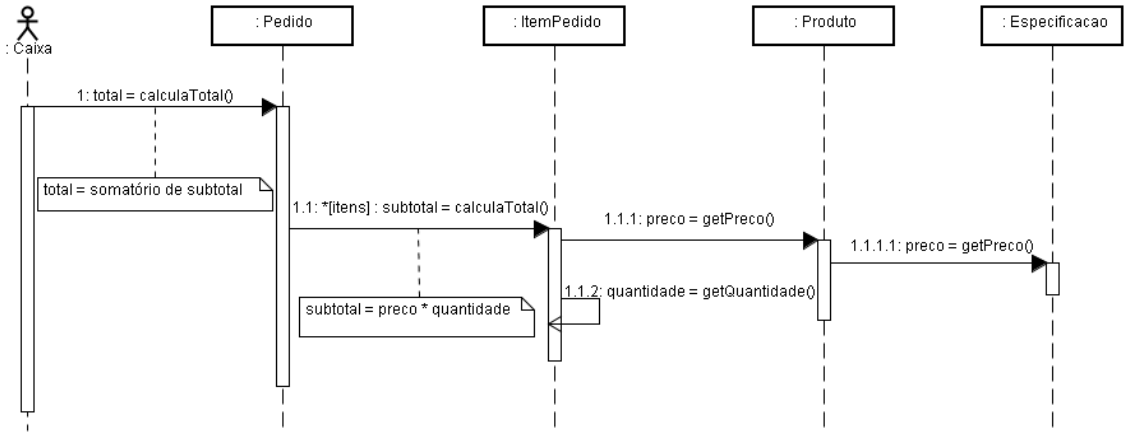
Desenvolva o modelo abaixo (através de modelo de seqüência) para inserir a responsabilidade de cálculo do total do pedido segundo o padrão *Expert*:



Resposta:

Conforme o padrão *Expert*, em um sistema de PDV, o responsável pelo cálculo do total de um pedido deveria ser a própria classe `Pedido`, uma vez que deve-se atribuir a responsabilidade ao especialista, ou seja, à classe que tem a informação necessária para

satisfazer a responsabilidade (no caso, o cálculo do total do pedido). Isso pode ser visualizado pelo diagrama de seqüência abaixo:



Questão 4 [2 pontos]

No contexto do desenvolvimento de software tradicional, sabe-se que a utilização do conceito de interfaces ajuda na substituição de componentes.

a) [1 ponto] Primeiramente, defina o que são interfaces e quais são os seus tipos. Explique-os e exemplifique-os considerando um componente de cálculo de conversão de moedas.
b) [1 ponto] A partir do fato de que existem alguns problemas relacionados com a utilização do conceito de interfaces na substituição de componentes, como poderia ser feito *refactoring* de um sistema que precisa ter partes substituídas, mas que não foi projetado para isso (não usa componentes)? E como poderia ser feita a modificação de um serviço declarado público na interface?

Resposta:

a) 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 que necessitam destes serviços. As interfaces requeridas (*required interfaces*) representam os serviços que um componente necessita para efetuar a sua função (cumprir o 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.

b) Considerando a primeira situação apresentada, dado que as partes desse sistema não se comunicam por meio de interfaces (não foram utilizados componentes em seu projeto), o *refactoring* iniciaria pela identificação das partes principais do sistema e introdução de interfaces entre essas partes. A partir desse momento, cada uma dessas partes do sistema poderia ser aprimorada individualmente, testada e substituída sem que as demais partes sejam afetadas. Em uma situação ideal, essas partes poderiam se tornar componentes, facilitando a manutenção futura. Considerando a segunda situação apresentada, se um serviço é declarado como público na interface de uma classe, outras classes podem fazer uso do mesmo. Sendo assim, é necessário, inicialmente, comunicar aos possíveis usuários do serviço que uma modificação precisa ser feita. Nesse período, tanto a versão antiga do serviço quanto a nova devem estar disponíveis, para possibilitar uma migração gradativa dos clientes. Além disso, durante o período de migração, a operação antiga referente ao serviço deve chamar a sua nova operação, para não correr o risco de haver código duplicado, o que poderia gerar retrabalho de manutenção. Somente quando o período de migração se encerrar é que a versão antiga do serviço pode ser removida, restando somente a nova versão do mesmo.