Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Engenharia de Software
AD2 1° semestre de 2019.
Gabarito

Atenção: Como a AD é individual, caso seja constatado que provas de alunos distintos são cópias uma das outras, independentemente de qualquer motivo, a todas será atribuída a nota ZERO. As soluções para as questões podem sim, ser buscadas por grupos de alunos, mas a redação final das respostas para as questões da prova tem que ser individual!
Os tópicos tratados como pesquisa na AD2 poderão ser questionados na AP2 ou AP3!

1. O que diferencia linguagens de programação interpretadas das linguagens de programação compiladas? De dois exemplos de cada tipo e indique para que categoria de software cada uma delas poderia ser mais adequada e por quê. (Valor: 1,5 ponto)

Programas escritos em linguagens de programação compiladas passam pelas seguintes etapas:

(i) desenvolvimento do código-fonte;(ii) compilação;

(iii) geração do código objeto;

(iv) linquedição (fusão) das partes de código-objeto; e

(v) geração do programa final em código de máquina.

Geralmente são aplicados em situações onde é necessário alto desempenho, aceitando-se a execução em apenas uma plataforma.

Programas escritos em linguagens de programação interpretadas passam pelas seguintes etapas:

(i) desenvolvimento do código-fonte;

(ii) compilação;

(iii) geração do código objeto em uma linguagem intermediária; e

(iv) interpretação da linguagem intermediária no computador destino.

Geralmente são aplicados em situações onde o programa deve rodar em diferentes plataformas e máquinas, cada qual com sua implementação do interpretador.

2. Quais são os componentes de um diagrama de atividades da UML? Mostre um diagrama de atividades e explique como estes componentes se relacionam e o quê cada um representa no diagrama. (Valor 1,5 ponto)

O Diagrama de Atividade é um dos diagramas comportamentais da UML, ou seja, representa aspectos dinâmicos de um sistema de software. Esse diagrama permite a representação do fluxo de mensagens de

uma atividade para a outra e se destaca por permitir a representação de diferentes partes de um sistema de software ou até mesmo representar questões relacionadas ao domínio de negócio. Por exemplo, é possível utilizar o diagrama de atividade para:

1- Descrever casos de uso;

2- Modelar processos de negócios;

3- Especificar os algoritmos a serem implementados no sistema;

4- Especificar regras de negócio ou cálculos complexos que o sistema deve implementar;

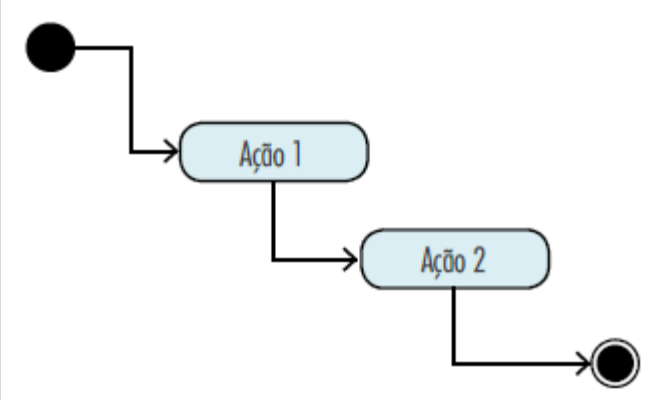
As figuras abaixo apresentam diferentes diagramas de atividades e seus elementos.

- Um círculo preenchido é denominado Atividade Inicial. Em seguida, os elementos descritos como “Ação 1” e “Ação 2” representam Ações.

- O elemento Atividade Final, que indica o término da atividade que está sendo modelada, é representado por um círculo preenchido e circulado

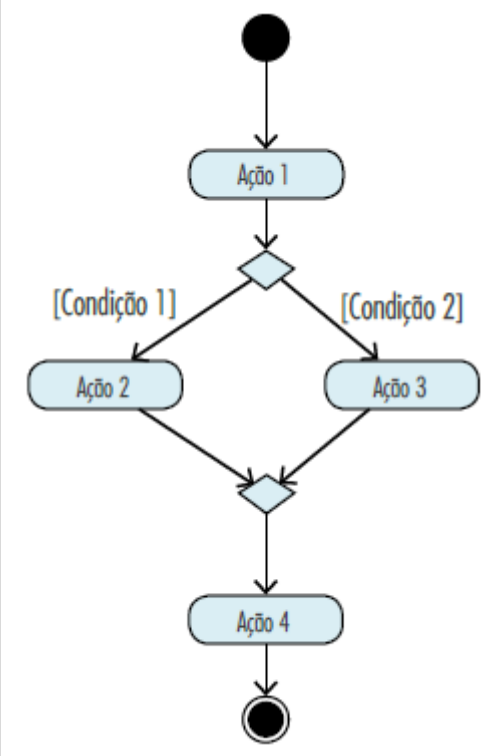
por um anel.

- As setas representam a possibilidade de transição de uma ação até a outra.

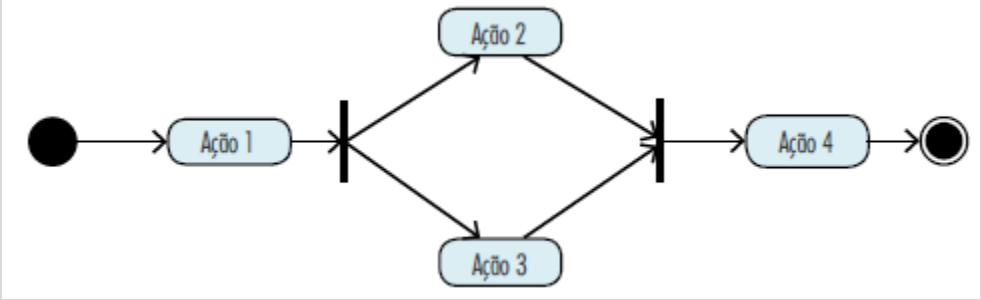


- Os elementos de condição de guarda, representados por um losango, modelam situações em que o fluxo de execução de uma atividade até outra atividade depende de uma decisão. Por exemplo, na Figura 9, a Ação 2 só é executada se a Condição 1 for verdadeira; a Ação 3 é executada apenas se a Condição 2 for verdadeira. É importante ressaltar que apenas uma dessas condições (Condição 1 ou Condição 2) pode ser verdadeira para cada fluxo de execução.

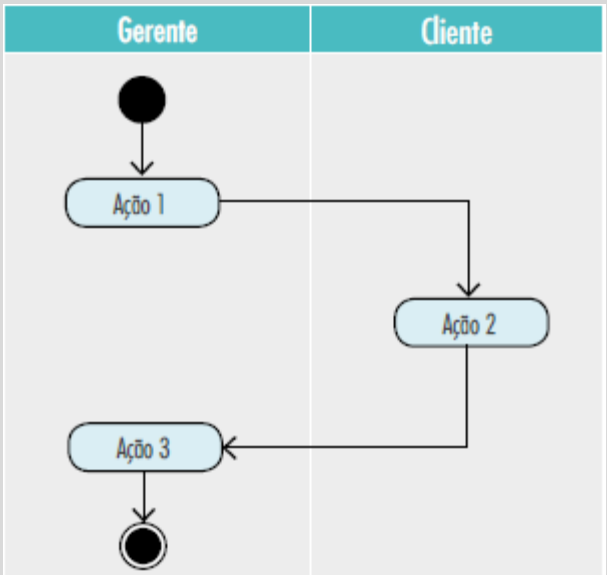
- O losango também é utilizado para realizar a junção do fluxo de execução (da Ação 2 ou Ação 3 para a Ação 4).



- Utiliza-se uma linha espessa para representar concorrência de ações, ou seja, duas ações sendo executadas ao mesmo tempo.



- Outro elemento disponibilizado pelo diagrama de atividade são as Raias. Esses elementos permitem representar a responsabilidade de execução de cada uma das Ações de um diagrama de atividade. A Figura 11 apresenta um exemplo no qual o Gerente é responsável pela execução da Ação 1 e Ação 3. Já o Cliente é responsável pela execução da Ação 2.



3. O que representa o conceito de coesão em software. Como podemos observar coesão no software? Como medir coesão? Apresente um exemplo de falta de coesão em software. (Valor: 1,5 ponto)

É uma característica de um elemento que compõe o software que diz que respeito à diversidade de tarefas executadas por este elemento. Por exemplo, um módulo altamente coeso realiza uma única tarefa, sem precisar executar rotinas de outros módulos.

Os níveis de coesão encontrados são:

- Baixo (Coincidental, Lógico, Temporal, Procedural); ou

- Alto (Comunicacional, Sequencial, Funcional).

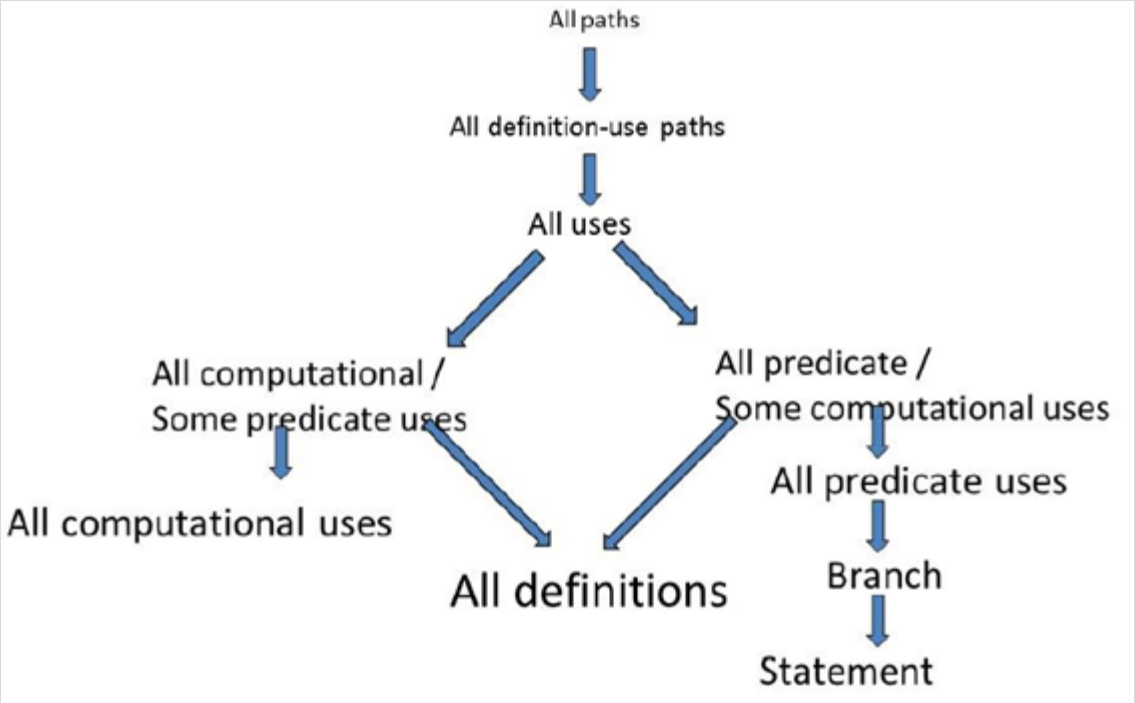
Em um software, é preferível a coesão Funcional, seguido pelos níveis Sequencial e Comunicacional.

A métrica LCOM (Perda de Coesão em Métodos) pode ser utilizada para medir a coesão.

4. Explique o que é teste estrutural. Apresente três exemplos de critérios de teste que podem ser utilizados para projetar casos de teste para este tipo de teste. (valor: 1,5 ponto)

Teste Estrutural toma por base a estrutura interna do software para planejar o teste. Neste sentido, fluxo de controle ou de dados servem como critérios para estabelecer os casos de teste e roteiros de teste. Usualmente, a estrutura do software é visualizada através do grafo do programa ou grafo de fluxo de controle.

Diferentes critérios podem ser utilizados: testar todos os nós do grafo, todos os arcos, todos os caminhos e outras opções. Quanto mais forte o critério (maior cobertura), maior o esforço do teste (planejamento e execução).



5. Atividade de pesquisa I (não vale copiar e colar! Você deve pesquisar e explicar com suas palavras. Indique fontes alternativas que usar!): (valor 2,0 pontos)

   O que diz o Código de ética do Profissional de Informática da Sociedade Brasileira de Computação (http://www.sbc.org.br/institucional-3/codigo-de-etica)? Como ele se compara ao Código de ética e de Prática Profissional da Engenharia de Software, segundo as recomendações da ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices (https://www.computer.org/cms/Computer.org/professional-education/pdf/doc.pdf)? Quais a consequências em não seguirmos um código de ética em nossos projetos de engenharia de software?

Tanto o código de ética da SBC quanto o da ACM/IEEE-CS fornecem diretrizes para a conduta dos profissionais de informática. O código da SBC consiste em um conjunto de 12 artigos que descrevem os deveres dos profissionais de informática. Já o código da ACM/IEEE-CS consiste em um conjunto de 8 artigos. Uma diferença entre os códigos é que o ACM/IEEE-CS, em sua versão completa, possui um detalhamento dos 8 artigos propostos

6. Atividade de pesquisa II (não vale copiar e colar! Você deve pesquisar e respondem com suas palavras. Indique fontes alternativas que usar!): (valor 2,0 pontos)

   A percepção da qualidade do produto de software tem evoluído ao longo do tempo. Abordagens clássicas e ainda muito importantes, observam a qualidade externa do produto através, por exemplo, da presença de defeitos. Entretanto, a evolução dos processos de desenvolvimento tem trazido preocupações com a evolução dos produtos. Ciclos interativos e incrementais dependem não apenas da qualidade externa, mas, principalmente, da qualidade interna do produto. Neste sentido o conceito de Dívida Técnica (Technical Debt) tem sido atualmente tratado. Portanto, o que é Divida Técnica? Como ela pode ser observada e/ou medida no produto? Quais são seus efeitos na evolução do produto? Toda Dívida deve ser paga? Por quê?

O termo dívida técnico é uma metáfora utilizada no contexto de desenvolvimento de software referindo-se a tarefas atrasadas e artefatos imaturos que constituem uma “dívida” porque incorrem em custos adicionais no futuro, sob a forma de aumento de custo de mudança durante a evolução e a manutenção.

Por exemplo, imagine que uma funcionalidade do sistema esteja finalizada, mas o código não está aderente aos padrões de qualidade estabelecidos pela organização (nome de variáveis muito longos, número grande de comandos em cada método, etc.).

Nesse cenário, pode haver uma decisão em disponibilizar o software para os usuários, mesmo com os problemas no código fonte. É importante perceber que esses problemas não são percebidos de imediato pelos usuários. Entretanto, cria-se uma dívida (dívida técnica), pois esse código precisa ser refatorado futuramente para atender aos padrões de qualidade da organização.

A dívida técnica pode ser observada através de métricas que representam a qualidade interna do produto. Por exemplo, a complexidade ciclomática.

O acúmulo de muitas dívidas pode prejudicar a evolução do produto, pois apesar de não serem diretamente percebidas pelos usuários finais, as dívidas aumentam o esforço de manutenção.

É importante explicitar que defeito não é dívida técnica, pois defeito é percebido diretamente pelos usuários do software.

Avgeriou P, Kruchten P, Ozkaya I, et al. (2016) Managing Technical Debt in Software Engineering Edited by. Dagstuhl Reports 6:110–138. doi: 10.4230/DagRep.6.4.110.

Disponível em: http://drops.dagstuhl.de/opus/volltexte/2016/6693/pdf/dagrep_v006_i004_p110_s16162.pdf