Engenharia de Software

para concursos

Questões comentadas

www.handbookdeti.com.br

Prefácio

O desenvolvimento de software é uma atividade de crescente importância na sociedade contemporânea. A utilização de computadores nas mais diversas áreas do conhecimento humano tem gerado uma crescente demanda por soluções computadorizadas.

Visando melhorar a qualidade dos produtos de software e aumentar a produtividade no processo de desenvolvimento, surgiu a Engenharia de Software. A Engenharia de Software trata de aspectos relacionados ao estabelecimento de processos, métodos, técnicas, ferramentas e ambientes de suporte ao desenvolvimento de software.

Devido a crescente utilização da Egenharia de Software, muitos concursos na área de Tecnologia de Informação têm exigido esse assunto como forma de averiguar o conhecimento e a habilidade dos candidatos na aplicação de métodos, técnicas e ferramentas no processo de desenvolvimento de software. Portanto, este volume foi preparado pelo Grupo Handbook de TI como forma de você se preparar melhor para os assuntos relacionados a esta área.

Bons estudos,

Grupo Handbook de TI

Direitos Autorais

A série *Handbook de Questões de TI Comentadas para Concursos – Além do Gabarito* é uma produção independente e contamos com você para mantê-la sempre viva.

Grupo Handbook de TI

Canais de Comunicação

O Grupo Handbook de TI disponibiliza diversos canais de comunicação para os concurseiros de TI.

Loja Handbook de TI

Acesse a nossa loja virtual em `http://www.handbookdeti.com.br`

Serviço de Atendimento

Comunique-se diretamente conosco através do e-mail `faleconosco@handbookdeti.com.br`

Twitter do Handbook de TI

Acompanhe de perto promoções e lançamentos de produtos pelo nosso Twitter `http://twitter.com/handbookdeti`

1. **Assuntos relacionados:** *Processo Unificado*,
Banca: *CESGRANRIO*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Eng. de Software*
Ano: *2008*
Questão: *48*

Um princípio fundamental do Processo Unificado é

(a). ser centrado em arquitetura.

(b). empregar times auto-dirigidos e auto-organizados.

(c). o desenvolvimento em cascata.

(d). a programação em pares.

(e). a propriedade coletiva do código fonte.

Solução:

O Processo Unificado de desenvolvimento de software reúne três características que o faz ser único. Essas três características se relacionam entre si e são igualmente importantes. São elas:

- **Orientado a Diagrama de Casos de Uso:** um caso de uso, de acordo com a UML (Unified Modeling Language), é uma sequência de ações de um sistema que devolve ao usuário um resultado de valor. Um conjunto de casos de uso definido sob determinado contexto forma um Diagrama de Casos de Uso, que descreve uma funcionalidade do sistema sob esse contexto. Em outras palavras, um Diagrama de Casos de Uso define a funcionalidade de um sistema para cada tipo de usuário. Esse tipo de abordagem favorece o atendimento das necessidades de cada tipo de usuário que interage com o sistema, evitando, dessa forma, que o sistema possa ser desenvolvido a ponto de apresentar funcionalidades desnecessárias;
- **Centrado na arquitetura do sistema:** arquitetura de sistema é uma visão do projeto como um todo, destacando suas características mais importantes de forma abrangente e sem detalhes específicos. Esse tipo de abordagem auxilia o arquiteto a se concentrar nas metas corretas, como inteligibilidade, poder de recuperação para mudanças futuras e reutilização. Ou seja, a arquitetura de um sistema deve ser projetada a ponto de permitir que o sistema evolua, não apenas durante o início do seu desenvolvimento, mas também ao longo das iterações futuras;
- **Metodologia iterativa e incremental:** uma iteração pode ser entendida como um miniprojeto, que resulta em uma nova versão do sistema. Justamente devido ao fato de a cada interação ser gerada uma nova versão do sistema que essa metodologia também se apresenta como incremental. Perceba que estas duas características, iterativa e incremental, nem sempre aparecem juntas. Há metodologias de desenvolvimento de software que são iterativos, mas não incrementais.

O Processo Unificado organiza suas iterações em quatro fases principais:

1. **Concepção:** o objetivo desta fase é levantar, de forma genérica e pouco precisa, o escopo do projeto. Não deve existir aqui a pretensão de especificar de forma detalhada requisitos, a ideia é ter uma visão inicial do problema, estimar de forma vaga esforço e prazos e determinar se o projeto é viável e merece uma análise mais profunda;

2. **Elaboração:** nesta fase, todos os requisitos (ou a grande maioria deles) são levantados em detalhes. Em uma primeira iteração, um ou dois requisitos, os de maior risco e valor arquitetural, são especificados e implementados. Eles servirão como base de avaliação junto aos usuários e desenvolvedores para o planejamento da próxima iteração. Em cada nova iteração dentro da fase de elaboração, pode haver um seminário de requisitos, onde requisitos antigos são melhores esclarecidos e novos são detalhados. Ao fim desta fase, 90% dos requisitos foram levantados em detalhes, o núcleo do sistema foi implementado com alta qualidade, os principais riscos foram tratados e pode-se então fazer estimativas mais realistas;
3. **Construção:** implementação iterativa dos elementos restantes de menor risco e menos complexos. Há também uma preparação para a implantação do sistema;
4. **Transição:** testes finais e implantação do sistema.

Agora que o básico sobre Processo Unificado já foi exposto, vamos a cada alternativa.

(A) CORRETA

Ao ler a explicação acima, fica evidente que "ser centrado em arquitetura" é um dos princípios mais evidentes do Processo Unificado. É justamente por isso que esta é a alternativa correta.

(B) ERRADA

Times auto-dirigidos e auto-organizados não é uma característica do Processo Unificado. Essa é sim um princípio de outra metodologia de desenvolvimento de sistemas, o Scrum. Portanto, esta alternativa está errada.

Scrum é um processo para construir software de forma incremental em ambientes complexos, onde os requisitos não são claros ou mudam com muita frequência. A metodologia é baseada em princípios semelhantes aos de XP (Extreme Programming). As principais são:

- equipes pequenas, multidisciplinares e auto-organizadas;
- requisitos pouco estáveis ou desconhecidos;
- iterações curtas (sprints) que seguem o ciclo PDCA e entregam incrementos de software prontos;
- reuniões diárias e curtas geralmente realizadas em pé;
- Scrum Master: membro que tem como função primária remover qualquer impedimento à habilidade de uma equipe de entregar o objetivo do sprint. O Scrum Master não é o líder da equipe (já que as equipes são auto-organizadas), mas atua como um mediador entre a equipe e qualquer influência desestabilizadora.

(C) ERRADA

O modelo de desenvolvimento em cascata é um modelo sequencial, no qual o desenvolvimento é visto como um fluir constante para frente (como uma cascata) através das fases de análise de requisitos, projeto, implementação, testes (validação), integração e manutenção de software. Como já descrito anteriormente, o Processo Unificado é uma metodologia iterativa e incremental, que praticamente nada tem a ver com desenvolvimento em cascata. Ou seja, esta alternativa não é a correta.

(D) e (E) ERRADAS

Estas duas alternativas trazem duas características associadas ao XP, e não ao Processo Unificado. Justamente por isso que elas não devem ser escolhidas pelo candidato.

A metodologia XP é destinada a grupos pequenos de desenvolvimento, e em projetos de duração de até 36 meses. Suas principais características são:

- metáforas: utilização de comparações e analogias para facilitar entendimento;
- design simples do sistema;
- testes automatizados: testes de unidade e de aceitação;
- refabricação: todo desenvolvedor tem o dever de melhorar um código que esteja funcionado porém está mal escrito;
- programação de dupla: todo código deve ser implementado em dupla;
- reuniões curtas e em pé;
- semana de 40 horas: ao se trabalhar a mais se obtém resultados no início, mas com o passar do tempo surge desgaste;
- integração contínua: eliminar erros graves de integração;
- releases curtos: release é um conjunto de funcionalidades bem definidas e que representam algum valor para o cliente. Um projeto XP pode ter uma ou mais releases no seu ciclo;
- CRC: linguagem para modelagem de classes do XP que utiliza os story cards (cartões escritos pelos usuários onde são descritas funcionalidades do sistema).

2. **Assuntos relacionados:** *Processo Unificado*, *UML*, *Modelo de Casos de Uso*,
Banca: *CESGRANRIO*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Eng. de Software*
Ano: *2008*
Questão: *49*

O modelo de casos de uso é um dos artefatos mais importantes previstos pelo Processo Unificado. Sobre o modelo de casos de uso, são feitas as afirmativas a seguir.

I Atores humanos são identificados com base no papel que desempenham do ponto de vista do sistema, e não necessariamente no cargo que ocupam na instituição em que o sistema rodará.

II A evolução dos casos de uso ao longo do ciclo de vida do projeto prevê que os mesmos ganhem em seu texto os detalhes específicos de implementação necessários à construção do software na tecnologia adotada.

III As combinações possíveis do fluxo principal com os fluxos alternativos de um caso de uso fornecem todos os cenários possíveis para o mesmo, os quais, por sua vez, podem ser utilizados como unidades de planejamento, implementação e testes.

IV É recomendável que cada caso de uso seja decomposto funcionalmente e passe a incluir casos de uso menores, sucessivamente, até a menor unidade implementável possível, atendendo ao princípio da decomposição funcional.

Estão corretas APENAS as afirmativas

(a). I e II

(b). I e III

(c). II e III

(d). II e IV

(e). III e IV

Solução:

O Processo Unificado de desenvolvimento de software tem três princípios básicos:

- **Centrado na arquitetura do sistema**;
- **Metodologia iterativa e incremental**;
- **Orientado a Diagrama de Casos de Uso**.

Diagrama de Caso de Uso é um dos principais diagramas da UML (Unified Modeling Language). Ele descreve, de forma gráfica e intuitiva, relacionamentos e dependências entre um grupo de casos de uso e os atores que interagem com o sistema. O ponto de vista assumido para essa descrição é sempre o dos atores envolvidos.

Um ator é uma entidade externa ao sistema. Ele é representado graficamente por meio de um boneco e um rótulo com o seu nome. Geralmente, é um ator que inicia um caso de uso. Alguns exemplos de atores são: usuários, outros sistemas que fazem interface com o sistema que está sendo modelado e eventos externos.

Um caso de uso é um conjunto de atividades do sistema que produz um resultado concreto e tangível. Ou seja, ele define uma grande função do sistema. Graficamente, casos de

uso são representados por elipses com seus respectivos rótulos.

s Enfim, esse tipo de diagrama descreve interações entre os atores e o sistema em si. Veja um exemplo na Figura 1

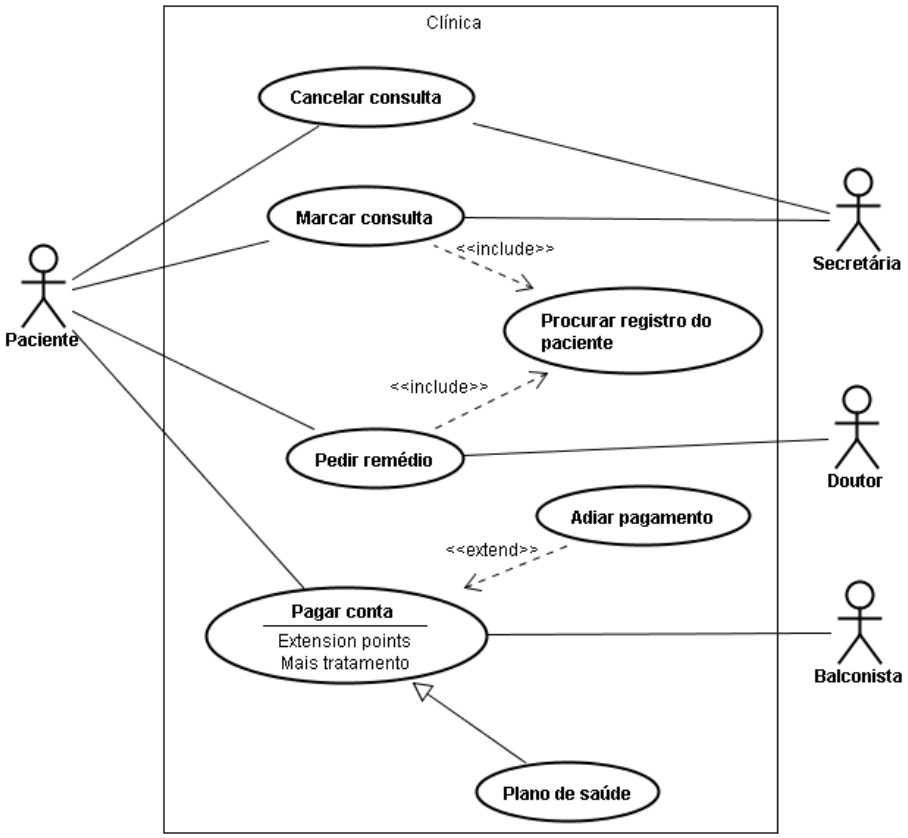


Figura 1: exemplo de Diagrama de Caso de Uso.

Os possíveis relacionamentos, que são representados no diagrama utilizando-se setas, são os seguintes:

- **entre um ator e um caso de uso**
  - **Relacionamento de Associação:** define do ponto de vista externo uma funcionalidade do sistema.
- **entre atores**
  - **Relacionamento de Generalização:** define que o ator que está na origem da seta tem seus próprios casos de uso. Já o ator que é apontado pela seta também tem os casos de uso do outro ator. Um exemplo típico desse tipo de relacionamento seria uma seta saindo do ator "Cliente On-line" e apontando para o ator "Cliente".
- **entre casos de uso**
  - **Relacionamento de Dependência-Inclusão:** um relacionamento de inclusão de um caso de uso A (origem da seta) para um caso de uso B (destino da seta) indica que B é essencial para o comportamento de A. Pode ser dito também que B é parte de A;
  - **Relacionamento de Dependência-Exclusão:** um relacionamento de extensão de um caso de uso B para um caso de uso A indica que o caso de uso B pode ser

acrescentado para descrever o comportamento de A (não é essencial). A extensão é inserida em um ponto de extensão do caso de uso A. Ponto de extensão em um caso de uso é uma indicação de que outros casos de uso poderão ser adicionados a ele. Quando o caso de uso for invocado, ele verificará se suas extensões devem ou não ser invocadas;

- **Relacionamento de Generalização ou Especialização (é-um):** um relacionamento entre um caso de uso genérico para um mais específico, que herda todas as características de seu pai. Quanto um caso de uso B é-um caso de uso A, A será uma generalização de B, ou seja, B será uma especialização de A.

A seguir, cada uma das afirmativas feitas no enunciado é analisada:

I verdadeira, pois os casos de uso são utilizados para representar o comportamento do sistema, e não a hierarquia da instituição;

II falsa, pois os casos de uso não devem incluir detalhes de implementação, e sim o comportamento esperado do sistema em um determinado cenário;

III verdadeira, pois o fluxo principal cobre o comportamento do sistema nos casos rotineiros e os fluxos alternativos cobrem os casos excepcionais. Dessa forma, todas as alternativas conhecidas de comportamento do sistema são cobertas, o que ajuda na fase de implementação e de testes;

IV falsa, pois o diagrama que oferece uma decomposição funcional do sistema é chamado Diagrama de Fluxo de Dados, e não o de Caso de Uso.

Portanto, a resposta correta é a alternativa B.

3. **Assuntos relacionados:** *Metodologia de Desenvolvimento de Software*,
Banca: *CESGRANRIO*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Eng. de Software*
Ano: *2008*
Questão: *50*

Estudos baseados na análise de diversos projetos de desenvolvimento de software sugerem que tais projetos têm maior chance de sucesso quando empregam metodologia e gerenciamento alinhados ao paradigma de desenvolvimento de novos produtos, em contraponto ao paradigma de produção industrial. Com base nessas observações, a maioria das metodologias modernas de desenvolvimento de software recomenda:

(a). concluir o trabalho de especificações dos requisitos do sistema, antes de iniciar as atividades de projeto e implementação.

(b). planejar detalhadamente no início do projeto todas as fases e atividades do mesmo, de forma que seja possível estimar com precisão o esforço necessário e os prazos de cada atividade.

(c). providenciar, desde o início do projeto, mecanismos para prevenir e bloquear solicitações de mudanças de forma a garantir que será entregue exatamente o que foi especificado.

(d). dividir o trabalho em iterações curtas, com prazos fixos, e não permitir que as mesmas avancem sobre os prazos, reduzindo o escopo da iteração, se necessário.

(e). não produzir documentação técnica para o sistema, tendo em vista que a mesma já nasce condenada a ficar desatualizada, investindo melhor o tempo em atividades de implementação e testes exaustivos.

Solução:

De certa forma, o ponto central que esta questão pretende avaliar é o fato da maioria das metodologias modernas de desenvolvimento de software abordarem projetos de desenvolvimento de novos produtos de forma cíclica (ou iterativa). Algumas dessas metodologias são: RUP (Rational Unified Process), XP (eXtreme Programming), Cleanroom, RAD (Rapid Application Development) e Espiral.

(A) ERRADA

Justamente pela abordagem das metodologias em questão ser cíclica, a fase de especificação dos requisitos do sistema não é concluída antes de se iniciar as atividades de projeto e implementação. Ou seja, mesmo após o início das atividades de projeto e implementação pode haver, e geralmente há, novas especificações de requisitos.

(B) ERRADA

Essa seria talvez uma boa abordagem, mas não é a que a maioria das metodologias modernas pregam. Elas pregam que no início do projeto, o planejamento e as estimativas do projeto como um todo sejam feitos de forma geral. Planejamentos e estimativas cada vez mais precisos são feitos a cada fase do projeto.

(C) ERRADA

Praticamente todas as metodologias de desenvolvimento de novos produtos, inclusive as modernas, pregam justamente o contrário dessa alternativa. Isso porque acredita-se que o quanto antes as eventuais mudanças forem consideradas em um projeto, menores serão os problemas com custo e cronograma desse projeto. As metodologias consideram também que pouco adianta ignorar as mudanças ao logo do projeto, pois, quando isso ocorre, mesmo que seja entregue exatamente o que foi especificado, o novo produto não será considerado conforme ou de qualidade.

(D) CORRETA

Como já mencionado no início deste comentário, a abordagem cíclica da maioria das metodologias modernas é o centro desta questão. Essa é a alternativa correta.

(E) ERRADA

Essa é, sem dúvida, a primeira alternativa que o candidato deve desconsiderar. Algumas metodologias até apontam para uma simplificação da documentação técnica, mas eliminar totalmente esse tipo de documentação não é uma alternativa.

4. **Assuntos relacionados:** *Processo Unificado*, *UML*, *Análise de Caso de Uso*,
Banca: *CESGRANRIO*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Eng. de Software*
Ano: *2008*
Questão: *51*

A atividade *analisar um caso de uso*, prevista no Processo Unificado, produz um artefato chamado *realização de análise de caso de uso*, que mostra como as classes de análise colaboram para que o caso de uso apresente o comportamento especificado. A esse respeito, assinale a afirmação correta.

(a). As classes de análise, neste artefato, devem conter referências a detalhes de implementação.

(b). Ao realizar a análise de um caso de uso, possíveis falhas e omissões no mesmo se tornam mais perceptíveis, tratando-se, portanto, de uma oportunidade para refinar o modelo de casos de uso.

(c). A interação entre as classes de análise é expressa primariamente através de diagramas de classes UML.

(d). Um outro produto desta atividade é o artefato chamado *realização de projeto de caso de uso*.

(e). O diagrama de robustez do sistema, que especifica os requisitos não funcionais de escalabilidade e tolerância a falhas, é um dos insumos para esta atividade.

Solução:

A Análise de Caso de Uso é uma das atividades previstas na UML, e tem como finalidades:

- identificar classes que desempenham os vários fluxos de eventos no caso de uso;
- distribuir o comportamento do caso de uso entre essas classes usando as "realizações";
- identificar responsabilidades e atributos associações de classes;
- identificar o uso de mecanismos arquiteturais para prover funcionalidades necessárias ao caso de uso e software em geral.

Obviamente, ao realizar essa análise de comportamento, possíveis falhas e omissões se tornam mais evidentes. Com isso, podemos afirmar que a resposta da questão é a alternativa B.

Ainda com relação aos casos de uso, é importante frisar que muitas vezes é ela é tida como uma espécie de análise de caixa preta, no qual os detalhes internos sobre como o sistema responde às ações de um ator estão ausentes ou descritas muito resumidamente. Ou seja, os casos de uso não apresentam detalhes de implementação. Com isso, a alternativa A está errada.

Os diagramas de classes, mencionados na alternativa C, são diagramas estáticos que descrevem os vários tipos de objetos no sistema e o relacionamento entre eles. Tais diagramas não provêem informação de interação entre as classes. Com isso, a alternativa C também está errada.

Os artefatos da análise de caso de uso são Classe de Análise, Realização de Casos de Uso, Modelo de Design, Modelo de Análise. Ou seja, "Realização de Caso de Uso", citado na

alternativa D, não é um produto da atividade "Analisa um Caso de Uso".

Por fim, a alternativa E traz o diagrama de robustez. Tal diagrama geralmente é utilizado para passar da análise (o que) para o desenho (como). Como ele não é um insumo para a análise de caso de uso, esta alternativa está errada.

5. **Assuntos relacionados:** *Engenharia de Software*, *RUP*,
Banca: *CESGRANRIO*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Eng. de Software*
Ano: *2008*
Questão: *52*

Sobre testes no Processo Unificado, é correto afirmar que um(a)

(a). caso de teste é composto por um ou mais planos de testes.

(b). script de teste define o cronograma e a estratégia utilizados na iteração.

(c). modelo de teste é um componente que efetua testes unitários em uma ou mais classes de domínio.

(d). prova de conceito é um tipo especial de caso de teste.

(e). avaliação de teste apresenta os resultados dos testes em termos de defeitos e cobertura.

Solução:

O Processo Unificado (PU) surgiu como um processo popular para o desenvolvimento de software visando à construção de sistemas orientados a objetos (o RUP – Rational Unified Process é um refinamento do PU). Ou seja, Processo Unificado é um processo de desenvolvimento, já o Rational Unified Process é um produto da Rational – IBM, que tem no seu núcleo o Processo Unificado.

O Processo Unificado utiliza um paradigma evolucionário para o desenvolvimento de softwares. O ciclo de vida iterativo é baseado em refinamentos e incrementos sucessivos a fim de convergir para um sistema adequado. Em cada iteração incrementa-se um pouco mais o produto, baseando-se na experiência obtida nas iterações anteriores e no feedback do usuário. O Processo Unificado consiste da repetição de uma série de ciclos durante a vida de um sistema. Cada ciclo é concluído com uma versão do produto pronta para distribuição e é subdividido em quatro fases: concepção, elaboração, construção e transição. Cada fase, por sua vez, é subdivida em iterações que passam por todos os cincos fluxos do trabalho do processo: análise de requisitos, análise, projeto, implementação e teste.

O **fluxo de teste** é desenvolvido com base no produto do fluxo de implementação. Os componentes executáveis são testados exaustivamente no fluxo de teste para então ser disponibilizados aos usuários finais. O principal propósito do fluxo de teste é realizar vários testes e sistematicamente analisar os resultados de cada teste. Componentes que possuírem defeitos retornarão a fluxos anteriores como os fluxos de projeto e implementação, onde os problemas encontrados poderão ser corrigidos.

O teste de um sistema é primeiramente empregado durante a fase de elaboração, quando a arquitetura do sistema é definida, e durante a fase de construção quando o sistema é implementado. Um planejamento inicial de testes pode ser feito durante a fase de concepção.

Na fase de transição, o fluxo de testes se limita ao conserto de defeitos encontrados durante a utilização inicial do sistema. O produto do fluxo de teste é o modelo de teste, esse modelo primeiramente descreve como componentes executáveis, provenientes do fluxo de implementação, são testados.

Mais especificamente, um **modelo de teste** descreve como os testes de integração e de sistema exercitarão componentes executáveis a partir do modelo de implementação. Todos os casos de teste devem ser registrados no modelo de teste. Uma das principais atividades de teste no Processo Unificado é a criação do modelo de teste, que deve descrever como os testes de integração e de sistema exercitarão componentes executáveis a partir do modelo de implementação. O modelo de teste também descreve como os testes de integração devem ser executados, bem como os testes de unidade. O modelo de testes também pode descrever como aspectos específicos dos sistemas testados, como por exemplo, se a interface do usuário é útil e consistente ou se o manual do usuário cumpre seu objetivo.

O papel do fluxo de teste é verificar se os resultados do fluxo de implementação cumprem os requisitos estipulados por clientes e usuários, para que possa ser decidido se o sistema necessita de revisões ou se o processo de desenvolvimento pode continuar.

Um **caso de teste** é um conjunto específico de entradas de teste, condições de execução e resultados esperados, identificados com a finalidade de avaliar um determinado aspecto de um item de teste-alvo.

Um **plano de testes** é uma definição das metas e dos objetivos dos testes no escopo da iteração (ou projeto), os itens-alvo, a abordagem adotada, os recursos necessários e os produtos que serão liberados.

Um **script de teste** é um conjunto de instruções passo a passo que permitem a execução de um teste. Os scripts de teste podem assumir a forma de instruções de texto documentadas e executadas manualmente ou de instruções que podem ser lidas pelo computador para ativar a execução automática do teste.

Uma **prova de conceito** é, como o nome sugere, uma prova de uma teoria. As provas de conceito apenas afirmam algo que antes só existia no papel e que, portanto, não existia nenhuma prova que de fato aquilo funcionaria. Durante o processo da análise arquitetural, aspectos de risco do projeto podem merecer uma investigação mais detalhada, para isso uma prova de conceito pode ser construída e pode tomar as seguintes formas: Modelagem conceitual; Protótipo Rápido; Simulação; Conversão automática de especificações em código; Especificações "executáveis" ou a Construção de "picos invertidos".

A **avaliação de teste** é a avaliação dos resultados de um conjunto de testes. Ela geralmente inclui uma lista de defeitos e seus níveis de prioridade, além de informações como a razão de defeitos por linha de código.

As principais medidas de um teste incluem a cobertura e a qualidade. A cobertura é a medida da abrangência do teste e é expressa pela cobertura dos requisitos e casos de teste ou pela cobertura do código executado. A qualidade é uma medida de confiabilidade, de estabilidade e de desempenho do objetivo do teste (sistema ou aplicativo em teste). Ela se baseia na avaliação dos resultados do teste e na análise das solicitações de mudança (defeitos) identificadas durante o teste.

Dado todo o exposto, a alternativa a ser marcada é a letra (E).

6. **Assuntos relacionados:** *Engenharia de Software*, *RUP*,
Banca: *FCC*
Instituição: *MPU*
Cargo: *Analista de Desenvolvimento de Sistemas*
Ano: *2007*
Questão: *67*

No ciclo de vida do Processo Unificado, os testes têm seu apogeu demonstrado na linha divisória entre

(a). Concepção e Elaboração.

(b). Requisitos e Análise.

(c). Projeto e Construção.

(d). Construção e Concepção.

(e). Construção e Transição.

Solução:

O Processo Unificado é um processo da Engenharia de Software e foi elaborado pela Rational (*Rational Unified Process* – RUP), visando à construção de sistemas utilizando o paradigma de orientação a objetos. Explora integralmente as capacidades da Linguagem de Modelagem Unificada (*Unified Modeling Language* – UML). Na verdade, o RUP não é exatamente um processo, mas uma infra-estrutura genérica de processo que pode ser especializada para uma ampla classe de sistemas de softwares.

O RUP está fundamentado em três princípios básicos: desenvolvimento iterativo, centrado à arquitetura e guiado por casos de uso. Os casos de uso orientam todo o processo de desenvolvimento do software e com base neles, são criados uma série de modelos de análise, projeto e implementação. É centrado à arquitetura, porque defende a definição de um esqueleto (a arquitetura) para o sistema, ganhando corpo gradualmente ao logo do desenvolvimento. O RUP é iterativo e incremental, oferecendo um método para particionar o trabalho em porções menores. A arquitetura provê a estrutura para guiar o desenvolvimento, e os casos de uso definem as metas e conduzem o trabalho de cada interação.

O ciclo de vida adotado no RUP é o evolutivo. Entretanto, a organização em fases é adotada para comportar os ciclos de desenvolvimento, permitindo uma gerência mais efetiva de projetos complexos. Diferentemente das tradicionais fases definidas na maioria dos modelos de ciclo de vida (planejamento, levantamento de requisitos, análise, projeto, implementação e testes), fases ortogonais a essas são definidas no RUP, as quais são: Concepção, Elaboração, Construção e Transição.

Na fase de Concepção, o escopo do projeto e suas fronteiras são estabelecidos, determinando os principais casos de uso do sistema, que devem ser precisos para proceder com as estimativas de prazos e de custos. A ênfase desta fase está no planejamento, sendo necessário levantar requisitos do sistema e preliminarmente analisá-los.

A fase de Elaboração envolve analisar refinadamente o domínio do problema, estabelecer uma arquitetura, desenvolver um plano de projeto para o sistema a ser construído e eliminar os elementos do projeto que oferecem maior risco. Esta fase assegura que os requisitos, a arquitetura e os planos estão suficientemente estáveis e os riscos mitigados.

Na fase de Construção, é desenvolvido um produto completo de uma forma iterativa e incremental, de modo que esteja pronto para o usuário.

Na fase de Transição, o software é disponibilizado para o usuário. Nesta fase, atividades, como construir novas versões para permitir ajustes no software, corrigir problemas ou concluir alguma características postergadas, são realizadas.

Dentro de cada fase, um conjunto de iterações envolvendo planejamento, levantamento de requisitos, análise, projeto e implementação e testes é realizado. Entretanto, de uma fase para outra, a ênfase sobre as atividades mudam. Na fase de Concepção, o foco principal é sobre o entendimento de requisitos e a determinação do escopo do projeto (planejamento e levantamento dos requisitos). Na fase de Elaboração, o foco principal é a captura e modelagem dos requisitos (levantamento de requisitos e a análise), e algum trabalho de projeto e implementação é realizado para prototipar a arquitetura, evitando alguns tipos de riscos. Na fase de Construção, o foco principal concentra-se no projeto e na implementação, visando evoluir e dar corpo ao protótipo inicial, até obter o primeiro produto operacional. Na fase de Transição, o foco é nos testes, visando garantir que o sistema possui um nível de qualidade aceitável, e ainda, os usuários são treinados, as características são ajustadas e os elementos esquecidos são adicionados.

Conforme explicado anteriormente, os testes têm seu apogeu demonstrado na linha divisória entre a Construção e a Transição. Portanto, a alternativa correta é a letra E.

7. **Assuntos relacionados:** *Metodologia de Desenvolvimento de Software*, *Extreme Programming (XP)*, *Scrum*, *DSDM*, *MVC*,
Banca: *FCC*
Instituição: *TRT 15a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2009*
Questão: *52*

   Histórias de usuários na atividade de planejamento, encorajamento de uso de cartões CRC e de refabricação, reuniões em pé e programação em pares são características típicas do modelo de processo de software

   (a). XP.

   (b). SCRUM.

   (c). DSDM.

   (d). DAS.

   (e). MVC.

Solução:

(A) CORRETA

A metodologia XP (Extreme Programming) é destinada a grupos pequenos de desenvolvimento, e em projetos de duração de até 36 meses. Suas principais características são:

- metáforas: utilização de comparações e analogias para facilitar entendimento;
- design simples do sistema;
- testes automatizados: testes de unidade e de aceitação;
- refabricação: todo desenvolvedor tem o dever de melhorar um código que esteja funcionado porém está mal escrito;
- programação de dupla: todo código deve ser implementado em dupla;
- reuniões curtas e em pé;
- semana de 40 horas: ao se trabalhar a mais se obtém resultados no início, mas com o passar do tempo surge desgaste;
- integração contínua: eliminar erros graves de integração;
- releases curtos: release é um conjunto de funcionalidades bem definidas e que representam algum valor para o cliente. Um projeto XP pode ter uma ou mais releases no seu ciclo;
- CRC: linguagem para modelagem de classes do XP que utiliza os story cards (cartões escritos pelos usuários onde são descritas funcionalidades do sistema).

Como se pode perceber acima, é esse o modelo de processo de software que possui as características mencionadas no enunciado. Portanto, esta é a alternativa correta.

(B) ERRADA

Scrum é um processo para construir software de forma incremental em ambientes complexos, onde os requisitos não são claros ou mudam com muita frequência. A metodologia é baseada em princípios semelhantes aos de XP. As principais são:

- equipes pequenas, multidisciplinares e auto-organizadas;
- requisitos pouco estáveis ou desconhecidos;
- iterações curtas (sprints) que seguem o ciclo PDCA e entregam incrementos de software prontos;
- reuniões diárias e curtas geralmente realizadas em pé;
- Scrum Master: membro que tem como função primária remover qualquer impedimento à habilidade de uma equipe de entregar o objetivo do sprint. O Scrum Master não é o líder da equipe (já que as equipes são auto-organizadas), mas atua como um mediador entre a equipe e qualquer influência desestabilizadora.

Muito embora Scrum e XP sejam complementares, pois Scrum provê práticas ágeis de gerenciamento enquanto XP provê práticas integradas de engenharia de software, e tenham algumas características em comum, nem todas as características apresentadas no enunciado são associadas ao Scrum. Portanto, esta alternativa não está correta.

(C) ERRADA

DSDM (Dynamic Systems Development Method) é progenitor do XP. Ele é um arcabouço para desenvolvimento rápido de aplicações (RAD). Suas principais particularidades são:

- projetos são divididos em 3 fases: pré-projeto, ciclo de vida e pós-projeto;
- a fase ciclo de vida é subdividida em 5 estágios: análise de viabilidade; análise de negócio; iteração do modelo funcional; iteração de elaboração e construção; e implantação;
- há 9 princípios nesse método de desenvolvimento:
  - alto envolvimento do usuário durante o desenvolvimento;
  - autonomia da equipe em relação a decisões;
  - foco em entregas incrementais frequentes;
  - o critério principal de eficácia é em relação a utilidade da entrega para o negócio, e não necessariamente com relação a aderência às especificações;
  - feedbacks de usuários são considerados a cada iteração;
  - todas as alterações são reversíveis;
  - o escopo e requisitos de alto nível são especificados antes do início do projeto;
  - testes são tratados fora do ciclo de vida do projeto;
  - comunicação constante e de qualidade durante todo o projeto.

Como se pode notar, o DSDM não apresenta as características em questão. Ou seja, esta alternativa está errada.

(D) ERRADA

A princípio, DAS não se refere a nenhum modelo de processo de software conhecido. Por isso, conclui-se que esta não é a alternativa correta.

Fora do âmbito de modelos de processo de desenvolvimento de software, DAS se refere a Direct-Attached Storage, uma tecnologia de armazenamento de dados em que o dispositivo de armazenamento é interligado diretamente à máquina detentora dos dados. Ou seja, não há nenhuma estrutura de rede entre o "dono" do dados e o dispositivo de armazenamento. Os principais protocolos utilizados nesse tipo de storage são: ATA, SATA, SCSI, SAS e

Fibre Channel.

(E) ERRADA

MVC (Model-view-controller) não é um modelo de processo de software. Portanto, esta alternativa está errada.

MVC é um padrão de arquitetura de software utilizado para separar dados ou lógica de negócios (Model) da interface do usuário (View) e do fluxo da aplicação (Control). Sua ideia central é permitir que uma mesma lógica de negócio possa ser acessada e visualizada através de várias interfaces. Outro benefício considerável dessa abordagem é que alterações feitas no layout não afetam a manipulação de dados.

8. **Assuntos relacionados:** *Desenvolvimento de Sistemas*, *Metodologia de Desenvolvimento de Software*, *Extreme Programming (XP)*,
Banca: *Cesgranrio*
Instituição: *BNDES*
Cargo: *Analista de Sistemas - Desenvolvimento*
Ano: *2008*
Questão: *31*

Que situação favorece a escolha do uso de XP para um projeto de desenvolvimento de software, em oposição à escolha do RUP ou do modelo Cascata?

(a). Equipe do projeto localizada em diferentes cidades e com poucos recursos de colaboração.

(b). Equipe do projeto formada por pessoas com alto grau de competitividade.

(c). Cliente do projeto trabalhando em parceria com a equipe do projeto e sempre disponível para retirar dúvidas.

(d). Requisitos do software com pequena probabilidade de mudanças.

(e). Presença de um processo organizacional que exige a elaboração de vários documentos específicos para cada projeto.

Solução:

O Rational Unified Process (RUP) é um framework muito difundido e utilizado que pode ser adaptado a vários tipos de projetos de software. Podem ser derivados do RUP processos para projetos de vários portes, pois este framework define uma grande lista de papéis, artefatos, atividades e fluxos. No entanto, o RUP é tido como muito complexo, e costuma ser visto como um pesado e burocrático, ao contrário das metodologias ágeis.

Em contrapartida, o EXtreme Programming (XP) aparece como uma alternativa mais leve para times de tamanho pequeno e médio porte, que desenvolvem software em um contexto de requisitos vagos e rapidamente modificados. O XP enfatiza a codificação e os testes de códigos, considerando a presença constante dos clientes no desenvolvimento fundamental. Pela característica de simplicidade que esta técnica apresenta, poucos artefatos, papéis e atividades são definidos. Portanto, a resposta da questão é a alternativa C.

Ainda sobre o XP, são muito úteis ainda as seguintes informações. Os quatro valores fundamentais da metodologia XP são a comunicação, simplicidade, feedback e coragem. Para fazer valer tais valores, a metodologia adota como práticas as seguintes:

- **Jogo de Planejamento**: O desenvolvimento é feito em iterações semanais. No início da semana, desenvolvedores e cliente reúnem-se para priorizar as funcionalidades. O escopo é reavaliado semanalmente, e o projeto é regido por um contrato de escopo negociável;
- **Pequenas Versões**: A liberação de pequenas versões funcionais do projeto auxilia muito no processo de aceitação por parte do cliente, que já pode testar uma parte do sistema que está comprando;
- **Metáforas**: A metologia prega o uso de metáforas, como forma de facilitar a comunicação com o clientes;
- **Time Coeso**: A equipe de desenvolvimento é formada pelo cliente e pela equipe de desenvolvimento;

- **Ritmo Sustentável**: Trabalhar com qualidade, buscando ter ritmo de trabalho saudável, sem horas extras. Horas extras são permitidas quando trouxerem produtividade para a execução do projeto;
- **Reuniões em pé**: Reuniões em pé para não se perder o foco nos assuntos, produzindo reuniões rápidas, apenas abordando tarefas realizadas e tarefas a realizar pela equipe;
- **Posse Coletiva do Código**: O código fonte não tem dono e ninguém precisa solicitar permissão para poder modificar o mesmo. Tal prática tem como objetivo fazer com que a equipe conheça todas as partes do sistema;
- **Programação em Duplas**: Geralmente a dupla é formada por um iniciante na linguagem e outra pessoa funcionando como um instrutor. Como é apenas um computador, o novato é que fica à frente fazendo a codificação, e o instrutor acompanha ajudando a desenvolver suas habilidades. Desta forma o programa sempre é revisto por duas pessoas, evitando e diminuindo assim a possibilidade de erros;
- **Padrões de Codificação**: A equipe de desenvolvimento precisa estabelecer regras para programar e todos devem seguir estas regras. Desta forma, parecerá que todo o código foi editado pela mesma pessoa, mesmo quando a equipe possui uma quantidade maior de programadores;
- **Refatoração**: É um processo que permite a melhoria continua da programação, que tem como alvos a melhoria da legibilidade do código, maiores modularização e reaproveitamento do código.

9. **Assuntos relacionados:** *Gerência de Projeto*, *Ciclo de Vida de Projeto*, *Estimativa de Custo*, *Refatoração*,
Banca: *Cesgranrio*
Instituição: *BNDES*
Cargo: *Analista de Sistemas - Desenvolvimento*
Ano: *2008*
Questão: *38*

Suponha que um projeto de software siga o modelo cascata e utilize técnicas de refatoração apoiadas por uma ferramenta durante a etapa de implementação. Qual o impacto resultante na etapa de análise e projeto?

(a). Pode aumentar o trabalho do analista, pois o código deve estar preparado para utilizar as técnicas de refatoração.

(b). Pode aumentar o trabalho do analista se o profissional que realizar a etapa de análise for diferente do profissional que implementará o software.

(c). Pode diminuir o trabalho do analista, mas apenas se o profissional que realizar a etapa de análise for o mesmo que implementar o software.

(d). Pode diminuir o trabalho do analista, já que o profissional de análise e projeto saberá que mudanças futuras no modelo gerado durante essa etapa poderão ser realizadas com um custo menor na etapa de implementação.

(e). Não terá impacto se o profissional já conhecer as técnicas de refatoração.

Solução:

O modelo cascata (*waterfall*) ou clássico é também conhecido como abordagem "top-down". Comparado com outros modelos de desenvolvimento de software, ele é mais rígido e menos administrativo. O modelo cascata é um dos modelos mais importantes, pois é referência para muitos outros modelos, embora seja criticado por ser linear, rígido e monolítico. Inspirado em modelos de outras atividades de engenharia, este modelo argumenta que cada atividade apenas deve ser iniciada quando a outra estiver terminada e verificada.

Existem muitas variações do modelo cascata propostas por diferentes pesquisadores ou empresas de desenvolvimento. A característica comum é um fluxo linear e sequencial de atividades semelhantes. A Figura 2 indica as atividades comumente utilizadas em um modelo cascata.

A fase de requerimento ou de análise de requisitos é focada na coleta os requisitos do software e na geração do documento de especificação do sistema que serve de base para o orçamento, cronograma, esforço, ferramentas a serem utilizadas, etc. Essa fase deve buscar uma compreensão clara e precisa do domínio do problema e das funcionalidades do software através do levantamento e revisão em conjunto com representantes do cliente, usuários chaves e outros especialistas da área de aplicação. Já a fase de projeto concentra-se na definição das estruturas de dados, arquitetura do software, detalhes procedimentais e caracterização da interface.

A refatoração é o processo de alteração de um sistema de software de modo que o comportamento observável do código não mude, mas que sua estrutura interna seja melhorada. É uma maneira disciplinada de aperfeiçoar o código que minimiza a chance de introdução de falhas. Em essência, refatorar é melhorar o projeto do código após este ter sido escrito.

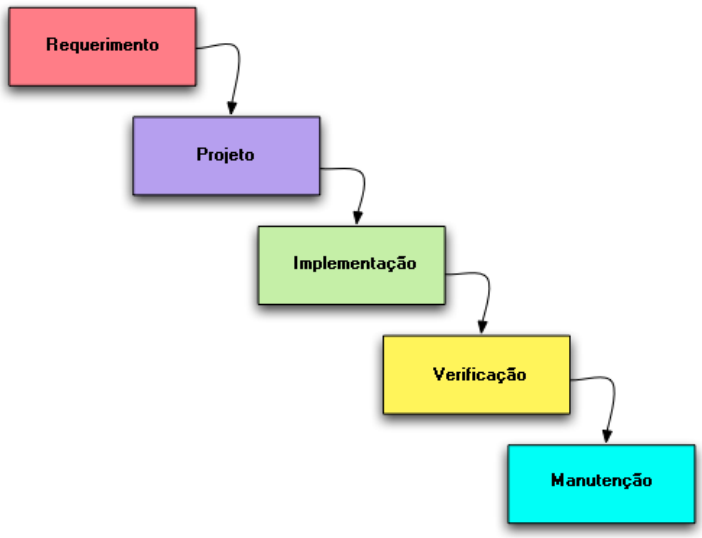


Figura 2: modelo cascata.

As técnicas de refatoração incluem, entre outras coisas, metodologias para detectar possíveis problemas no código.

Quando as técnicas de refatoração são utilizadas de forma correta, o software aumenta consideravelmente seu tempo de vida útil, sua extensibilidade e modularidade. A maioria das ferramentas de refatoração segue um processo comum, no qual operações de refatoração são aplicadas ao código-fonte dos programas de acordo com passos bem definidos. Em geral, esses passos incluem:

1. Detectar trechos do código com oportunidades de refatorações.
2. Determinar que refatorações aplicar a cada trecho do código selecionado.
3. Garantir que as refatorações escolhidas preservem o comportamento.
4. Aplicar as refatorações escolhidas aos seus respectivos locais.
5. Verificar que o comportamento do programa foi preservado após as refatorações terem sido aplicadas.

Apesar de podermos aplicar refatorações manualmente, ferramentas que automatizem o processo diminuem o risco de erros e inconsistência no código, além de poupar um grande trabalho em se tratando de sistemas com centenas ou milhares de linhas de código.

Sabemos, então, que a refatoração aumenta a extensibilidade e a modularidade do software e que o apoio de uma ferramenta de refatoração é importantíssimo para garantir a qualidade e um menor custo da mesma. Isso torna o trabalho dos analistas das fases de análise e projeto muito mais seguro, pois eles saberão que mudanças que forem feitas no futuro poderão ser mais facilmente absorvidas na fase de implementação do software. Logo, a alternativa correta é a letra **D**.

10. **Assuntos relacionados:** *Processo Unificado*, *Extreme Programming (XP)*, *Scrum*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Processos de Negócio*
Ano: *2008*
Questão: *53*

Diversos processos de software emergiram nos últimos anos, principalmente aqueles de natureza iterativa. Podemos citar o Processo Unificado, XP (Extreme Programming) e, mais recentemente, o Scrum, como alguns dos mais difundidos. Correlacione, a seguir, esses processos de software e suas características.

Processos	Características
I - Processo Unificado II - Extreme Programming III - Scrum	P - Reuniões em pé, diárias, com perguntas específicas Q - Refatoração frequente R - Priorização dos maiores riscos S - Eliminação do desperdício

A relação correta é

(a). I - P, II - Q, III - R.

(b). I - R, II - Q, III - P.

(c). I - R, II - S, III - Q.

(d). I - S, II - P, III - R.

(e). I - S, II - R, III - P.

Solução:

Processo Unificado

O Processo Unificado (UP) de desenvolvimento de software é o conjunto de atividades necessárias para transformar requisitos do usuário em um sistema de software. O processo unificado de desenvolvimento de sistemas combina os ciclos iterativos e incrementais para a construção de softwares. É fundamental na visão de que o avanço de um projeto deve estar baseado na construção de artefatos de software, e não apenas em documentação.

Ele é baseado em componentes, o que significa o sistema ser construído a partir de componentes de software interconectados via interfaces muito bem definidas. O processo unificado utiliza a Linguagem de Modelagem Unificada (*Unified Modeling Language* – UML) no preparo de todos os artefatos do sistema.

Um projeto usando o processo unificado sempre vai ser iterativo e incremental. Uma iteração é um mini projeto que resulta em versão do sistema. Como a cada iteração teremos uma versão melhor que a anterior, temos aí a questão do incremental. Os riscos de projeto podem ser tratados dentro de cada iteração. Cada iteração/incremento produz um produto com novas características implementadas. Com isto é importante sempre integrar o trabalho gerado para ter certeza que o projeto continua funcionando como deve.

O Processo Unificado organiza suas iterações em quatro fases principais:

1. Concepção: o objetivo desta fase é levantar, de forma genérica e pouco precisa, o escopo do projeto. Não deve existir aqui a pretensão de especificar de forma detalhada requisitos, a ideia é ter uma visão inicial do problema, estimar de forma vaga esforço e prazos e determinar se o projeto é viável e merece uma análise mais profunda.
2. Elaboração: na fase de elaboração todos (ou a grande maioria dos requisitos) são levantados em detalhes. Numa primeira iteração um ou dois requisitos, os de maior risco e valor arquitetural, são especificados são implementados e servem como base de avaliação junto ao usuário e desenvolvedores para o planejamento da próxima iteração. Em cada nova iteração na fase de elaboração pode haver um seminário de requisitos, onde requisitos antigos são melhores esclarecidos e novos são detalhados. Ao fim da fase, 90
3. Construção: implementação iterativa dos elementos restantes de menor risco e mais fáceis e preparação para a implantação.
4. Transição: testes finais e implantação.

Xtreme Programming

Extreme Programming (XP) é um processo de desenvolvimento que possibilita a criação de software de alta qualidade, de maneira ágil, econômica e flexível, empregando a equipe de desenvolvimento, em atividades que produzam resultados rápidos na forma de software testado, e ainda customizando o produto de acordo com a necessidade do usuário.

Cada vez mais as empresas convivem com ambientes de negócios que requerem mudanças freqüentes em seus processos, as quais afetam os projetos de software. Os processos de desenvolvimento tradicionais são caracterizados por uma grande quantidade de atividades e artefatos que buscam proteger o software contra mudanças, o que faz pouco ou nenhum sentido, visto que os projetos devem se adaptar a tais mudanças ao invés de evitá-las. O processo Extreme Programming define as seguintes práticas: refatoração, programação em par, mudanças rápidas, feedback constante do cliente, desenvolvimento iterativo, testes automatizados, entre outras.

O XP concentra os esforços da equipe de desenvolvimento em atividades que geram resultados rapidamente na forma de software intensamente testado e alinhado às necessidades de seus usuários. Além disso, simplifica e organiza o trabalho combinando técnicas comprovadamente eficazes e eliminando atividades redundantes. Por fim, reduz o risco dos projetos desenvolvendo software de forma iterativa e reavaliando permanentemente as prioridades dos usuários.

Scrum

Scrum é um processo de desenvolvimento iterativo e incremental que pode ser aplicado a qualquer produto ou no gerenciamento de qualquer atividade complexa. É um processo ágil que permite manter o foco na entrega do maior valor de negócio, no menor tempo possível. Isto permite a rápida e contínua inspeção do software em produção em intervalos de duas a quatro semanas. As necessidades do negócio determinam as prioridades do desenvolvimento de um sistema. As equipes se auto organizam para definir a melhor maneira de entregar as funcionalidades de maior prioridade. Entre cada duas a quatro semanas todos podem ver o real software em produção, decidindo se o mesmo deve ser liberado ou continuar a ser aprimorado por mais uma iteração. As reuniões diárias são usadas para avaliar os progressos do projeto e as barreiras encontradas durante o desenvolvimento, essa é uma característica

do processo.

Como pode ser observado, o processo unificado trabalha principalmente com os requisitos de maior risco e valor arquitetural, o processo Extreme Programming utiliza como uma de suas práticas a refatoração e por último o processo Scrum que tem como característica as reuniões diária usadas para avaliar os progressos do projeto.

11. **Assuntos relacionados:** *RUP*, *Processo Unificado*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas Pleno - Processos*
Ano: *2006*
Questão: *47*

Um gerente de projeto decidiu utilizar o Processo Unificado (RUP – rational unified process) como seu processo de desenvolvimento de software. Com base no RUP, quais os objetivos que o gerente deve direcionar para a fase de Elaboração?

(a). Produzir Documento Visão completo e estável; detalhar os atores e casos de uso chave; determinar pelo menos uma solução possível para o problema.

(b). Produzir Documento Visão completo e estável; fazer o design dos casos de uso críticos; obter um entendimento mais detalhado dos requerimentos.

(c). Fazer o design dos casos de uso críticos; obter um entendimento mais detalhado dos requerimentos; implementar e testar cenários críticos.

(d). Fazer o design do Banco de Dados; implementar e testar cenários críticos; liberar uma versão beta do produto.

(e). Detalhar os atores e casos de uso chave; fazer o design, implementação, validação e estabelecer uma linha de base para a arquitetura; determinar pelo menos uma solução possível para o problema.

Solução:

O RUP (Rational Unified Process) é um produto/processo de Engenharia de Software proprietário baseado no Processo Unificado (PU). O PU surgiu como um processo popular para desenvolvimento de software visando construção de sistemas orientados a objetos. Já o RUP foi criado pela Rational Software Corporation, que foi comprada em 2003 pela IBM. Em decorrência dessa compra, ele ganhou um novo nome IRUP (IBM Rational Unified Process). Pode-se dizer, portanto, que o Processo Unificado é um processo de desenvolvimento de software e o Rational Unified Process é o produto da IBM/Rational que tem no seu núcleo o Processo Unificado.

No website da IBM (`http://www-01.ibm.com/software/br/rational/rup.shtml`) pode-se encontrar a seguinte definição:

IBM Rational Unified Process, ou RUP, é uma plataforma de processo de desenvolvimento de software configurável que oferece melhores práticas comprovadas e uma arquitetura configurável. A plataforma RUP inclui:

- *Ferramentas para configurar a RUP para as necessidades específicas de seu projeto.*
- *Ferramentas para desenvolver seu próprio conhecimento interno em componentes de processo.*
- *Ferramentas de implementação eficientes e personalizáveis baseadas na Web.*
- *Uma comunidade online para trocar melhores práticas com colegas e líderes do segmento de mercado.*

RUP é baseado em um conjunto de princípios de desenvolvimento de software e melhores práticas, por exemplo:

- desenvolvimento de software iterativo
- gerenciamento de requisitos
- uso de arquitetura baseada em componente
- modelagem visual de software
- verificação da qualidade do software
- controle de alteração no software

Além disso, ele herda as principais características do Processo Unificado. Essas características se relacionam entre si e são igualmente importantes:

- **Orientado a Diagrama de Casos de Uso:** um caso de uso, de acordo com a UML (Unified Modeling Language), é uma sequência de ações de um sistema que devolve ao usuário um resultado de valor. Um conjunto de casos de uso definido sob determinado contexto forma um Diagrama de Casos de Uso, que descreve uma funcionalidade do sistema sob esse contexto. Em outras palavras, um Diagrama de Casos de Uso define a funcionalidade de um sistema para cada tipo de usuário. Esse tipo de abordagem favorece o atendimento das necessidades de cada tipo de usuário que interage com o sistema, evitando, dessa forma, que o sistema possa ser desenvolvido a ponto de apresentar funcionalidades desnecessárias;
- **Centrado na arquitetura do sistema:** arquitetura de sistema é uma visão do projeto como um todo, destacando suas características mais importantes de forma abrangente e sem detalhes específicos. Esse tipo de abordagem auxilia o arquiteto a se concentrar nas metas corretas, como inteligibilidade, poder de recuperação para mudanças futuras e reutilização. Ou seja, a arquitetura de um sistema deve ser projetada a ponto de permitir que o sistema evolua, não apenas durante o início do seu desenvolvimento, mas também ao longo das iterações futuras;
- **Metodologia iterativa e incremental:** uma iteração pode ser entendida como um miniprojeto, que resulta em uma nova versão do sistema. Justamente devido ao fato de a cada interação ser gerada uma nova versão do sistema que essa metodologia também se apresenta como incremental. Perceba que estas duas características, iterativa e incremental, nem sempre aparecem juntas. Há metodologias de desenvolvimento de software que são iterativos, mas não incrementais.

O RUP, assim com o Processo Unificado, organiza suas iterações em quatro fases:

1. **Concepção:** o objetivo desta fase é levantar, de forma genérica e pouco precisa, o escopo do projeto. Não deve existir aqui a pretensão de especificar de forma detalhada requisitos, a ideia é ter uma visão inicial do problema, estimar de forma vaga esforço e prazos e determinar se o projeto é viável e merece uma análise mais profunda;
2. **Elaboração:** nesta fase, todos os requisitos (ou a grande maioria deles) são levantados em detalhes. Em uma primeira iteração, um ou dois requisitos, os de maior risco e valor arquitetural, são especificados e implementados. Eles servirão como base de avaliação junto aos usuários e desenvolvedores para o planejamento da próxima iteração. Em cada nova iteração dentro da fase de elaboração, pode haver um seminário de requisitos, onde requisitos antigos são melhores esclarecidos e novos são detalhados. Ao fim desta fase, 90% dos requisitos foram levantados em detalhes, o núcleo do sistema foi implementado com alta qualidade, os principais riscos foram tratados e pode-se então fazer estimativas mais realistas;
3. **Construção:** implementação iterativa dos elementos restantes de menor risco e menos complexos. Há também uma preparação para a implantação do sistema;

4. **Transição:** testes finais e implantação do sistema.

Tendo em vista o exposto, em especial as quatro fases anteriormente descritas, não é difícil identificar que a alternativa C é a única que traz características típicas da fase de Elaboração.

12. **Assuntos relacionados:** *Engenharia de Software*, *Modelo Ágil de Software*, *DAS*, *DSDM*, *FDD*, *XP*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas Pleno - Processos*
Ano: *2006*
Questão: *58*

Há um considerável debate sobre os benefícios e a aplicabilidade do desenvolvimento ágil de software em contraposição aos processos mais convencionais de engenharia de software. Relacione o modelo ágil de software com a sua respectiva característica.

Modelo

I - DAS
II - DSDM
III - FDD
IV - XP

Característica

(P) Define um ciclo de vida que incorpora três fases: especulação, colaboração e aprendizado. Durante a fase de aprendizado, à medida que os membros de uma equipe começam a desenvolver os componentes que fazem parte de um ciclo adaptativo, a ênfase está tanto no aprendizado quanto no progresso em direção a um ciclo completo.

(Q) O conceito característica é uma função valorizada pelo cliente, que pode ser implementada em duas semanas ou menos. Este modelo define seis marcos de referência durante o projeto e implementação de uma característica: travessia do projeto, projeto, inspeção de projeto, código, inspeção de código, promoção para construção.

(R) Fornece um arcabouço para construir e manter sistemas que satisfazem às restrições de prazo apertadas por meio do uso de prototipagem incremental em ambiente controlado de projeto. Essa abordagem sugere uma filosofia que é emprestada de uma versão modificada do princípio de Pareto.

A relação correta é:

(a). I - P, II - Q, III - R.
(b). I - P, II - R, III - Q.
(c). I - Q, III - R, IV - P.
(d). I - P, III - R, IV - Q.
(e). II - Q, III - P, IV - R.

Solução:

A partir da década de 90, surgiram novos métodos que sugerem uma abordagem de desenvolvimento ágil onde os processos adotados tentam se adaptar às mudanças, apoiando a equipe de desenvolvimento no seu trabalho. Tais métodos surgiram como uma reação aos métodos tradicionais de desenvolvimento de sistemas (o modelo do ciclo de vida, por exemplo), ganhando com o passar dos anos um número cada vez maior de adeptos.

Com o intuito de definir um manifesto para encorajar melhores meios de desenvolver sistemas, em fevereiro de 2001 um grupo inicial de 17 metodologistas formou a **Aliança para o Desenvolvimento Ágil de Software**, que formulou um manifesto contendo um conjunto de princípios que definem critérios para os processos de desenvolvimento ágil de sistemas. São eles:

1. a prioridade é satisfazer ao cliente através de entregas contínuas e freqüentes;
2. receber bem as mudanças de requisitos, mesmo em uma fase avançada do projeto;
3. entregas com freqüência, sempre na menor escala de tempo.
4. as equipes de negócio e de desenvolvimento devem trabalhar juntas diariamente;
5. manter uma equipe motivada fornecendo ambiente, apoio e confiança necessários;
6. a maneira mais eficiente da informação circular através de uma conversa face a face;
7. ter o sistema funcionando é a melhor medida de progresso;
8. processos ágeis promovem o desenvolvimento sustentável;
9. atenção contínua a excelência técnica e a um bom projeto aumentam a agilidade;
10. simplicidade é essencial;
11. as melhores arquiteturas, requisitos e projetos provêm de equipes organizadas;
12. em intervalos regulares, a equipe deve refletir sobre como se tornar mais eficaz.

É importante entendermos que metodologias ágeis são uma atitude, não um processo prescritivo. Elas agem como um suplemento aos métodos existentes (não é uma metodologia completa). Podemos ver a adoção destas metodologias como uma forma efetiva de se trabalhar em conjunto para atingir as necessidades das partes interessadas no projeto, utilizando um modelo iterativo e incremental, onde o cliente faz parte da equipe de desenvolvimento.

Agora, abordaremos as quatro metodologias ágeis apresentadas.

XP (*eXtreme Programming*- Programação extrema) é uma metodologia ágil de desenvolvimento de software que reúne boas práticas de programação e de gerenciamento de projetos com o objetivo de produzir software de alta qualidade. Apresentamos abaixo os valores da XP. Os valores são as diretrizes da XP que definem as atitudes das equipes e as principais prioridades da metodologia. Para uma empresa estar realmente utilizando o XP, ela deve respeitar e utilizar todos os valores e práticas listadas abaixo e caso um destes valores ou práticas não seja utilizado pela empresa, esta empresa não está trabalhando com a metodologia XP.

- *feedback* rápido;
- simplicidade é o melhor negócio;
- carregue a bandeira das mudanças (não valorize o medo);
- comunicação intensa;

As práticas são um conjunto de atividades que deverão ser seguidas pelas equipes que desejam utilizar a XP. Os valores já apresentados somados a estas práticas são um conjunto entrelaçado de boas atitudes. São elas:

- Cliente disponível ou presente: o XP sugere que o cliente esteja no dia-a-dia do projeto, acompanhando os passos dos desenvolvedores, onde a sua ausência representa sérios riscos ao projeto;

- Jogo de planejamento: este planejamento é feito várias vezes durante o projeto. É o momento onde o cliente solicita as funcionalidades desejadas através de estórias (pequenos cartões em que o cliente deve descrever o que deseja com suas palavras e da forma mais simples possível), onde a equipe estima o custo de cada estória e planeja as *releases* e as iterações. todas as funcionalidades do sistema são descritas em estórias;
- *Stand up meeting*: reunião rápida pela manhã, com aproximadamente 20 minutos;
- Programação em par: o XP exige que todo e qualquer código implementado no projeto seja efetuado em dupla;
- *Refactoring*: o XP prega que todo desenvolvedor ao encontrar um código duplicado, pouco legível, mal codificado, sem padronização, lento, com código legado ou uso incorreto de outras implementações, tem por obrigação alterar este código deixando-o mais legível e simples;
- Desenvolvimento guiado por testes: todo código implementando deve coexistir com um teste automatizado, assim garantindo o pleno funcionamento da nova funcionalidade;
- Código coletivo: cada desenvolvedor tem acesso a qualquer parte do sistema e tem liberdade para alterá-la a qualquer momento e sem qualquer tipo de aviso;
- Padrões de desenvolvimento: uma forma clara de se comunicar através do código, é manter um padrão no projeto para que qualquer um possa rapidamente entender o código lido;
- *Design* simples: todas as práticas do XP focam que o maior valor possível seja gerado para o cliente;
- Metáfora: ao realizar comparações e analogias com o assunto que está em questão, as pessoas passarão a entender deste assunto de uma forma muito mais rápida e possivelmente não a esquecerão mais;
- Ritmo sustentável: um trabalhador não deve ser submetido a uma carga horária superior a 40 horas semanais;
- Integração contínua: se possível, a integração deve ser efetuada diversas vezes ao dia para que toda a equipe tenha conhecimento do código recém desenvolvido;
- *Releases* curtos: o XP pretende dar o máximo de valor econômico ao cliente em um curto espaço de tempo.

Resumindo, a metodologia de desenvolvimento *eXtreme Programming* pode ser considerada extremamente nova, porém vem acompanhando as necessidades humanas dos desenvolvedores pelo mundo. Uma empresa, ao utilizar este processo por completo, só estará agregando valor aos seus negócios e melhorando o ambiente de seus colaboradores e clientes, tratando-os como pessoas e parceiros. Aqui, podemos concluir que a metodologia XP não está relacionada com nenhuma característica apresentada pelo enunciado(já podemos eliminar as alternativas C, D e E).

FDD (*Feature Driven Development* — Desenvolvimento Guiado por Funcionalidades (Características, a tradução já sugere uma associação com a característica Q, mas vamos prosseguir) é uma metodologia ágil para o processo de engenharia de software, com foco na entrega frequente do sistema funcionando e na utilização de boas práticas durante o ciclo de desenvolvimento. Trata-se de uma metodologia muito objetiva, possuindo apenas duas fases:

- Concepção & Planejamento: pensar um pouco antes de fazer (tipicamente de 1 a 2 semanas);

- Construção: fazer de forma iterativa (tipicamente em iterações de 2 semanas).

Além disso, a FDD possui cinco processos bem definidos e integrados, são eles:

1. desenvolver um modelo abrangente (DMA): análise orientada por objetos;
2. construir a lista de funcionalidades (CLF): decomposição funcional do modelo do domínio;
3. planejar por funcionalidade (PPF): planejamento incremental;
4. detalhar por funcionalidade (DPF): desenho (projeto) orientado por objetos;
5. construir por funcionalidade (CPF): programação e teste orientados por objetos.

Aqui, apresentamos as características da FDD:

- situa-se numa posição intermediária entre as abordagens mais tradicionais e as ágeis;
- envolvimento do cliente nos processos de planejamento e desenvolvimento de software;
- não é extremamente focada apenas na programação ou no modelo, mas sim utiliza o bom senso para abstrair o melhor dos dois mundos;
- tamanho dos times de 16 - 20 membros, suportando composições bem maiores;
- entrega resultados funcionais e tangíveis a cada 2 semanas ou menos (isto sugere uma associação com a característica Q);
- pequenos blocos de funcionalidades (*features*) valorizados pelo cliente menos (isto reforça a associação com a característica Q);
- planejamento detalhado e guiado para medição;
- produção de software com qualidade.

As práticas que deverão ser seguidas pelas equipes que desejam utilizar a FDD são:

- modelagem de objetos de domínio;
- desenvolvimento por *feature*;
- posse individual do código/classe;
- equipes de *features*;
- inspeções e *builds* reguladores;
- gerenciamento de configuração;
- relatório/visibilidade de resultados.

Por fim, apresentamos os seis marcos (*milestosnes*) que a FDD possui em seu ciclo interativo (constituído pelos processos DPF e CPF):

- travessia do projeto: interno ao processo DPF (corresponde a 1% da *feature*);
- projeto: interno ao processo DPF (corresponde a 40% da *feature*);
- inspeção de projeto: interno ao processo DPF (corresponde a 3% da *feature*);
- código: interno ao processo CPF (corresponde a 45% da *feature*);
- inspeção de código: interno ao processo CPF (corresponde a 10% da *feature*);
- promoção para construção: interno ao processo CPF (corresponde a 1% da *feature*).

Portanto, a metodologia FDD deve ser associada à característica Q do enunciado (neste ponto, já podemos adiantar que a alternativa B está correta, mas prosseguiremos com a resolução da questão).

DSDM (*Dynamic Systems Development Method* - Método Dinâmico de Desenvolvimento de Sistemas) é a metodologia progenitora do XP que funciona como um arcabouço para o Desenvolvimento Rápido de Aplicações (RAD). Tal metodologia, a fim de entregar softwares no tempo e custo estimados, controla e ajusta os requisitos ao longo do desenvolvimento, em outras palavras, ela fixa tempo e recursos ajustando a quantia de funcionalidades e controlando mudanças nos requisitos durante o ciclo de vida. Trata-se de uma metodologia de desenvolvimento iterativo e incremental, baseada em pequenas equipes, que enfatiza o envolvimento constante do usuário.

Antes de prosseguirmos, é preciso dizer que o termo RAD (*Rapid Application Development*) se aplica a projetos que têm prazos curtos, e que, em geral, envolvem o uso de prototipagem e ferramentas de desenvolvimento de alto nível.

Pronto, já temos condições de associar DSDM à característica R do enunciado.

Apresentamos, apenas a título de curiosidade, os nove princípios da DSDM:

- Envolvimento: o envolvimento do usuário é o ponto principal para eficiência e eficácia do projeto;
- Autonomia: a equipe deve estar empenhada em tomar decisões.
- Entregas: o foco na entrega frequente de produtos, assumindo que entregar algo bom logo é melhor que entregar algo perfeito somente no fim;
- Eficácia: o critério principal para ser considerado "entregável" é entregar um sistema que demonstre auxiliar nas necessidades e negócio atuais;
- *Feedback*: o desenvolvimento é iterativo e incremental controlado por feedbacks de usuários;
- Reversibilidade: todas as alterações feitas no desenvolvimento são reversíveis;
- Previsibilidade: o escopo e requisitos de alto nível devem ser definidos antes que o projeto se inicie;
- Ausência de testes no escopo: testes são tratados fora do ciclo de vida do projeto;
- Comunicação: É necessária excelente comunicação e cooperação de todos os envolvidos.

A Figura 3 apresenta os quatro estágios do ciclo de vida do projeto na DSDM.

ASD (*Adaptive Software Development*, em português: Desenvolvimento Adaptável de Software - DAS) é uma metodologia iterativa e incremental, que funciona bem em sistemas grandes e complexos. Nela, o cliente deve estar sempre presente. Podemos dizer que é uma metodologia voltada para o desenvolvimento de aplicações em conjunto (*Joint Application Development* - JAD).

A metodologia DAS possui um ciclo de vida contendo três fases (estas fases duram de 4 a 8 semanas):

- Especulação: fixa prazos e objetivos, além definir um plano baseado em componentes;
- Colaboração: construção concorrente de vários componentes;

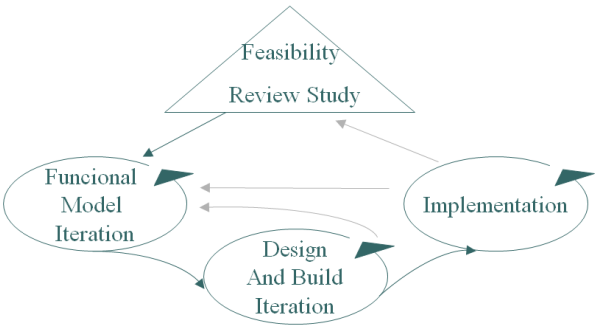


Figura 3: fases do ciclo de vida do projeto na DSDM.

- Aprendizado: repetitivas revisões de qualidade e foco na demonstração das funcionalidades desenvolvidas (*Learning loop*). Há presença do cliente de especialistas do domínio.

Já podemos associar, facilmente, o modelo DAS à característica P do enunciado.

Apenas para complementar o comentário, apresentamos seis propriedades do modelo DAS:

- orientado a missões (*misson-driven*): atividades são justificadas através de uma missão (que pode mudar ao longo do projeto);
- baseado em componentes (*component-based*): construir o sistema em pequenos pedaços;
- iterativo (*iterative*): foco em refazer do que fazer corretamente já na primeira vez;
- prazos pré-fixados (*time-boxed*): força os participantes do projeto a definir severamente decisões do projeto logo cedo.
- tolerância a mudanças (*change-tolerant*) as mudanças são freqüentes;
- orientado a riscos (*risk driver*): tens de alto risco são desenvolvidos primeiro;

Portanto, a alternativa **B** está correta.

13. **Assuntos relacionados:** *Engenharia de Software*, *Modelo de Processo de Software*, *Desenvolvimento Evolucionário*, *Transformação Formal*, *Desenvolvimento em Cascata*,
Banca: *ESAF*
Instituição: *Agência Nacional de Águas (ANA)*
Cargo: *Analista Administrativo - Tecnologia da Informação e Comunicação / Desenvolvimento de Sistemas e Administração de Banco de Dados*
Ano: *2009*
Questão: *8*

O modelo de processo de software caracterizado por intercalar as atividades de especificação, desenvolvimento e validação, denomina-se

(a). modelo de workflow.

(b). modelo de fluxo de dados.

(c). desenvolvimento evolucionário.

(d). transformação formal.

(e). modelo em cascata.

Solução:

(A) ERRADA

Um modelo de workflow é responsável por analisar e modelar todo o tramite de informações e atividades referentes a um processo de negócio de uma empresa para que possa ser definida uma forma organizada de execução das tarefas que compõem o processo. A automação do processo de negócio de uma empresa é o resultado final da modelagem de workflow, consistindo na interligação de tarefas executadas por participantes do processo ou até mesmo por sistemas computacionais definindo a ordem de execução e suas prioridades. Durante todo o desenvolvimento de um workflow várias etapas são executadas em uma determinada ordem. O ciclo de vida de um workflow (WFLC - Workflow Life Cycle) define a maneira como estas etapas são desenvolvidas em determinado tempo sobre todo o desenvolvimento do workflow. Para tanto, são definidos três atividades principais:

- **Construção e Estabilização**: este processo representa a fase de análise e modelagem do processo de negócio que a partir de uma linguagem gráfica é definido o Workflow, utilizando conceitos como processo, atividade, tarefa, ator, etc. Nesta fase é utilizado o processo de prototipagem para a detecção de erros de modelagem, por esse processo ser iterativo pode detectar e corrigir os erros de definição do modelo;
- **Detecção de Inconsistência**: nesta fase é gerada uma versão operacional referente à modelagem construída na fase anterior contendo todas as características e eventos do modelo. A versão é executada para que possa verificar o fluxo de informação que o modelo representa analisando informações de entrada do processo, recursos utilizados, participantes e resultado de saída do processo a fim de detectar inconsistência da análise da fase de construção do modelo. Ao encontrar anomalias no modelo o mesmo é submetido à estabilização novamente para melhorar a qualidade do Workflow;
- **Reengenharia de processo de Negócio**: os dados obtidos na fase anterior ao executar a versão operacional são usados a fim de verificar a adequação do modelo de processo de negócio para os objetivos da organização. Se este não for o caso, uma revisão do modelo do processo de negócio deve ser feita. O modelo atualizado é o ponto de partida de uma nova iteração dos processos de estabilização e depuração.

(B) ERRADA

O modelo de fluxo de dados é uma forma de representar os processos de um sistema, também, o fluxo das informações que entram nos processos e o resultado (saída) do processamento. Um fluxo de dados de um sistema pode ser visto como uma rede que interliga os processos mostrando a movimentação dos dados (entrada/saída). Este tipo de modelagem é importante para que o analista entenda todo o tramite das informações no sistema. A modelagem de fluxo de dados utiliza componentes para sua representação que são:

- **Processo**: é parte do sistema representa uma seqüência de operações (procedimento) que pode receber uma ou várias entradas, processar e retorna uma ou várias saídas;
- **Fluxo**: representa a movimentação dos dados, indicando onde as informações trafegam como entrada e onde são saídas. O Fluxo é representado por uma seta que indica a direção da movimentação dos dados;
- **Depósito**: Enquanto o fluxo representa as informações em movimento o depósito indica onde os dados estão armazenados (sem movimentação);
- **Entidade Externa**: É uma entidade (sistema externo) que se comunica com o sistema, a sistema externo pode ser uma pessoa, empresa, organização.

O modelo de fluxo de dado inicia em um nível de abstração alto e sofre processos de refinamento até chegar a um nível de detalhamento correspondente a sua necessidade. Para cada iteração de refinamento é criado um novo modelo atribuindo ao nome do documento um número correspondente a quantidade de iterações.

(C) CORRETA

O modelo evolucionário desenvolve todo o projeto de um sistema intercalando as suas atividades de especificação, desenvolvimento e validação. Este modelo trabalha com muitas versões dos sistemas que são apresentadas para o cliente e então refinadas, ou seja, as versões iniciais são construídas a partir de informações mais gerais e/ou abstratas e ao possa que são finalizadas e expostas para o cliente sofrem avaliações e melhoramentos. As atividades de desenvolvimento não são executadas separadamente para a construção do sistema, mas sim, concorrentemente para que ter uma resposta do cliente e um melhoramento/agregação das funcionalidades até chegar à versão final.

Há dois tipos de desenvolvimento evolucionário:

1. **Desenvolvimento exploratório**: tem como objetivo definir com o cliente os requisitos iniciais melhor compreendidos. O sistema sobre alterações à medida que o cliente detecta a necessidade de novas funcionalidades;
2. **Fazer protótipos descartáveis**: tem como objetivo refinar os requisitos dos sistemas baseando em protótipos para melhorar os requisitos que estão mal compreendidos.

(D) ERRADA

A transformação formal é um modelo de processo de software que utiliza de métodos formais para o desenvolvimento do software. Isto é, o processo de transformação formal executa várias etapas (atividades) que sequencialmente transformam os requisitos de software em um modelo matemático e posteriormente em um software. Este modelo tem como característica uma modelagem limpa, livre de erro. Isso é obtido através de modelos matemáticos que permitem uma verificação ou validação do modelo antes que o sistema seja implementado.

A primeira etapa é a definição dos requisitos juntamente com o cliente. Depois esta descrição do cliente é então detalhada saindo de um nível de abstração de usuário para o nível de especificação. A especificação é então transformada em um modelo formal detalhado (modelo matemático). Por fim, a implementação do modelo e teste utilizando alguma linguagem de programação.

(E) ERRADA

O modelo em cascata, também denominado clássico ou sequência linear, sugere uma abordagem sequencial para o desenvolvimento de software, que inicia pela análise e modelagem dos requisitos, projeto, codificação, teste até manutenção do sistema. A vantagem dessa abordagem seqüencial é visto no gerenciamento do projeto, uma vez que suas atividades possuem pontos de inicialização e finalização, assim, em principio esta modelagem é confiável e abrangem projeto de tamanhos variados. Por outro lado, por ter essa rigidez nas atividades de processo às etapas iniciais como levantamento de requisito e análise devem ser cuidadosamente implementadas, caso contrário, todas as outras etapas serão comprometidas.

As principais atividades de desenvolvimento são:

- **Análise**: as funcionalidades e as restrições do sistema são obtidas através de entrevistas com usuários do sistema. Posteriormente os requisitos são definidos e então é gerado um documento com a especificação das funcionalidade de forma detalhada;
- **Projeto de software**: O projeto de software reuni os requisitos (funcionais e não funcionais) em sistemas de hardware ou de software, para então, construir a arquitetura do sistema em um âmbito geral. O projeto de software envolve a identificação de todos os componentes e seus relacionamentos necessários para o desenvolvimento do sistema;
- **Implementação**: O projeto de software é construído gerando um comum conjunto de módulos (programas) ou um único de programas;
- **Testes de sistemas**: O sistema desenvolvido é submetido a testes que representam os requisitos desejados pelos usuários do sistema a fim de verificar erros e inconformidades do sistema;
- **Manutenção**: Depois dos testes o sistema é instado e colocado em funcionamento. A manutenção envolve corrigir erros que não foram descobertos na fase de testes e posteriormente de acordo com as necessidades dos clientes sofrer adições ou melhoramento de funcionalidades.

Concluindo, o único processo de software que intercala as atividades de especificação, desenvolvimento e validação produzindo em cada iteração uma nova versão do software é o modelo evolucionário de software. Portanto, a resposta correta é alternativa C.

14. **Assuntos relacionados:** *Engenharia de Software*, *Padrões de Projeto*, *Padrão Facade*,
Banca: *ESAF*
Instituição: *Controladoria-Geral da União (CGU)*
Cargo: *Analista de Finanças e Controle - Tecnologia da Informação / Desenvolvimento de Sistemas de Informação*
Ano: *2008*
Questão: *29*

Ao longo das últimas décadas, a engenharia de software fez progressos significativos no campo de padrões de projeto – arquiteturas comprovadas para construir software orientado a objetos flexível e fácil de manter. Com relação ao padrão Facade, é correto afirmar que

(a). fornece um objeto representante ou um marcador de outro objeto para controlar o acesso ao mesmo.

(b). define o esqueleto de um algoritmo em uma operação, postergando a definição de alguns passos para subclasses.

(c). define uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe a ser instanciada.

(d). fornece uma interface unificada para um conjunto de interfaces em um subsistema.

(e). define uma dependência "um para muitos" entre objetos, de modo que, quando um objeto muda de estado, todos os seus dependentes são automaticamente notificados e atualizados.

Solução:

A chave para maximizar o "reuso" reside em antecipar novos requisitos e novas mudanças relativos aos requisitos existentes, projetando sistemas que conseguem evoluir adequadamente.

Para projetar sistemas que sejam robustos em face a mudanças, deve-se considerar como o sistema pode precisar ser alterado em seu ciclo de vida. Um projeto que não considere este aspecto arrisca-se a ser "reprojetado" no futuro. Estas mudanças podem envolver redefinição e reimplementação de classes, modificações no cliente e reteste. O reprojeto afeta muitas partes do sistema de software e modificações não-antecipadas são invariavelmente caras.

Os padrões de projeto auxiliam o desenvolvedor a evitar problemas dessa natureza garantindo que um sistema pode ser modificado de determinadas maneiras. Cada padrão de projeto permite que alguns aspectos da estrutura do sistema variem independentemente de outras aspectos, tornando assim o sistema mais robusto a certos tipos de mudanças.

Estruturar um sistema em subsistemas auxilia na redução da complexidade. Um objetivo comum em padrões de projeto é minimizar as comunicações e as dependências entre os subsistemas. Uma forma de atingir tal objetivo é introduzir um objeto Facade que proporcione uma interface única e simplificada para as facilidades mais gerais de um subsistema.

Considerando o orientação a objetos (OO), os objetos podem variar tremendamente em tamanho e em número. Eles podem, em uma palavra, criar representações de tudo para o hardware subjacente e/ou para as aplicações. Decidir o que deve ser um objeto é uma tarefa delicada. O padrão de projeto Facade descreve como representar subsistemas completos como objetos, provendo uma interface unificada para um conjunto de interfaces em

um subsistema. Assim, ele define uma interface de alto-nível que torna um subsistema mais fácil de ser utilizado. Em geral, apenas um único objeto Facade é necessário.

A alternativa A não descreve uma característica do padrão de projeto Facade, pois, neste, não há representação de um objeto por outro, mas sim, de subsistemas inteiros por um objeto que atua como interface. O definição do objeto Facade é feita por completo, não havendo qualquer tarefa às classes do subsistema nesse tocante. Por este motivo, a afirmativa B está incorreta. Semelhantemente, após a definição desse objeto Facade, o mesmo deve ter ciência das classes que interfaceia para, no momento oportuno, instanciá-las adequadamente. A alternativa C não está correta por contradizer tal característica. Por fim, a assertiva E busca introduzir o conceito de dependência, inexistente no padrão de projeto em questão e, desta forma, não se alinha às características do Facade. Resta a alternativa D, que apresenta-se em acordo com a teoria exposta, sendo a resposta para a questão.

15. **Assuntos relacionados:** *Engenharia de Software*, *Modelo de Processo de Software*, *Desenvolvimento Formal*,
Banca: *ESAF*
Instituição: *Controladoria-Geral da União (CGU)*
Cargo: *Analista de Finanças e Controle - Tecnologia da Informação / Desenvolvimento de Sistemas de Informação*
Ano: *2008*
Questão: *42*

Um modelo de processo de software é uma representação abstrata de um processo de software. Assinale a opção que identifica um dos modelos de processo de software.

(a). Análise e definição de requisitos.

(b). Projeto de sistemas e de software.

(c). Análise e projeto estruturado.

(d). Análise e projeto orientado a objetos.

(e). Desenvolvimento formal de sistemas.

Solução:

(A) ERRADA

A análise e a definição de requisitos são etapas/atividades de processos de software que tem como objetivo identificar as necessidades (funcionalidades) que o sistema deve implementar, além de suas restrições. O ator responsável por essa atividade é o analista de sistema. O analista através de entrevistas, questionários e análise de documento identifica as funcionalidade juntamente com o cliente ou usuário final. Depois de reunidas as informações são avaliadas/interpretadas e organizadas em um Documento Conceitual do Sistema. A organização das informações implica na estruturação dos requisitos correspondentes o seu tipo (requisitos funcionais, requisitos não funcionais, requisitos de domínio).

(B) ERRADA

O projeto de software é o processo de transformar os documentos conceituais do sistema em uma especificação de uma solução. O projeto de software utiliza a especificação e os modelos de requisitos gerados na fase de análise para se definir o problema. Com os documentos referentes aos requisitos interpretados em mãos o projetista indicará soluções para o problema. A solução que satisfizer completamente os requisitos sobre as funcionalidades requeridas pelo cliente é construída.

(C) ERRADA

A análise estrutura, como todos os métodos de análise de requisitos de software, é uma atividade de construção de modelos. Usando uma notação que é própria para atividade de análise estruturada, são criados modelos que representam o fluxo e informações de domínio. O analista divide o sistema em uma parte que representa suas funcionais (comportamentais) e em uma parte que representa o domínio do sistema. Durante a análise estrutura são criadas representações gráficas e diagramas como o Diagrama de Fluxo de Dados e o Modelo Comportamental.

(D) ERRADA

A Análise Orientada a Objeto (AOO) baseia-se em conceitos de objetos, atributos, classes e membros. A AOO defini os objetos de um problema a partir dos atributos que descrevem um objeto para incluí-lo no modelo de análise. São os atributos que definem o objeto - que esclarecem aquilo que o objeto significa no contexto do problema.

O método de análise de requisitos de software OO (Orientado a Objeto) possibilita que o analista modele um problema ao representar classes, objetos, atributos e operações como os componentes da modelagem OO. O projeto orientado a objeto (POO) cria uma representação do domínio de problema do mundo real e modela um domínio de solução que é o software. O POO interliga objetos de dados e operações de processamento de uma forma que a informação e o processamento seja um único elemento (objeto), e não só o processamento com no projeto estruturado.

(E) CORRETA

O Desenvolvimento Formal de Sistema (DFS) abrange um conjunto de atividades que levam a especificação matemática formal de software de computador. O DFS especifica, desenvolve e executa verificações do sistema baseado na aplicação de uma rigorosa notação matemática. A construção do produto software é baseada na especificação matemática do sistema e podem ser verificado apresentado argumentos matemáticos e mostrando que eles atendem a suas especificações. Apesar de não vir a ser uma abordagem de uso geral, o modelo de transformação formal oferece a promessa de software livres de defeitos.

Atividades envolvida no processo de software desenvolvimento formal são:

- **Definição dos requisitos**: definição das funcionalidades desejadas;
- **Especificação formal**: detalhamento dos requisitos funcionais;
- **Transformação formal**: mapeamento dos requisitos (detalhados) em um modelo matemático;
- **Integração e teste de Sistema**: implementação do modelo e teste utilizando alguma linguagem de programação.

Esse processo de software é geralmente aplicado a sistema que possuem requisitos críticos de segurança, isso porque o sistema deve ser livre de falhas. Isso é obtido pela modelagem matemática.

Concluindo, todos os itens descritos acima, análise e definição de requisitos, análise estruturada e orientada a objeto e projeto de software são elementos da modelagem de processo de software. O único processo de software descrito acima é o Desenvolvimento Formal de Sistema. Logo, a questão correta é a letra E.

16. **Assuntos relacionados:** *Engenharia de Software*, *Modelo de Processo de Software*, *Modelo Espiral*,
Banca: *ESAF*
Instituição: *Controladoria-Geral da União (CGU)*
Cargo: *Analista de Finanças e Controle - Tecnologia da Informação / Desenvolvimento de Sistemas de Informação*
Ano: *2008*
Questão: *44*

No modelo de desenvolvimento em espiral, cada ciclo da espiral representa uma fase do processo de software. Nesse modelo, a atividade que obrigatoriamente estará presente em todos os ciclos é:

(a). Planejamento de desenvolvimento.

(b). Análise de requisitos.

(c). Teste de unidade.

(d). Análise, Projeto, Implementação e Teste.

(e). Análise de riscos.

Solução:

O modelo de desenvolvimento em espiral é um processo de software que agrupa as melhores características dos modelos de software cascata e prototipação. As propriedades combinadas são o controle sequencial – para descrever o fluxo organizado de atividades que são executadas e a características de iteratividade que o modelo de prototipação trabalha – iteração com o cliente para o refinamento dos requisitos e correção de requisitos mal interpretados. Além dessas características, o modelo espiral adiciona uma nova atividade – análise dos riscos – que é exigida em todos os estágios (ciclos) do projeto. Este novo elemento se executado corretamente proporciona o desenvolvimento do projeto sem a ocorrência de fortes impactos sobre seu fluxo de execução do projeto com situações problemáticas. A partir destes elementos, o modelo espiral obtém o desenvolvimento rápido de novas versões do software/sistema partindo de um conjunto de funcionalidades básicas até se tornar um software completo.

Para tanto, o modelo espiral segue um conjunto de 6 atividades como descrito e apresentado na Figura 4.

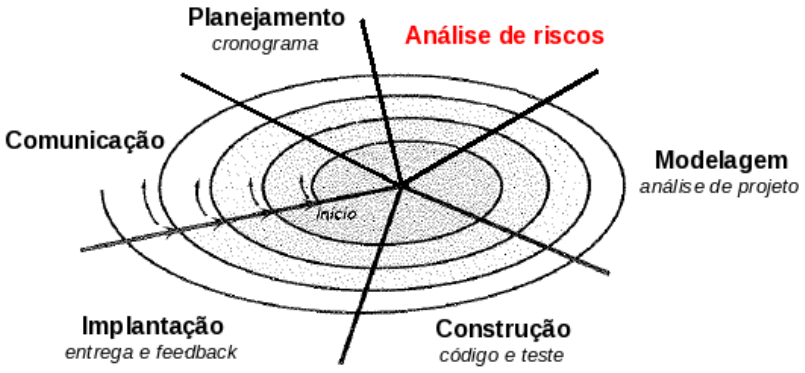


Figura 4: representação do modelo de desenvolvimento em espiral.

- **Planejamento**: determinação dos objetivos, alternativas e restrições;
- **Análise de riscos**: análise de alternativas e identificação/resolução dos riscos;
- **Modelagem**: modelagem das funcionalidades que serão adicionadas nesse estágio (ciclo) do projeto;
- **Construção**: implementação do sistema em um linguagem de programação resultando numa versão de software e aplicação de testes sobre essa nova versão;
- **Implantação**: instalação da versão corrente de um determinado estágio do projeto e retorno da satisfação do cliente;
- **Comunicação**: avaliação dos resultados da engenharia.

O modelo espiral é um método de desenvolvimento de sistemas e software de grande porte. Como o software evoluí à medida que as atividades avançam, o desenvolvedor e o cliente refinam melhor os requisitos e reagem aos riscos de cada nível evolucionário.

Concluindo, para que este processo de software tenha sucesso ao finalizar um projeto é necessário que em cada iteração ou nível evolucionário a etapa de análise de riscos deve ser executada. Portanto, a questão correta é a letra E.

17. **Assuntos relacionados:** *Engenharia de Software*, *Projeto de Interface com Usuário*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Processos de Negócio*
Ano: *2008*
Questão: *60*

Assinale a opção que **NÃO** expressa um princípio de projeto de interface com o usuário.

(a). Reduzir a demanda de memória de curto prazo do usuário.

(b). Basear o layout visual em uma metáfora do mundo real.

(c). Permitir que a interação com o usuário seja interruptível e possa ser desfeita (undo).

(d). Estabelecer defaults (para escolhas e preenchimento de formulários) que façam sentido para o usuário.

(e). Mostrar informações completas a *priori*, permitindo que o usuário reduza o nível de detalhe se desejar.

Solução:

O termo interface é aplicado normalmente àquilo que interliga dois sistemas. Tradicionalmente, considera-se que uma interface homem-máquina é a parte de um artefato que permite a um usuário controlar e avaliar o funcionamento deste artefato através de dispositivos sensíveis às suas ações e capazes de estimular sua percepção. No processo de interação usuário-sistema a interface é o combinado de software e hardware necessário para viabilizar e facilitar os processos de comunicação entre o usuário e a aplicação.

A interface entre usuários e sistemas computacionais diferencia-se das interfaces de máquinas convencionais por exigir dos usuários um maior esforço cognitivo em atividades de interpretação e expressão das informações que o sistema processa. A interface de usuário deve ser entendida como sendo a parte de um sistema computacional com a qual uma pessoa entra em contato físico, perceptiva e conceitualmente.

A interface possui componentes de software e hardware. Os componentes de hardware compreendem os dispositivos com os quais os usuários realizam as atividades motoras e perceptivas. Entre eles estão à tela, o teclado, o mouse e vários outros. O software da interface é a parte do sistema que implementa os processos computacionais necessários para controle dos dispositivos de hardware, para a construção dos dispositivos virtuais com os quais o usuário também pode interagir, para a geração dos diversos símbolos e mensagens que representam as informações do sistema, e finalmente para a interpretação dos comandos dos usuários.

Os seguintes princípios devem ser seguidos durante o projeto da interface com o usuário para se obter um produto final usável:

Minimizar a Carga de Memória do Usuário

O usuário não deve ser forçado a memorizar informações ao passar de uma parte do diálogo a outra. Em geral, as pessoas são muito melhores em reconhecer algo que lhes é mostrado do que em recuperar a mesma informação da memória sem nenhuma ajuda.

É mais fácil para o usuário modificar informações fornecidas pelo computador do que gerar o resultado desejado partindo do zero. Por exemplo, quando usuários querem renomear um arquivo, é razoável esperar que o novo nome seja próximo do nome antigo na maioria dos casos, logo a caixa de texto em que a leitura do novo nome será feita deve trazer o nome antigo como default. Quando os usuários devem prover uma entrada com um formato especial, o sistema deve descrever o formato esperado (caracteres separadores, faixa de valores válidos) e, se possível, prover um exemplo válido como default. Para minimizar a carga de memória do usuário, o sistema deve-se basear em um número reduzido de regras que se apliquem a várias situações dependendo do contexto. O uso de comandos genéricos como "Cortar" e "Colar" tanto para pedaços de texto quanto para elementos gráficos é um bom exemplo desta técnica.

Metáfora do Mundo Real

Os diálogos devem ser expressos claramente em palavras, expressões e conceitos familiares à comunidade de usuários, não em termos orientados ao sistema. Por exemplo, um aplicativo financeiro não deve exigir que usuários especifiquem o tipo de moeda com códigos como 312 para dólares e 213 para reais, mas sim com termos como "Dólares" ou "Reais". As interações devem ser vistas da perspectiva do usuário. Por exemplo, o diálogo de confirmação de uma transação comercial deveria ser algo como "Você está adquirindo 29 CD's" e não "Nós estamos lhe vendendo 29 CD's".

As metáforas usadas pela interface com o usuário devem respeitar o conceito que o usuário tem da metáfora. Por exemplo, a metáfora "um processador de textos é como uma máquina de escrever" ajudará os usuários a descobrir funcionalidades como retrocesso e tabulação, mas não os dará nenhuma pista sobre uma funcionalidade de substituição global de frases. Se a metáfora do sistema não for exatamente igual à metáfora do mundo real, este fato deve ficar claro para o usuário.

Opções defaults

A interface deve proporcionar uma adequada proteção aos dados do usuário, através da definição de opções de comando default e na apresentação antecipada de avisos sobre o resultado de ações do usuário, e da solicitação de confirmação de suas ações. Uma interface deste tipo apresenta valores default (para os campos de dados, lista, *check boxes*) capaz de acelerar as entradas individuais e fornece o preenchimento automático de vírgulas, pontos decimais e zeros à direita da vírgula nos campos de dados. Uma interface ágil não solicita aos usuários dados que podem ser deduzidos pelo sistema; não força o usuário a percorrer em seqüência todas as páginas de um documento de modo a alcançar a página específica e não solicita o mesmo dado ao usuário diversas vezes em uma mesma seqüência de diálogo.

Controle explícito pelo usuário

Refere-se tanto ao processamento explícito pelo sistema das ações do usuário quanto ao controle que este mantém sobre o processamento de seus passos pelo sistema. Os indivíduos devem, declaradamente, controlar entradas e saídas de dados, pois esta providência diminui os erros e as ambigüidades e, na medida em que mantêm controle sobre o diálogo, tendem a aceitar melhor o sistema. Podemos inferir que o controle explícito trata das relações entre processamento pelo computador e ações do usuário, com o lembrete de que essa inter-relação

deve ser explícita, vez que o computador deve processar somente as demandas explicitadas e somente quando requisitadas. Logo, os comandos do usuário devem ser seguidos de Enter depois de editados ou de click no mouse, se o ambiente é orientado a evento e a objeto. O cursor não deve ser automaticamente movido sem o controle do usuário, salvo para procedimentos estáveis e consolidados, como preenchimento de formulários. É recomendado, ainda, a chance de o indivíduo interromper ou cancelar a transação mediante a opção cancelar ou desfazer, que permite apagar ou retomar qualquer mudança recente. A opção de desfazer é conhecido como reversibilidade que é possibilidade de desfazer operações errôneas efetuadas pelo usuário.

Desta forma, deixar informações completas para o usuário mesmo com o princípio de controle explícito permitindo o mesmo reduzir a quantidade de informações não leva a uma boa prática de projeto de interface com usuário porque informações excessivas poluem o campo visual. Logo a alternativa **E** está incorreta.

18. **Assuntos relacionados:** *Interface com Usuário*, *Usabilidade*,
Banca: *ESAF*
Instituição: *Controladoria-Geral da União (CGU)*
Cargo: *Analista de Finanças e Controle - Tecnologia da Informação / Desenvolvimento de Sistemas de Informação*
Ano: *2008*
Questão: *40*

Como características de usabilidade ou facilidade de uso, uma interface com o usuário deve possuir, entre outros, atributos tais como: facilidade de aprendizado, velocidade de operação, robustez, facilidade de recuperação e facilidade de adaptação. Para o atributo robustez, o resultado da avaliação de uma interface deve determinar

(a). o nível de tolerância do sistema aos erros do usuário.

(b). até que ponto o sistema está integrado a um único modelo de trabalho.

(c). quanto tempo leva um novo usuário para se tornar produtivo com o sistema.

(d). com que eficiência o sistema se recupera a partir dos erros cometidos pelos usuários.

(e). em que grau a resposta do sistema combina com a prática de trabalho do usuário.

Solução:

Geralmente, os usuários avaliam um sistema pela sua interface (responsável pela interação do usuário com o computador) e não pela sua funcionalidade. Para você ter uma idéia, a interface pode determinar o sucesso ou o fracasso da utilização de sistema, por isso é extremamente importante que os engenheiros de softwares estejam preparados para projetar uma interface com o usuário.

Uma boa abordagem que os engenheiros de software adotam é fazer com que o projeto de interface seja centrado no usuário, onde o fator de sucesso é proveniente da análise das atividades do(s) usuário(s). A Figura 5 exemplifica o processo de projeto de interface (note a existência de vários).

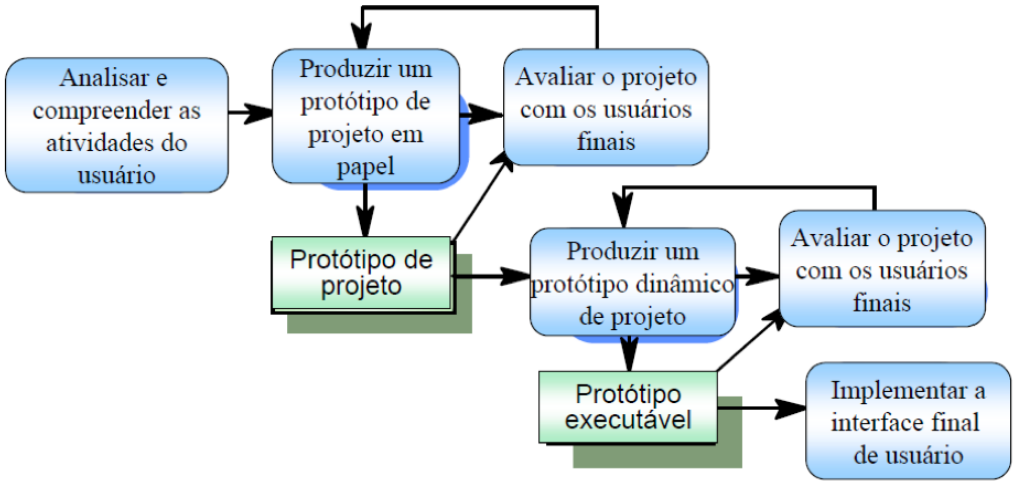


Figura 5: processo de projeto de interface com usuário.

E o que seria usabilidade no escopo de interface com o usuário?

Saiba que quando você classifica um sistema que está utilizando em fácil ou difícil, você está, na verdade, analisando a usabilidade do programa. Quando o usuário precisa de uma ferramenta, geralmente ele procura aquela que atenda a suas necessidades e que não lhe traga problemas durante o seu trabalho, ou seja, ele procura um produto com um bom nível de usabilidade, e para se obter esse nível de usabilidade é necessário levar em consideração as características acima apresentadas.

Segundo a norma ISO CD 9241: *"usabilidade está relacionada com a forma como um produto pode ser usado por um grupo de usuários para alcançar um determinado conjunto de objetivos com eficácia (precisão e completeza com que os usuários atingem objetivos específicos), eficiência (precisão e completeza dos objetivos atingidos em relação aos recursos despendidos) e satisfação (conforto e aceitabilidade na utilização do sistema) de uso em um determinado contexto de aplicação."*

Já segundo Nielsen (2000): *"usabilidade é o conjunto de todas as características que permitem ao usuário interagir com o computador satisfatoriamente."*

Achou meio vaga a definição de Nielsen, né? Realmente, "todas as características" não ajudou muito. Bom, para facilitar, nós apresentaremos as 10 características por ele consideradas as mais importantes:

- **facilidade de aprendizado**: fornece condições que permitem que o usuário aprenda a interagir com o sistema de forma natural, independentemente de seu nível de habilidade e conhecimento. Desta forma, a interface deve se apresentar de acordo com o que é mais apropriado ao comportamento humano. Descartamos aqui a alternativa C;
- **facilidade de utilização (adaptação)**: está intrinsecamente relacionada à flexibilidade do sistema, isto é, à capacidade de adaptação ao nível de conhecimento e habilidade dos usuários do sistema. Por exemplo, alguns aspectos relacionados à flexibilidade são: iniciação do diálogo, o multi-threading, a possibilidade de migração de tarefas, e a adaptabilidade. Já podemos descartar as alternativas B e E;
- **interface intuitiva**: os comandos para a execução de tarefas devem estar claramente visíveis, a fim de se evitar a necessidade de memorização de funções (não confunda com a memorização de atalhos, que é opcional para a realização de tarefas);
- **diálogo simples e natural**: uso de expressões e conceitos que façam parte do conhecimento do usuário. Portanto, evite o emprego de termos técnicos da computação, assim como, informações irrelevantes competem com o que realmente é necessário para a realização da tarefa em questão;
- **feedback ao usuário**: a interface dever oferecer mecanismo para informar ao usuário como o sistema está se comportando internamente, de forma que o usuário não fique. Por exemplo, informar o tempo restante para a conclusão da tarefa requisitada;
- **retenção de conhecimento**: quanto tempo um usuário pode ficar afastado do sistema (sem interação) e ainda assim lembrar dos principais comandos disponíveis, sem necessitar consultar manuais de operação?
- **velocidade na execução das tarefas**: quanto maior a velocidade de processamento da máquina utilizada e a performance dos algoritmos, maior será a usabilidade do sistema;
- **satisfação subjetiva**: como o usuário sente-se ao utilizar o sistema? Por exemplo, o grau de satisfação do usuário estará diretamente relacionado à diversão proporcionada por um sistema de entretenimento;

- **mensagens de erros consistentes**: mensagem que realmente informem ao usuário o que foi feito errado, onde está o erro e ofereça a possibilidade de corrigi-lo;
- **robustez**: em poucas palavras é a preparação do sistema para evitar/tolerar erros do usuário. O desenvolvimento da interface deve ser um processo criterioso a fim de minimizar todas as possibilidades de erro. Como é praticamente impossível prever todas as ações do usuário que podem ocasionar erros, torna-se muito importante que o sistema possua mensagens de erro consistentes. Para um sistema ser robusto ele deve atender o princípio da:
- **observabilidade**: habilidade do usuário avaliar o estado interno do sistema referente a sua representação perceptível, por exemplo, a navegabilidade;
- **possibilidade de recuperação**: habilidade do usuário tomar ação corretiva uma vez que um erro seja reconhecido, como, por exemplo, "Undo" e "Lixeira" no caso de arquivos removidos acidentalmente. Note que robustez não é somente possibilidade de recuperação, por isso descartamos da alternativa D;
- **compreensibilidade**: como o usuário percebe a relação de comunicação com o sistema, isto é, o grau de estabilidade;
- **conformidade de tarefas**: qualidade com que um serviço do sistema dá suporte a todas as tarefas que o usuário deseja realizar;

Do exposto acima, podemos concluir facilmente que a alternativa A está CORRETA.

Candidato, se você ainda estiver se perguntando o que levaria uma empresa a investir em usabilidade, recomendamos que leia o restante do comentário.

É preciso saber que a aceitabilidade de um produto está divida entre a aceitabilidade social e a aceitabilidade prática. A aceitabilidade social diz respeito à quão bem recebido um produto será em uma sociedade, o que é fundamental na determinação do sucesso ou do fracasso de um determinado produto, isso porque, quanto melhor aceito for um produto socialmente, maior será o número de pessoas que o adquirirá. Entretanto somente a aceitabilidade social não é suficiente para sua aquisição. Sem a aceitabilidade prática, de nada vale a aceitabilidade social.

A aceitabilidade prática relaciona-se a fatores tais como a utilidade, o custo, a compatibilidade e a confiabilidade. O fator utilidade, por sua vez, relaciona-se à funcionalidade oferecida pelo produto, assim como a usabilidade que esse produto oferece. Dessa forma, pode-se perceber que sem usabilidade a aceitabilidade do sistema é abalada, pois o produto, embora possivelmente ofereça diversas funcionalidades, não oferece uma forma fácil de acessá-la, o que pode torná-lo inútil. Sendo o produto inútil, a aceitabilidade prática não se efetua e, por sua vez, a aceitabilidade do sistema também não.

Portanto, aqueles que não investem em usabilidade correm riscos consideráveis, como o de perder na comparação com um outro produto submetido às práticas de usabilidade. Além disso, quando não se investe em usabilidade é quase certo que o produto apresentado tenha que retornar aos laboratórios de desenvolvimento, para corrigir erros que poderiam ter sido detectados através da prática dos métodos de usabilidade.

19. **Assuntos relacionados:** *Métricas de Software*, *Pontos de Função*,
Banca: *FCC*
Instituição: *TCE/CE*
Cargo: *Analista de Controle Externo - Auditoria de Tecnologia da Informação*
Ano: *2008*
Questão: *79*

Considere que 20 é o nível de influência global (ajuste fino total) aplicado em um cálculo de pontos por função. Então, o número de pontos por função ajustado (PFA) em relação ao bruto (PFB) é dado por

(a). $PFA = PFB \times 0,15$.

(b). $PFA = PFB \times 0,20$.

(c). $PFA = PFB \times 0,45$.

(d). $PFA = PFB \times 0,85$.

(e). $PFA = PFB \times 1,20$.

Solução:

Segundo Hazan, a Análise de Pontos de Função (APF) é um método para a medição do desenvolvimento de software, visando estabelecer uma medida de tamanho do software em Pontos de Função (PFs), com base na funcionalidade a ser implementada, sob o ponto de vista do usuário. Os objetivos da APF são:

- medir as funcionalidades do sistema requisitadas e recebidas pelo usuário;
- medir projetos de desenvolvimento e manutenção de software, sem se preocupar com a tecnologia que será utilizada na implementação.

O procedimento para cálculo dos pontos de função envolve os seguintes passos:

1. *Determinar o tipo de contagem de pontos de função*:
Existem três tipos de contagem: contagem de PF de projeto de desenvolvimento, de aplicações instaladas e de projetos de manutenção.
2. *Identificar o escopo de contagem e a fronteira da aplicação*:
Neste passo, definem-se as funcionalidades que serão incluídas em na contagem. A fronteira da aplicação é definida estabelecendo um limite lógico entre a aplicação que está sendo medida, o usuário e outras aplicações. O escopo de contagem define a parte do sistema (funcionalidades) a ser contada.
3. *Determinar a contagem de pontos de função não ajustados*:
Os pontos de função não ajustados (PFNA), também chamados de pontos de função brutos (PFB), refletem as funcionalidades fornecidas pelo sistema para o usuário. Essa contagem leva em conta dois tipos de função: de dados e transacionais, bem como sua complexidade (simples, média ou complexa).
4. *Contagem das funções de dados*:
As funções de dados representam as funcionalidades relativas aos requisitos de dados internos e externos à aplicação. São elas os arquivos lógicos internos (ALIs) e os arquivos de interface externa (AIEs). Um ALI é mantido dentro da fronteira da aplicação, enquanto um AIE é apenas referenciado pela aplicação, sendo mantido dentro da fronteira de outra aplicação.

5. *Contagem das funções transacionais*:
   As funções transacionais representam as funcionalidades de processamento de dados do sistema fornecidas para o usuário. São elas: as entradas externas (EEs), as saídas externas (SEs) e as consultas externas (CEs). As EEs são processos elementares que processam dados que entram pela fronteira da aplicação. As SEs são processos elementares que enviam dados para fora da fronteira da aplicação. Seu objetivo é mostrar informações recuperadas através de um processamento lógico (isto é, que envolva cálculos ou criação de dados derivados) e não apenas uma simples recuperação de dados. Uma SE pode, também, manter um ALI ou alterar o comportamento do sistema. Por fim, uma CE, assim como uma SE, é um processo elementar que envia dados para fora da fronteira da aplicação, mas sem realização de nenhum cálculo nem a criação de dados derivados. Seu objetivo é apresentar informação para o usuário, por meio apenas de uma recuperação das informações. Nenhum ALI é mantido durante sua realização, nem o comportamento do sistema é alterado.
6. *Determinar o valor do fator de ajuste*:
   O fator de ajuste é baseado em 14 características gerais de sistemas, que avaliam a funcionalidade geral da aplicação que está sendo contada, e seus níveis de influência. O nível de influência de uma característica é determinado com base em uma escala de 0 (nenhuma influência) a 5 (forte influência).
7. *Calcular os pontos de função ajustados*:
   Finalmente, os PFs ajustados são calculados, considerando-se o tipo de contagem definido no primeiro passo e o valor do fator de ajuste.

O fator de ajuste influencia os pontos de função não ajustados em +/- 35%, obtendo-se o número de PFs ajustados. Para se calcular o fator de ajuste, são usadas 14 características gerais dos sistemas, a saber:

1. Comunicação de Dados;
2. Processamento de Dados Distribuído;
3. Desempenho;
4. Utilização do Equipamento (Restrições de Recursos Computacionais);
5. Volume de Transações;
6. Entrada de Dados On-line;
7. Eficiência do Usuário Final (Usabilidade);
8. Atualização On-line;
9. Processamento Complexo;
10. Reusabilidade;
11. Facilidade de Implantação;
12. Facilidade Operacional (Processos Operacionais, tais como Inicialização, Cópia de Segurança, Recuperação etc);
13. Múltiplos Locais e Organizações do Usuário;
14. Facilidade de Mudanças (Manutenibilidade).

Para cada uma dessas 14 características deve-se atribuir um valor para o grau (ou nível) de influência de 0 (nenhuma influência) a 5 (forte influência). O grau de influência indica o quanto determinada característica tem influência no sistema. Os 14 graus de influência (GIs) informados são somados, resultando no nível de influência total (NIT). O valor do fator de ajuste (VFA) é determinado pela fórmula:

$$VFA = (NIT * 0,01) + 0,65 \quad (1)$$

Portanto, os pontos de função ajustados são dados pela fórmula:

$$PFA = PFB * VFA \quad (2)$$

No caso da questão, o valor do NIT é 20. Portanto, $VFA = (20 * 0,01) + 0,65 = 0,85$. Logo, $PFA = PFB * 0,85$, e a resposta da questão é a alternativa D.

20. **Assuntos relacionados:** *Engenharia de Software*, *Análise de Pontos de Função*, *Funções Transacionais*,
Banca: *Cesgranrio*
Instituição: *BNDES*
Cargo: *Analista de Sistemas - Desenvolvimento*
Ano: *2008*
Questão: *69*

O sistema de cadastro de eventos de uma empresa de consultoria em TI dispõe de uma tela que lista as palestras gratuitas realizadas no mês, ordenadas por dia, com totalização. No contexto de Análise de Pontos de Função, essa tela do sistema é contada como

(a). Consulta Externa (CE), pois não há dados derivados.

(b). Consulta Externa (CE), pois há totalização de dados.

(c). Arquivo Lógico Interno (ALI), já que os dados foram extraídos de um arquivo referenciado.

(d). Saída Externa (SE), pois há dados derivados.

(e). Entrada Externa (EE), já que existe mudança de comportamento do sistema.

Solução:

A Análise de Pontos de Função (APF) é um método padrão para medir o tamanho de um software, em Pontos de Função (PFs), com base na funcionalidade a ser implementada, sob ponto de vista do usuário.

O Ponto de Função é uma unidade de medida, que tem por objetivo tornar a medição independente da tecnologia (ferramentas, linguagens de programação, métodos, etc) empregada na construção do software. Ou seja, os PFs medem o tamanho do que o software faz, ao invés de como ele é desenvolvido e implementado.

Os objetivos da APF são: medir as funcionalidades do sistema requisitadas e recebidas pelo usuário; e medir projetos de desenvolvimento e manutenção de software.

O procedimento para contagem de PFs compreende 5 etapas, como mostrado na Figura 6.

A Etapa I do processo de contagem (Determinar o Tipo de Contagem) consiste na identificação do objetivo a ser medido (qual o tipo de contagem), sendo que existem três: projeto de desenvolvimento, projeto de manutenção e de aplicação. Os PFs de projeto de desenvolvimento estão associados com a instalação inicial de um software novo (primeira instalação). Os PFs de projetos de manutenção estão associados com a melhoria de um software já existente (inclui funcionalidade que é adicionada, modificada ou excluída). E, os PFs de aplicação estão associados com uma aplicação já instalada (sistemas liberados para o usuário final, que se encontram em plena utilização).

A Etapa II (Identificar o Escopo de Contagem e Fronteira da Aplicação) consiste em definir o escopo de contagem do sistema, isto é, as funcionalidades que serão incluídas na contagem de PFs, sob o ponto de vista do usuário. A fronteira da aplicação é definida estabelecendo um limite lógico entre a aplicação que está sendo medida, o usuário e outras aplicações. Para isso, todos os relacionamentos do sistema com o exterior, todas as pertinências dos dados e os processos suportados pelo sistema que está sendo contado são identificados. A

fronteira entre aplicações é definida com base no ponto de vista do usuário e com base na funcionalidade separada do negócio e, não na implementação tecnológica.

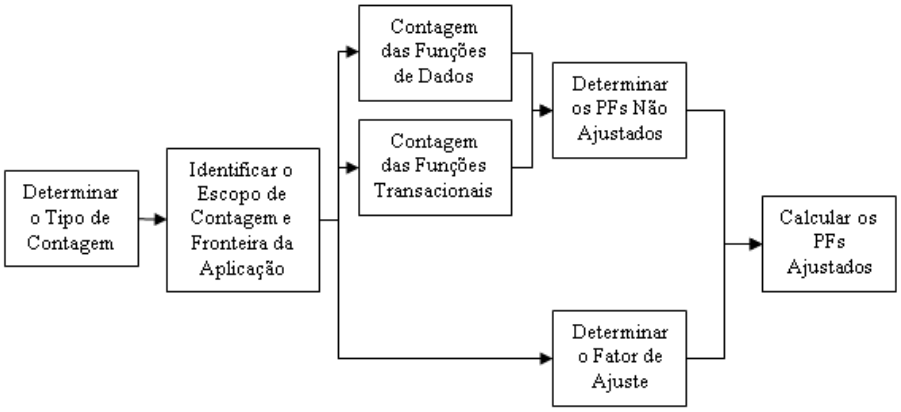


Figura 6: etapas da contagem de pontos de função.

A contagem de pontos de função não ajustados, Etapa III (Determinar os PFs não Ajustados), reflete o conjunto de funções disponibilizadas ao usuário, e o resultado da contagem pode ser considerado como pontos de função brutos, face à necessidade de se observar outras variáveis que influenciam nos cuidados e esforços a serem despendidos durante o processo de desenvolvimento do sistema. Esta contagem leva em conta dois tipos de função: de dados e transacionais, bem como a complexidade de cada uma, que pode ser simples, média ou complexa.

As funções de dados representam as funcionalidades relacionadas aos requisitos de dados internos e externos à aplicação, e são classificadas em Arquivo Lógico Interno (ALI) e Arquivo de Interface Externa (AIE). O ALI trata grupo de dados ou informações de controle (dado utilizado pelo sistema para garantir que todas as funções sejam realizadas conforme solicitado pelo usuário), requisitados pelo usuário como necessidades de informação, cuja manutenção, realizada por um processo da aplicação (alteração, inclusão, exclusão), acontece dentro da fronteira da aplicação. Um ALI equivale a um depósito de dados no Diagrama de Fluxo de Dados (DFD) ou a uma entidade no Modelo Entidade Relacionamento (MER). O AIE são grupos de dados necessários para a aplicação, mantidos e armazenados fora do sistema que está sendo dimensionado. Baseia-se na identificação dos dados armazenados fora da fronteira da aplicação, dados que não sofrem manutenções pela aplicação que está sendo avaliada. Isso significa que um AIE contado em uma aplicação, deve ser um ALI em outra aplicação. A complexidade do ALI e AIE é determinada pela quantidade de registros lógicos e itens de dados referenciados.

As funções transacionais representam as funcionalidades providas ao usuário para processamento de dados por uma aplicação, e são classificadas em Entradas Externas (EE), Saídas Externas (SE) e Consultas Externas (CE). As EE são grupos de dados que entram pela fronteira da aplicação, utilizados para a manutenção dos ALI, ou seja, provocam uma inclusão, exclusão e/ou alteração nos dados dos ALI. A intenção primária de uma EE é manter um ou mais ALI e/ou alterar o comportamento do sistema. As SE representam as atividades do sistema que transformam dados dos arquivos lógicos internos e geram resultados que são enviados para fora da fronteira da aplicação. A intenção primária de um SE é apresentar informações ao usuário através de processamento lógico, além da recuperação de dados e

informação de controle. O processamento lógico deve conter pelo menos uma fórmula matemática ou cálculo, ou criar dados derivados. Uma SE também pode manter um ou ALIs e/ou alterar o comportamento de um ou mais ALIs e/ou alterar o comportamento do sistema. Por fim, as CE são requisições de informações que, para serem satisfeitas, precisam que sejam combinados parâmetros de entradas e saídas que permitem a recuperação da informação solicitada pelo usuário. Assim como na SE, as CE enviam dados para fora da fronteira da aplicação, mas sem realização de nenhum cálculo ou criação de dados derivados. Nenhum arquivo lógico interno é mantido durante o processo e nem o comportamento do sistema é alterado.

A próxima etapa, Etapa IV (Determinar o Fator de Ajuste), considera que outros fatores afetam o tamanho funcional de um sistema. O fator de ajuste é baseado em 14 características gerais de sistemas, que avaliam a funcionalidade geral da aplicação que está sendo contada, e seus níveis de influência. O nível de influência de uma característica é determinado com base em uma escala de 0 (nenhuma influência) a 5 (forte influência). As características gerais do sistema são: comunicação de dados, processamento de dados distribuídos (funções distribuídas), performance, configuração do equipamento, volume de transações, entrada de dados on-line, interface com usuário, atualização on-line, processamento complexo, reusabilidade, facilidade de implementação, facilidade operacional, múltiplos locais e facilidade de mudanças. Então, cada uma dessas características é classificada de acordo com seu nível de influência.

A última etapa, Etapa V (Calcular os PFs Ajustados), trata do processo que realiza a correção das possíveis distorções cometidas durante o cálculo dos pontos de função não ajustados, aproximando as medidas à situação real. Os PFs ajustados são calculados, considerando-se o tipo de contagem definido na primeira etapa.

O enunciado desta questão quer saber se a funcionalidade do sistema de listar as palestras gratuitas realizadas no mês, ordenadas por dia, com totalização se trata de uma Função de Dados ou de uma Função Transacional. Esta funcionalidade envia dados para fora da fronteira da aplicação, o que pode ser caracterizado como uma SE ou CE. Entretanto, esta funcionalidade, ao recuperar a informação de um ALI no sistema, realiza um processamento lógico (ordenação) e cria dados derivados (totalização), antes de apresentar a informação ao usuário. Isso caracteriza esta funcionalidade como uma SE. Portanto, a alternativa correta é a letra (D).

21. **Assuntos relacionados:** *Engenharia de Software*, *Análise de Ponto de Teste*, *TPA*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas Pleno - Processos*
Ano: *2006*
Questão: *51*

Uma das métricas de teste mais utilizadas no mercado é a Análise de Ponto de Teste (TPA - *Test Point Analysis*). Sobre a TPA é **FALSO** afirmar que:

(a). pode ser usada para preparar uma estimativa do esforço de teste para os testes de sistema ou de aceitação e cobre somente os chamados testes de caixa-preta.

(b). os pontos de teste dinâmicos levam em consideração o sistema como um todo, diferentemente dos pontos de teste estáticos, que consideram cada uma das características isoladamente. Os pontos de teste estáticos e dinâmicos somente devem ser considerados quando a equipe de testes adotar processos de revisão de documentação e de códigos usando checklists, caso contrário deve ser descartado e terá valor nulo.

(c). a qualificação da equipe de teste (fator de produtividade da equipe de teste) é determinada por uma base histórica e, na prática, varia entre 0,7 e 2,0, sendo que quanto melhor e mais qualificada for a equipe tanto menor será o fator.

(d). o total de horas primárias calculada através da métrica TPA pode ser divida entre quatro fases de teste, sendo a seguinte a divisão percentual das horas entre as fases: preparação, 10%; especificação, 40%; execução, 45%; transição (completion), 5%.

(e). uma das variáveis utilizadas para calcular a influência do ambiente de teste no resultado do total de horas primária é o ambiente de desenvolvimento que leva em consideração fatores como, por exemplo, se o sistema foi desenvolvido utilizando uma linguagem 4GL integrada ao banco de dados ou uma combinação de linguagem 4GL e 3GL.

Solução:

Após a implementação de um programa, e antes da entregá-lo ao cliente, o mesmo deve ser testado a fim de se verificar a presença de defeitos, aumentando a confiabilidade de que está correto e de acordo com as especificações. Os testes de software devem ser planejados antes de serem realizados, e quando realizados devem utilizar uma quantidade mínima de esforços.

Existem diversas maneiras de testar um programa, sendo que as principais técnicas são: teste de caixa-branca e teste de caixa-preta. Os testes de caixa-branca ou testes estruturais avaliam diretamente o código fonte do programa, onde aspectos como teste de caminho lógico, teste de condição e teste de fluxo de dados são testados. Os testes de caixa-preta ou funcionais avaliam se as funções do software estão operacionais, isto é, se a entrada é adequadamente aceita, a saída é produzida de forma correta e a integridade de informação externa (por exemplo, uma base de dados) é mantida.

O planejamento para a realização de testes é importante, e pode ser divido em fases. Por exemplo, um planejamento de testes pode ter as seguintes fases: teste de unidade, teste de integração, teste de sistema, teste de aceitação. A fase teste de unidade ou módulo testa as menores unidades de software desenvolvidas (no caso do paradigma estruturado, a menor unidade é uma função ou procedimento). A fase de teste de integração tem como objetivo encontrar falhas provenientes da integração dos componentes do programa. A fase de teste

de sistema tem como objetivo identificar erros de requisitos funcionais (funcionalidades) e não-funcionais (características de desempenho) que não estão de acordo com as especificações. A fase de teste de aceitação é realizada por um grupo de usuários finais do programa, que realizam simulações no programa de forma a verificar se o comportamento do sistema está de acordo com o solicitado.

A Análise de Ponto de Teste (TPA - *Test Point Analysis*) é utilizada especificamente para estimar o esforço necessário para a execução de teste de aceitação e de teste de sistema. Para isso, além do tamanho funcional determinado pelos pontos de função (pela técnica de Análise de Pontos de Função - APF), a TPA considera importante outros dois elementos: a estratégia de teste e a produtividade.

Com a técnica TPA também é possível determinar ou calcular a importância relativa de várias funções do sistema, com a visão de utilizar o tempo de teste disponível tão eficientemente quanto possível.

Dentre as técnicas de teste de software, a TPA cobre somente os testes funcionais ou de caixa-preta. Em uma estimativa para testes de caixa-preta, três elementos são importantes: o tamanho do sistema a ser testado, a estratégia de teste e o nível de produtividade (Figura 7) . Os dois primeiros elementos juntos determinam o volume de trabalho de teste a ser realizado, sendo expresso em pontos de testes. Se a quantidade de pontos de teste é multiplicada pela produtividade, nós obtemos a estimativa de teste em horas. Ou seja, o resultado da aplicação da TPA é medido em unidade de esforço (horas).

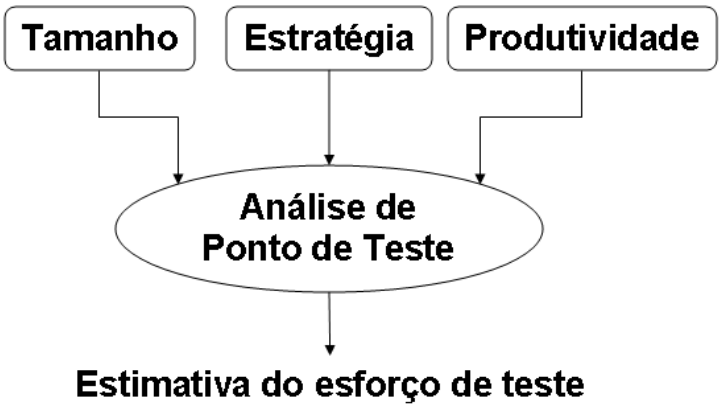


Figura 7: estimativa de teste com TPA.

O procedimento básico da TPA é mostrado na Figura 8.

A seguir analisamos as alternativas:

(A) CORRETA

Conforme explicamos anteriormente, a TPA é utilizada para estimar o esforço necessário para a execução de teste de aceitação e de teste de sistema, cobrindo somente os testes de caixa-preta. Portanto, a alternativa está correta.

(B) ERRADA

A soma dos pontos de teste atribuído a cada função do sistema é o número de pontos de teste dinâmico (*dynamic test points*), ou seja, os pontos de teste dinâmico levam em consideração as características de cada função.

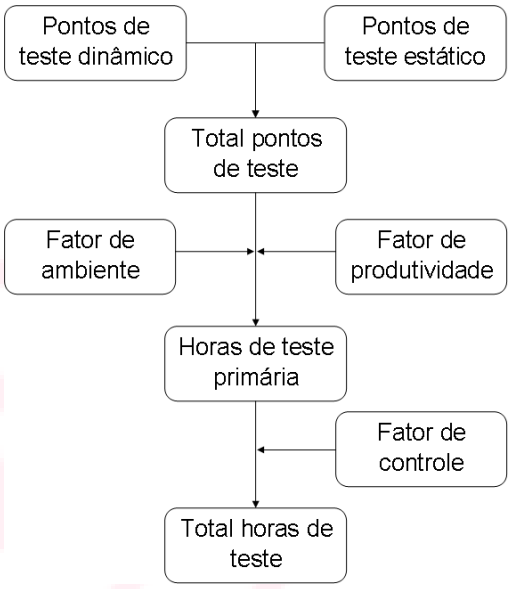


Figura 8: procedimento básico da TPA.

A soma dos pontos de função atribuído ao sistema todo é o número de pontos de teste estático (*static test points*), ou seja, os pontos de teste estático levam em consideração o sistema como um todo.

Esta alternativa está errada, pois afirma que os pontos de teste dinâmico levam em consideração o sistema como todo e os pontos de teste estático levam em consideração cada uma das características isoladas.

(C) CORRETA

O fator de produtividade indica o número de horas de teste necessário por ponto de teste. Quanto maior o fator de teste, maior o número de horas de teste requeridas. O fator de produtividade é definido com base na experiência, no conhecimento e na habilidade da equipe, o que pode variar de uma empresa para outra. O fator de produtividade pode ser determinado com base na análise de dados de projetos já concluídos. Na prática, o fator de produtividade tem mostrado ter um valor entre 0,7 e 2.0.

Então, o fator de produtividade depende da qualidade da equipe de teste. Quanto mais qualificada a equipe for, menor o fator e, consequentemente, menos números de horas de teste serão necessárias. Portanto, alternativa correta.

(D) CORRETA

O número total de pontos de teste é a soma dos pontos de teste dinâmico e estático. O número de horas de teste primárias pode ser calculada multiplicando o número total de

pontos de teste pelo fator de ambiente (environmental factor) e pelo fator de produtividade. As horas de teste primárias representam o volume de trabalho envolvido nas atividades de teste primárias, isto é, o tempo necessário para a conclusão das fases de teste de Preparação, Especificação, Execução e Conclusão.

A experiência com a técnica TPA sugere que os seguintes percentuais são apropriados para a divisão percentual das horas primárias em cada fase: Preparação, 10%; Especificação, 40%; Execução, 45%; e Conclusão, 5%. Logo, a alternativa está correta.

O número total de horas de teste é obtido adicionando horas de atividade secundárias (Planejamento e Controle) ao número de horas de teste primária. A quantidade de número de horas de atividade secundárias representa o volume de trabalho envolvido em atividades de gerenciamento. O número total de horas de teste é uma estimativa do tempo necessário para todas as atividades de teste, excluindo as de planejamento de teste.

(E) CORRETA

O número de horas de teste necessárias para cada ponto de teste é influenciado não somente pelo fator de produtividade, mas também pelo fator de ambiente. A quantidade de variáveis de ambiente é que defini o cálculo do fator de ambiente, que leva em consideração as seguintes variáveis de ambiente: ferramentas de testes, testes de desenvolvimento, testes básicos, ambiente de desenvolvimento, ambiente de teste, testware, método de cálculo.

A variável ambiente de desenvolvimento reflete a natureza do ambiente na qual o sistema foi desenvolvido. Neste contexto, o grau no qual o ambiente de desenvolvimento preveniu erros e método de trabalho inapropriado tem importância. Esta variável leva em consideração se um sistema foi desenvolvido utilizando: uma linguagem 4GL (linguagem de programação que descreve o que você quer que seja feito) com integração com banco de dados; uma linguagem 4GL com integração com uma linguagem 3GL; ou uma linguagem 3Gl como PASCAL e COBOL.

Portanto, a alternativa está correta.

22. **Assuntos relacionados:** *Engenharia de Software*, *Análise e Especificação de Requisitos*, *Requisito Funcional*, *Requisito Não-Funcional*,
Banca: *FCC*
Instituição: *MPU*
Cargo: *Analista de Desenvolvimento de Sistemas*
Ano: *2007*
Questão: *68*

Considere a seguinte especificação: "O sistema deverá inserir os dados por ordem de telefonema (data e hora) atentando para os critérios de segurança e confiabilidade ora estabelecidos. A arquitetura deve ser suficientemente prática, a fim de oferecer a máxima manutibilidade e a orientação a objeto é fundamental para garantir a reusabilidade". São requisitos não funcionais

(a). confiabilidade e manutibilidade, mas não reusabilidade.

(b). ordem de inserção e confiabilidade, mas não segurança.

(c). manutibilidade e reusabilidade, mas não segurança.

(d). confiabilidade, manutibilidade e reusabilidade, mas não ordem de inserção.

(e). segurança e confiabilidade, mas não ordem de inserção e nem manutibilidade.

Solução:

A Análise e Especificação de requisitos pode ser dividida em duas fases: a Especificação de requisitos e a Análise. Na fase de Especificação, requisitos são capturados de acordo com a perspectiva do usuário, ou seja, requisitos funcionais (funcionalidades) e não funcionais (restrições) são consideradas para atender às necessidades dos usuários. Na Análise, as estruturas internas de um sistema são moldadas para satisfazer os requisitos identificados.

Os requisitos funcionais são requisitos que expressam serviços ou funções que um sistema deve ter ou poder ser capaz de fornecer ou executar. Em geral, as funções ou serviços são processos que utilizam entradas para produzir saídas. São exemplos de requisitos funcionais: um sistema tem que possibilitar o cálculo dos gastos diários, semanais, mensais e anuais com o pessoal; um sistema deve emitir relatórios de compras e vendas por mês e entre outros.

Os requisitos não-funcionais são requisitos relacionados às restrições, ou atributos de qualidade para um sistema ou para um processo de desenvolvimento do sistema. Custo, confiabilidade, segurança, manutenabilidade, portabilidade, performance e rastreabilidade de informações são exemplos de requisitos não funcionais.

A distinção entre requisitos funcionais e não-funcionais nem sempre é clara. Entretanto, um requisito funcional expressa alguma transformação realizada pelo software, enquanto um requisito não-funcional expressa como essa transformação irá se comportar ou que qualidades específicas ela terá.

No caso do enunciado da questão, os requisitos funcionais são: inserir os dados por ordem de telefone. Enquanto, os requisitos não-funcionais são: segurança, confiabilidade, manutibilidade e reusabilidade.

Então, conforme explicado anteriormente, a alternativa correta desta questão é a letra D.

23. **Assuntos relacionados:** *Engenharia de Software*, *Análise e Especificação de Requisitos*,
Banca: *FCC*
Instituição: *MPU*
Cargo: *Analista de Desenvolvimento de Sistemas*
Ano: *2007*
Questão: *69*

A fase do desenvolvimento de sistema na qual as necessidades dos usuários são identificadas e as funcionalidades do sistema são modeladas é atualmente conhecida como

(a). Análise de Petri.

(b). Elicitação de Requisitos.

(c). Modelagem de Dados.

(d). Elaboração da Rede de Petri.

(e). Licitação e Referencial Técnico.

Solução:

Em um processo de desenvolvimento de software, a Análise e a Especificação de requisitos se destaca entre as principais atividades. Nesta atividade, os requisitos de um sistema são levantados e modelados, para só então a solução ser projetada e implementada. Nesta atividade, os principais profissionais envolvidos são o analista e o cliente/usuário.

A atividade de Análise e Especificação de requisitos é um processo de descoberta, refinamento, modelagem e especificação do sistema a ser desenvolvido. As funções de desempenho do software são especificadas, as interfaces com os outros sistemas são indicadas e restrições que o software deve atender são estabelecidas. Modelos dos dados, do controle e do comportamento operacional são construídos, e critérios para a avaliação da qualidade em atividades subseqüentes são instituídos.

A Análise e Especificação de requisitos pode ser dividia em duas fases: a Elicitação (especificação) de Requisitos e a Análise.

(A) Vide comentário da alternativa D.

(B) Na fase de Especificação, os requisitos são capturados de acordo com a perspectiva do usuário, ou seja, requisitos funcionais (funcionalidades) e não funcionais (restrições) são considerados para atender às necessidades dos usuários. Na análise, as estruturas internas de um sistema são moldadas para satisfazer os requisitos identificados.

A fase de Especificação de requisitos trata os requisitos do sistema de uma perspectiva externa, independente do paradigma ser estruturado ou orientado a objetos. Entretanto, a Análise, que é moldada a partir de uma perspectiva interna, depende do paradigma adotado.

Um aspecto fundamental no desenvolvimento de software é a captura dos requisitos dos usuários. Para dar apoio a este trabalho, algumas técnicas de levantamento de requisitos podem ser utilizadas, como: amostragem, investigação, entrevistas, questionários, observação e prototipação.

(C) A especificação de um sistema, o produto da fase de Análise, é composta de dois modelos: o Modelo Ambiental e o Modelo Comportamental. No primeiro, a fronteira entre o

sistema e o resto do mundo é definida, por exemplo: a finalidade do sistema e a lista de eventos. No modelo Comportamental, o comportamento necessário das partes internas do sistema para interagir com o ambiente é definida.

A modelagem de dados é a primeira atividade a ser realizada no processo do modelo Comportamental. Para realizar a modelagem de dados, utiliza-se o modelo de Entidades e Relacionamentos (ER).

O modelo ER é uma técnica top-down de modelagem conceitual, empregada para representar os dados que serão armazenados no sistema (banco de dados). Basicamente, o modelo ER representa as entidades com os respectivos atributos e os relacionamentos entre elas, descrevendo o modelo de dados de um sistema em um nível mais abstrato.

(D) A rede de Petri é uma das várias representações matemáticas para sistemas distribuídos discretos. Algumas aplicações típicas da rede de Petri: sistemas distribuídos, banco de dados, sistemas de produção, circuitos integrados, e etc.

Esta técnica possui particularidade de permitir modelar sistemas paralelos, concorrentes, assíncronos e não-determinísticos.

(E) Esta alternativa foge completamente ao enunciado da questão.

24. **Assuntos relacionados:** *Requisito Não-Funcional*, *Requisito Funcional*, *Engenharia de Software*,
Banca: *FCC*
Instituição: *TRT 2a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2008*
Questão: *41*

A frase "o tempo médio de resposta do sistema não deve ultrapassar 5 segundos" indica

(a). uma funcionalidade do sistema.

(b). uma atividade do cronograma do sistema.

(c). uma função executada pelo usuário do sistema.

(d). uma possível definição de requisito não funcional.

(e). um ponto de controle nas etapas de desenvolvimento do sistema.

Solução:

No contexto desta questão, tempo de resposta se refere ao período de tempo necessário para que um sistema execute uma determinada tarefa e apresente o seu resultado ao usuário. Por exemplo, imagine que você tenha um sistema que, dentre outras coisas, criptografa arquivos. O período entre uma solicitação de criptografia e o retorno do arquivo criptografado é o tempo de retorno dessa funcionalidade (criptografar arquivo) do sistema. Portanto, não faz sentido dizer que uma restrição de tempo de resposta é uma função de sistema. Dessa forma, pode-se eliminar as alternativas A e C.

A alternativa B, na verdade, não faz nenhum sentido. Talvez o autor desta questão quis dizer "uma atividade do cronograma do desenvolvimento do sistema". Mas mesmo assim, não seria difícil de concluir que ela ainda estaria errada.

Também se pode eliminar facilmente a alternativa E, já que o frase indica um requisito para o sistema, e não para o desenvolvimento do sistema.

A verdade é que uma atenta leitura já bastaria para se concluir que a alternativa correta é a D.

De qualquer forma, vale destacar os conceitos de requisitos funcionais e requisitos não-funcionais. De certa, requisitos funcionais descrevem funções que um sistema deverá realizar para os usuários. Dois exemplos são: (1) contar palavras em um editor de texto e (2) enviar arquivos em um sistema de mensagens instantâneas. Já os requisitos não-funcionais influenciam em requisitos de desempenho e outros atributos de qualidade do sistema. Dois exemplos são: (1) contar palavras de arquivos que contenham até 1 milhão de palavras em um editor de texto e (2) enviar arquivos de até 500 MB em um sistema de mensagens instantâneas.

25. **Assuntos relacionados:** *Análise e Especificação de Requisitos*, *Implantação da Função de Qualidade (IFQ)*, *Requisitos Normais*, *Requisitos Esperados*, *Requisitos Excitantes*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas Pleno - Processos*
Ano: *2006*
Questão: *48*

Analise as afirmativas abaixo a respeito de técnicas de levantamento de requisitos:

I - Uma entrevista não estruturada deve "fluir" entre o entrevistado e o entrevistador e, para isso, as questões a serem feitas não se devem ser definidas previamente.

II - A Implantação da Função de Qualidade (IFQ) é uma técnica que traduz as necessidades do cliente para requisitos técnicos de software, identificando três tipos de requisitos: normais, esperados e excitantes.

III - Amostragem é o processo de seleção sistemática de elementos representativos de uma população, que permite revelar informações úteis acerca da população como um todo.

IV - Uma técnica importante no levantamento de requisitos é observar o comportamento e o ambiente do indivíduo tomador de decisões, já que muitas informações passam desapercebidas com a utilização de outras técnicas.

Estão corretas apenas as afirmativas:

(a). I e II.

(b). III e IV.

(c). I, II e III.

(d). I, II e IV.

(e). II, III e IV.

Solução:

A Análise de Requisitos é o processo de compreensão, de modelagem e de especificação das necessidades do usuário de software a fim de entender os requisitos específicos que precisam ser satisfeitos para construir software de alta qualidade. Pode ser assim dividida: reconhecimento do problema; avaliação e síntese; modelagem; especificação; e revisão.

Antes de analisar, modelar ou especificar os requisitos, é preciso reuni-los (reconhecimento do problema) por meio de um processo chamado Levantamento de Requisitos. A técnica para levantamento de requisitos mais comumente utilizada é a condução de reuniões ou entrevistas. É importante que as questões apresentadas sejam previamente formuladas para que o foco da reunião/entrevista seja mantido e para que seu objetivo seja alcançado.

Uma técnica de gestão de qualidade que traduz as necessidades do cliente para requisitos técnicos de software chama-se Implantação da Função de Qualidade (ou Desdobramento da Função de Qualidade – Quality Function Deployment). Ela busca enfatizar o entendimento do que tem valor para o cliente e, depois, desdobrar esses valores através do processo de engenharia. Esta técnica identifica três tipos de requisitos:

- **Requisitos Normais:** objetivos e metas que são estabelecidos para o produto ou sistema durante reuniões com o cliente;

- **Requisitos Esperados:** requisitos implícitos no produto ou sistema que podem ser tão fundamentais que o cliente não se refere a eles explicitamente;
- **Requisitos Excitantes:** características que vão além das expectativas do cliente e mostram-se muito satisfatórias quando presentes.

A Implantação de Função de Qualidade utiliza entrevistas com o cliente e observação, levantamentos e exame de dados históricos durante a atividade de levantamento de requisitos. Entretanto, nem todos os aspectos relevantes para a construção do software solicitado são suscitados em reuniões formais. O comportamento e o ambiente dos futuros operadores do produto de software (indivíduo tomador de decisões) devem ser observados e analisados sob a óptica da Engenharia de Software. Dados importantes do fluxo de trabalho podem ser coletados através dessa técnica.

Como o universo de indivíduos envolvidos na utilização do futuro produto de software pode ser muito grande (a ponto de inviabilizar entrevistas individuais), é importante que uma amostragem dessa população seja sistematicamente selecionada para compor a equipe que fará parte do processo de levantamento de requisitos. Essa amostragem permite obter estatisticamente informações úteis acerca da população como um todo.

Face ao exposto, dentre as assertivas de I a IV apresentadas na questão, a única que não condiz com a teoria é a I, pois a condução de uma entrevista não estruturada pode comprometer o objetivo da mesma, não elucidando questões importantes que precisam ser previamente formuladas. A letra é (E) apresenta o conjunto correto de assertivas verdadeiras.

26. **Assuntos relacionados:** *Engenharia de Software*, *Análise e Especificação de Requisitos*, *Gerenciamento de Requisitos*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas Pleno - Processos*
Ano: *2006*
Questão: *49*

Sobre a Análise e o Gerenciamento de Requisitos, é **FALSO** afirmar que:

(a). quanto mais tarde for identificado um problema na análise de requisitos, maior será o custo com o retrabalho.

(b). a elicitação é o processo de identificação e entendimento das necessidades e restrições dos usuários, enquanto que a especificação é o processo de formalização das necessidades e restrições dos usuários em requisitos funcionais de software.

(c). na análise de requisitos o cliente utiliza as melhores práticas de engenharia de requisitos na tarefa de descrever suas necessidades.

(d). o gerenciamento de requisitos corresponde ao conjunto de atividades que auxilia a equipe do projeto a identificar, controlar e rastrear os requisitos, bem como a fazer as alterações nos requisitos durante o projeto.

(e). o gerenciamento de requisitos implica a alteração, inclusão e/ou exclusão de requisitos ao produto de software, o que pode levar a alterações de prazos, de recursos humanos, de equipamentos e de tecnologia.

Solução:

(A) CORRETA

Os requisitos de um sistema de computação constituem uma especificação das características e propriedades do sistema ou uma descrição do que o sistema deve fazer, de como ele deve se comportar, bem como das suas restrições de operação.

O gerenciamento de requisitos esta associado aos principais problemas do desenvolvimento de software. Erros de requisitos podem ocorrer em várias fases do ciclo de vida de um projeto de desenvolvimento de software. Se uma unidade de custo 1 é associada ao esforço requerido para detectar e reparar um erro durante a fase de codificação, o custo para detectar e reparar um erro durante a fase de requisitos está entre 5 a 10 vezes menor. Por outro lado, o custo para detectar e reparar um erro durante a fase de manutenção é 20 vezes maior. As razões para essa grande diferença é que a maioria desses erros não é detectada logo após terem sido cometidos. Esse atraso na descoberta dos erros significa que o custo para repará-los inclui o custo para corrigir o erro em questão, o custo para corrigir as consequências do erro e o custo dos investimentos subseqüentes. Esses investimentos incluem refazer o código, reescrever a documentação e reinstalação da aplicação. Isso sem citar custos indiretos que não são medidos monetariamente, como desgaste junto ao cliente, perda de credibilidade, perda de novas oportunidades, dentre outros.

(B) CORRETA

Elicitar requisitos é usualmente atribuído a atividade voltada para descobrir (identificar, deduzir, extrair e obter) os requisitos de um sistema, através de entrevistas com os interessados pelo sistema, de documentos do sistema existente (manual ou automatizado), da

análise do domínio do problema ou de estudos de mercado. A elicitação consiste na identificação dos dados de uso, a análise dos fluxos de trabalho dos usuários, a definição dos atributos de qualidade do sistema e o desenvolvimento de mecanismos que possibilitem o reuso de requisitos.

Com os requisitos compreendidos, analisados e aceitos, os mesmos devem ser documentados com um nível de detalhamento adequado, produzindo a especificação de requisitos funcionais do software. Pode ser utilizada a linguagem natural ou diagramas, como os propostos pela UML na elaboração deste documento. Logo, a especificação de requisitos é uma conversão dos requisitos em alguma forma-padrão.

(C) ERRADA

Depois de algumas entrevistas com o cliente e análise de documentos da empresa, o analista terá obtido algum conhecimento do domínio do problema e terá chegado ao entendimento tanto do problema a ser resolvido quanto das percepções do usuário sobre as características de uma solução efetiva. Além disso, o analista pode sumarizar as principais necessidades do usuário ou características do produto que foram definidos na entrevista utilizando boa práticas de engenharia de requisitos. Essas necessidades do usuário vivem no topo de nossa pirâmide de requisitos e servirão de guia para nos orientar em todas as tarefas seguintes.

(D) CORRETA

Requisitos definem as capacidades que o sistema deve apresentar. Normalmente, a adequação ou não do sistema ao conjunto de requisitos determina o sucesso ou o fracasso dos projetos. Assim, é importante descobrir quais são os requisitos do sistema, descrevê-los, organizá-los, e rastrear os eventos que provocam as suas mudanças. O gerenciamento de requisitos é uma abordagem sistemática de elucidar, organizar e documentar os requisitos do sistema e também um processo que estabelece e mantém um contrato entre cliente e a equipe de projeto sobre as mudanças de requisitos do sistema.

(E) CORRETA

Gerenciamento de Requisitos controla as alterações nos requisitos do sistema incluindo ou excluindo características obtidas no início do entendimento do problema do produto. Para estas alterações ocorrerem de forma natural sem possíveis prejuízos futuros é necessário o rastreamento. A rastreabilidade mostra o impacto das alterações sobre o sistema. Após esta etapa de analise dos requisitos impactados é gerado uma nova versão atualizada dos requisitos, atualização do Documento de Requisitos de Software (DRS) e a comunicação as áreas envolvidas

Uma parte crítica do gerenciamento de requisitos de software são as alterações e a avaliação do impacto da mudança durante o ciclo de desenvolvimento. Quando a mudança é proposta enquanto os requisitos estão sendo coletados, deve ser identificado como a alteração afeta outros requisitos. Se a alteração é proposta enquanto o sistema está em desenvolvimento, o impacto da alteração envolve verificar como a alteração afeta os requisitos, o projeto do sistema e sua implementação. Se a alteração é proposta depois que o sistema foi colocado em produção, deve haver também uma verificação adicional a fim de identificar como todos os stakeholders do sistema podem ser afetados pela alteração. Contudo, antes de iniciar ou mesmo depois de iniciar o projeto, as alterações impactam não somente a descrição do

problema, mas também prazos de entrega de versões ou o software completo, nos recursos requeridos para o projeto tanto humanos quantos tecnológicos.

Portanto, a resposta falsa é alternativa C.

27. **Assuntos relacionados:** *Engenharia de Software*, *Requisito de Software*, *Engenharia de Requisitos*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas Júnior - Processos de Negócio*
Ano: *2008*
Questão: *51*

Analise as afirmativas a seguir, sobre requisitos em projetos de software.

I - O rastreamento de requisitos é de grande importância para conduzir análises de impacto quando há mudanças em requisitos.

II - O acrônimo FURPS+ se refere aos requisitos não funcionais das categorias de Feasibility, Usability, Reliability, Performance e Supportability.

III - Um requisito pode conter, além da especificação, atributos que sirvam ao seu gerenciamento.

IV - Casos de uso são descrições da interação entre um ator e o sistema e, portanto, especificam apenas requisitos funcionais.

Estão corretas APENAS as afirmativas

(a). I e II.

(b). I e III.

(c). II e III.

(d). II e IV.

(e). III e IV.

Solução:

Requisito de software pode ser entendido como uma condição ou uma capacidade com a qual o sistema deve estar de acordo. Os requisitos também representam características que definem o critério de aceitação do software. Uma classificação clássica de requisitos (não é a única!) é feita dividindo-se os requisitos em:

- **funcionais:** aqueles que representam os comportamentos que um software deve apresentar diante de certas ações de seus usuários. Portanto, eles especificam o comportamento de entrada e saída do software. Ou seja, eles representam **o que** tem que ser feito pelo software sob quais circunstâncias;
- **não-funcionais:** aqueles que qualificam determinados aspectos do comportamento do software. Portanto, eles expressam **como** o software deve fazer o seu trabalho. Em geral, esses requisitos se relacionam com os seguintes aspectos: confiabilidade, performance, robustez, etc. Perceba, então, que requisitos não-funcionais, de certa forma, podem constituir restrições aos requisitos funcionais.

Um exemplo rápido de cenário para esclarecer os conceitos é o seguinte. Em um terminal de caixa eletrônico, os tipos de transações bancárias suportadas são requisitos funcionais. Já a facilidade de uso, o tempo de resposta e o tempo médio entre falhas são requisitos não-funcionais.

Os requisitos funcionais e não-funcionais podem ainda ter um segundo nível de classificação:

- **explícitos:** são os identificados e relacionados em um documento chamado "Especificação de Requisitos";
- **implícitos:** são os não-identificados e, consequentemente, que não fazem parte da documentação gerada. Normalmente, eles são expectativas dos clientes e usuários, que possivelmente reclamam por tais requisitos ausentes no produto. Como se pode perceber, esse tipo de requisito é muito indesejável durante um desenvolvimento de software;
- **normativos:** são aqueles que decorrem de leis, regulamentos, padrões e normas que o software deve obedecer.

A Engenharia de Requisitos é uma disciplina da Engenharia de Software composta por técnicas de levantamento, documentação e análise de requisitos de software. Resumidamente, a sua proposta é fazer um levantamento bem-feito dos requisitos, minimizando os implícitos, analisá-los de forma adequada e gerar uma documentação que possa ser corretamente entendida pelos desenvolvedores.

Feita essa apresentação sobre requisitos de software, é hora de irmos para os itens apresentados no enunciado.

Rastrear um requisito é descobrir e registrar aspectos como: a sua origem; a sua justificativa; com quais outros requisitos ele se relaciona; qual é a sua influência no projeto, na implementação, nos testes e na documentação para o usuário. Portanto, o rastreamento de requisitos gera uma espécie de mapa com todos os requisitos identificados. É bastante sensato pensar que esse tipo de ferramenta auxilia em análises de impacto em casos de mudanças em requisitos. Conclui-se, então, que o item I é uma afirmação verdadeira.

O item II é uma afirmação falsa. O problema está no trecho "se refere aos requisitos não funcionais". FURPS é o acrônimo para Feasibility, Usability, Reliability, Performance e Supportability. Ele é um dos vários modelos existentes para classificação de requisitos (funcionais e não-funcionais). A classificação é feita com 5 principais categorias (e suas subcategorias):

- **Funcionalidade (Feasibility)**
  - conjuntos de recursos
  - habilidades
  - segurança
- **Usabilidade (Usability)**
  - fatores humanos
  - estética
  - consistência na interface do usuário
  - ajuda on-line e contextual
  - assistentes e agentes
  - documentação do usuário
  - materiais de treinamento
- **Confiabilidade (Reliability)**
  - frequência e gravidade de falha
  - possibilidade de recuperação
  - possibilidade de previsão
  - exatidão

  - tempo médio entre falhas (MTBF)
- **Desempenho (Performance)**
  - velocidade
  - eficiência
  - disponibilidade
  - exatidão
  - taxa de transferência
  - tempo de resposta
  - tempo de recuperação
  - uso de recurso
- **Suportabilidade (Supportability)**
  - possibilidade de teste
  - extensibilidade
  - adaptabilidade
  - manutenibilidade
  - compatibilidade
  - possibilidade de configuração
  - possibilidade de serviço
  - possibilidade de instalação
  - possibilidade de localização (internacionalização)

FURGS foi desenvolvido pela HP como um conjunto de fatores de qualidade de software. A principal proposta desse modelo é a utilização desses fatores em cada passo do processo de engenharia de software (investigação, especificação, projeto, implementação, testes, suporte) em busca de uma maior qualidade para os produtos e serviços gerados.

O "+" ao lado do acrônimo serve para nos lembrar que há outros requisitos em jogo. São eles:

- **Requisito de Design:** especifica e/ou restringe o design de um software;
- **Requisito de Implementação:** especifica e/ou restringe o código ou a construção de um software. Alguns exemplos são:
  - padrões obrigatórios
  - linguagens de implementação
  - políticas de integridade de banco de dados
  - limites de recursos
  - ambientes operacionais
- **Requisito de Interface:** em geral, especifica itens externos com o quais o sistema deve interagir e restringe formatos, tempos ou outros fatores usados por tais interações;
- **Requisito Físico:** especifica uma característica física que um sistema deve possuir. Geralmente é utilizado para representar requisitos de hardware, como:
  - material
  - forma
  - tamanho
  - peso

– topologia

No item III, temos uma afirmação verdadeira. Atributos de requisitos estão estreitamente ligados ao conceito de rastreabilidade apresentado anteriormente. Isso porque eles representam fontes de informações que possibilitam o rastreamento. Durante todo o processo de engenharia de software, os requisitos podem (e devem) ser consultados e atualizados em relação aos seus atributos, o que facilita de certa forma o gerenciamento do processo como um todo. Alguns exemplos de atributos são:

- **origem:** pessoa, documento ou outro requisito;
- **prioridade:** obrigatório, crítico, opcional, alta, média, baixa;
- **atribuído a:** pessoa, organização;
- **comentários:** sobre revisões;
- **dificuldade de Implementação:** alta, média, baixa;
- **status:** completado, parcial, não iniciado;
- **risco de Insatisfação:** alto, médio, baixo, escala de zero a dez;
- **método de Verificação:** análise, demonstração, inspeção, teste, auditoria.

O que torna o item IV falso é a palavra "apenas". Os concurseiros devem prestar muito atenção nesse tipo de palavra. Como o próprio item nos revela, um artefato Casos de Uso são descrições da interação entre um ator e o sistema. Na verdade, um Caso de Uso define um conjunto de instâncias de casos de uso, onde cada instância é uma sequência de iterações realizadas pelo sistema que produz um resultado palpável para um determinado ator. É um erro comum achar que esse tipo de artefato apenas aborda requisitos funcionais. É fato que esse tipo de requisito é mais visível, por conta de estarem no próprio diagrama. Contudo, há nesse tipo de artefato, uma séria de campos que descrevem, com palavras, várias propriedades. Dentre elas a propriedade de Requisitos Especiais que contemplam os requisitos não-funcionais.

Vimos, portanto, que apenas os itens I e III são afirmativas corretas. Logo, a alternativa que o concurseiro deve marcar é a letra b.

28. **Assuntos relacionados:** *Engenharia de Software*, *Requisito de Software*, *Requisito Funcional*, *Requisito de Domínio*, *Requisito Não-Funcional*,
Banca: *ESAF*
Instituição: *Agência Nacional de Águas (ANA)*
Cargo: *Analista Administrativo - Tecnologia da Informação e Comunicação / Desenvolvimento de Sistemas e Administração de Banco de Dados*
Ano: *2009*
Questão: *12*

Analise as seguintes afirmações sobre requisitos de sistemas de software:

I. Requisitos funcionais declaram as funções que o sistema deve fornecer, seu comportamento, e ainda, o que o sistema não deve fazer.

II. Requisitos de domínio são, exclusivamente, funcionais, pois exibem as características do domínio de aplicação do sistema.

III. Requisitos não-funcionais compreendem restrições sobre serviços ou funções do sistema.

Assinale a opção correta.

(a). Apenas as afirmações I e II são verdadeiras.

(b). Apenas as afirmações I e III são verdadeiras.

(c). Apenas as afirmações II e III são verdadeiras.

(d). As afirmações I, II e III são verdadeiras.

(e). Nenhuma das afirmações é verdadeira.

Solução:

Primeiramente, requisitos são as propriedades e funcionalidades utilizadas na construção de um sistema de software que implementa todas ou parte das necessidades de uma organização. Os requisitos se dividem no que diz respeito ao grau de detalhamento (diferentes níveis de descrição) em requisitos de usuário e requisitos de sistema. Resumidamente, requisitos de usuários são descrições de funcionalidades com baixo nível de detalhamento. E os requisitos de sistema são descrições detalhadas sobre o que o sistema deverá fazer. Os requisitos de usuários e sistemas são classificados em requisitos funcionais, requisitos não funcionais ou requisitos de domínio, mais detalhes sobre os mesmos são apresentados a seguir.

ITEM I

Os requisitos funcionais são procedimentos que o sistema deve implementar determinando as ações que serão tomadas a partir de entradas específicas. Além disso, em alguns casos os requisitos funcionais declaram o que o sistema não devem fazer. Quando são requisitos de usuários a especificação é construída a partir de declarações em um nível mais alto de detalhes (de maneira geral), porém, os requisitos funcionais de sistema descrevem detalhadamente o comportamento do sistema determinando entradas, saídas e exceções. Esta assertiva esta correta.

ITEM II

Os requisitos de domínio são requisitos que se originam do domínio de aplicação do sistema e que refletem características/propriedades do domínio da aplicação. Podem ser requisitos

funcionais ou não-funcionais e até mesmo restringir requisitos funcionais estabelecidos pelo cliente. Por exemplo: realização de algum cálculo específico. Os requisitos de domínio não são exclusivamente requisitos funcionais, desta forma, esta assertiva esta incorreta.

ITEM III

Os requisitos não-funcionais são condições ou propriedades que o sistema deve possuir geralmente sobre qualidade. Estes requisitos não descrevem o comportamento do sistema como os requisitos funcionais, eles apresentam atributos que devem ser esperado do sistema, como confiabilidade, tempo de resposta para determinado operação, desempenho, portabilidade, políticas da empresa (linguagem de programação usada, padrões de projeto), portabilidade, bem como restrições sobre capacidade de alguns dispositivos de hardware que são utilizados pelo sistema. Esta assertiva esta correta.

Portanto, a letra B indica acertadamente os itens I e III como corretos, sendo a resposta para a questão.

29. **Assuntos relacionados:** *Engenharia de Software*, *Requisitos de Usuário*, *Requisitos de Sistema*,
Banca: *ESAF*
Instituição: *Controladoria-Geral da União (CGU)*
Cargo: *Analista de Finanças e Controle - Tecnologia da Informação / Desenvolvimento de Sistemas de Informação*
Ano: *2008*
Questão: *45*

Requisitos são capacidades e condições às quais o sistema – e em termos mais amplos, o projeto – deve atender. Entre as diversas classificações e tipos de requisitos, encontramos requisitos do usuário, requisitos de sistema e especificação de projeto de software.

Assinale a opção que trata de requisitos de usuário.

(a). O usuário deve dispor de recursos para definir o tipo de arquivos externos.

(b). O software deve oferecer um meio de representar e acessar arquivos externos criados por outras ferramentas.

(c). Cada tipo de arquivo externo pode ter uma ferramenta associada que pode ser aplicada a ele.

(d). Cada tipo de arquivo externo pode ser representado com um ícone específico na tela do usuário.

(e). Devem ser fornecidos recursos para o ícone que representa um arquivo externo, a ser definido pelo usuário.

Solução:

Requisitos são condições que descrevem o comportamento exigido do sistema de software após a sua parcial ou completa implementação. Os primeiros requisitos de um sistema são definidos pelo usuário a partir de uma declaração abstrata das funcionalidades requeridas pelo mesmo. Outro tipo de requisito é o de sistema (requisitos de sistema) que incorpora os requisitos de usuário, porém o detalhamento dos procedimentos executados são mais especificados seguindo um padrão de organização das funções definidas. Os requisitos de usuários e de sistemas podem ser definidos da seguinte forma:

- **Requisitos do usuário**: são descrições em linguagem natural (ou formulário e diagramas) sobre o que o sistema deve fazer. Esses requisitos explicam as funcionalidades que o sistema deve fornecer, além, das restrições sobre o que o sistema não deve fazer. As descrições do usuário devem apresentar o comportamento do sistema numa ótica de usuário sem muitos detalhes técnicos;
- **Requisitos de sistema**: são as declarações apresentadas pelo usuário refinadas até chegar em uma descrição mais detalhada. Os requisitos de sistema são utilizados pelos engenheiros de software para a elaboração do projeto de sistema. Portanto, os requisitos de sistema devem determinar o que o sistema faz e não como ele será implementado.

Os requisitos de usuário de sistema podem ser divididos em requisitos funcionais e não-funcionais. Resumidamente, os requisitos funcionais descrevem as funcionalidades que o sistema deve apresentar. Já os requisitos não-funcionais referem-se às propriedades desejadas do sistema, por exemplo, tempo de resposta, confiabilidade do sistema. É claro, que estes requisitos devem ser expostos em uma linguagem de usuário e não em termos técnicos quando relacionados aos requisitos de usuários. Em contra-partida, caso sejam requisitos de

sistema devem possuir um nível de detalhamento mais técnico.

Depois da apresentação conceitual sobre os tipos de requisitos podemos ter certeza que o requisito de usuário deste problema é a descrição: **O software deve oferecer um meio de representar e acessar arquivos externos criados por outras ferramentas**, por ser definido de forma não-técnica e pouco detalhe. Logo, a questão correta é a letra B.

30. **Assuntos relacionados:** *Engenharia de Software*, *Análise de Sistemas*, *Análise Estruturada*, *PSPEC*,
Banca: *ESAF*
Instituição: *Superintendência de Seguros Privados (SUSEP)*
Cargo: *Analista Técnico da SUSEP - Tecnologia da Informação*
Ano: *2010*
Questão: *22*

A especificação de processos (PSPEC) em análise estruturada

(a). é uma listagem organizada de todos os elementos de dados pertinentes ao sistema.

(b). é uma listagem estruturada dos elementos de dados gerados por processos externos.

(c). é usada para priorizar os processos do Diagrama de Fluxo de Dados (DFD) de maior complexidade.

(d). especifica os processos do Diagrama de Fluxo de Dados (DFD) que não comportam descrição formal.

(e). é usada para descrever todos os processos do Diagrama de Fluxo de Dados (DFD) que aparecem no nível de refinamento final.

Solução:

A Modelagem de Análise é uma atividade de construção de modelos onde os requisitos (dados, funções e comportamento) do software são organizados e representados tanto na forma textual quanto na forma de diagramas, de modo que seja mais fácil entender e revisar os requisitos.

A Modelagem de Análise tem como objetivo: descrever o que o cliente exige; estabelecer a base para criação do projeto (design) do software; e definir um conjunto de requisitos que possam ser validados quando o software for construído.

A Modelagem de Análise é a primeira representação técnica do sistema. Muitos métodos ou metodologias foram propostos, mas os mais aplicados são a Análise Estruturada e a Análise Orientada a Objetos. A Análise Orientada a Objetos tem como propósito definir todas as classes (e relacionamentos e comportamentos associados) que são relevantes para o software.

A Análise Estruturada tem como foco os processos e os dados e é fundamentada no princípio da decomposição funcional por meio de uma abordagem top-down. Ou seja, os dados são considerados separadamente das funções que os transformam e a decomposição funcional é usada intensamente Tem como finalidade retratar o fluxo e o conteúdo das informações utilizadas pelo software, dividir o software em partes funcionais e comportamentais e descrever a essência daquilo que será construído.

A Análise Estruturada utiliza as técnicas de modelagem, as seguintes: Modelagem de Dados, utilizando Diagrama de Entidade-Relacionamento (DER); a Modelagem Funcional, utilizando Diagrama de Fluxo de Dados (DFD); e Modelagem de Controle, utilizando Diagrama de Transição de Estados (DTE) (vide Figura 9).

O DER mostra as relações (cardinalidade e modalidade) existentes entre os itens de dados. O DFD indica como os dados são transformados à medida que se movem no sistema,

isto é, indica as funções e sub-funções que transformam os fluxos de dados (processos). O DTE indica como o software se comporta em conseqüência de eventos externos.

O Dicionário de Dados é uma lista organizada dos elementos do software, com definições precisas e rigorosas, de modo que o usuário e o analista de sistemas tenham uma compreensão comum dos fluxos de entrada, dos fluxos de saída, dos componentes dos depósitos de dados e dos cálculos intermediários.

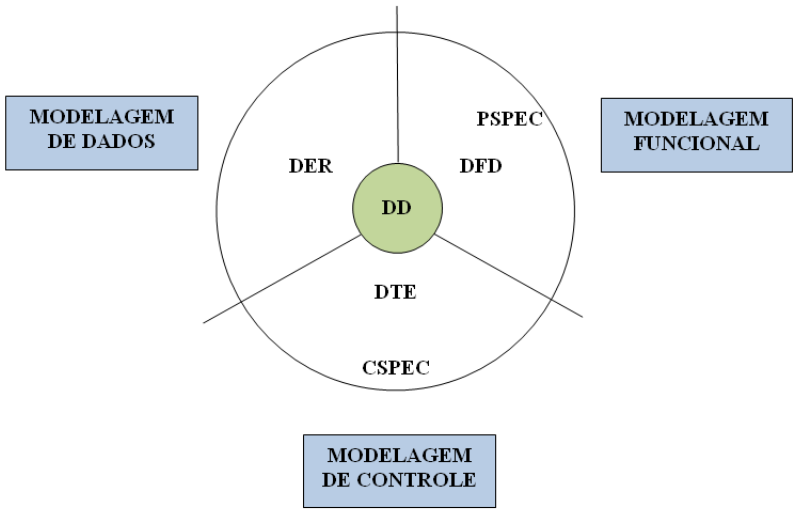


Figura 9: técnicas de modelagem da Análise Estruturada.

A Especificação de Processo (*Process Specification* - PSPEC) pode ser utilizada para especificar os detalhes de processamento implícito de uma bolha do DFD (processo), por exemplo: a entrada para uma função, o algoritmo, restrições e características de desempenho. Em outras palavras, PSPEC é utilizada para descrever o funcionamento interno de um processo representado em um diagrama de fluxo. O conteúdo da especificação de processo pode incluir texto narrativo, uma descrição em linguagem de projeto de programa do algoritmo do processo, equações matemáticas, tabelas, diagramas ou gráficos. As especificações criadas para o processo do DFD pode servir como primeiro passo para criação da especificação de requisitos do software e como diretriz para o projeto de componente de software que vai implementar o processo.

A Especificação de Controle (*Control Specification* - CSPEC) é usada para indicar como o software se comporta quando um evento ou sinal de controle é percebido e que os processos são invocados como consequência da ocorrência do evento. A CSPEC representa o comportamento do sistema de duas maneiras. O CSPEC contém um diagrama de transição de estado, que é uma especificação de comportamento seqüencial. Ele também contém uma tabela de ativação de processos (PAT), especificação combinatória de comportamento. A PAT representa a informação contida no DTE, no contexto de processos, não de estados. Isto é, a tabela indica que bolhas (processos) DFD serão invocados quando um evento ocorrer. A CSPEC descreve o comportamento do sistema, mas não nos dá informação sobre o trabalho interno dos processos

Um sistema desenvolvido usando um método estruturado, freqüentemente, é difícil de ser mantido. As principais dificuldades que os analistas enfrentavam com a aplicação da Análise

Estruturada é a distinção entre requisitos lógicos e físicos, e a ausência de uma abordagem para particionar o sistema em partes tão independentes quanto possível, de modo a facilitar o processo de análise.

De acordo com o explicado anteriormente, a alternativa correta é a letra **(E)**, a PSPEC é usada para descrever todos os processos do DFD que aparecem no nível de refinamento final.

31. **Assuntos relacionados:** *UML*, *Diagrama de Estados*, *Diagrama de Classes*, *Diagrama de Atividades*, *Diagrama de Componentes*, *Diagrama de Sequência*,
Banca: *CESGRANRIO*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Eng. de Software*
Ano: *2008*
Questão: *53*

Aplicações com interfaces web podem apresentar fluxos de navegação complexos entre suas páginas. Há links e botões a serem clicados a qualquer momento, disparando eventos de transição de uma página para outra e até para si mesma. Utilizando Javascript no lado do cliente, e possivelmente recursos Ajax, é possível habilitar ou desabilitar links e permitir ou impedir o envio de formulários com base em condições verificadas em tempo real. Dentre os tipos de diagrama UML listados a seguir, qual o que melhor modelaria o fluxo de navegação descrito?

(a). Classe

(b). Atividade

(c). Máquina de estado

(d). Componente

(e). Seqüência

Solução:

Antes de apontar qual é a alternativa correta, vamos rever cada tipo de diagrama UML envolvido nesta questão.

Diagrama de Classes

O objetivo de um diagrama de classes é descrever estaticamente os vários tipos de objetos no sistema, bem como o relacionamento entre eles. É importante ressaltar que esse tipo de diagrama pode oferecer três perspectivas distintas:

- **Conceitual:** perspectiva destinada aos clientes. Ela representa os conceitos do domínio em estudo;
- **Especificação:** perspectiva destinada às pessoas que não precisam saber detalhes de desenvolvimento. Ela evidencia as principais interfaces da arquitetura e métodos, sem detalhes sobre como eles serão implementados;
- **Implementação:** perspectiva destinada ao time de desenvolvimento. Ela aborda vários detalhes importantes sobre a implementação.

Como o diagrama de classes faz uma descrição estática do sistema, ele não é adequado para modelar nenhum tipo de fluxo. Ou seja, a alternativa A não é a que procuramos.

Diagrama de Atividades

Um diagrama de atividades é um diagrama de estados em que todos ou a grande maioria dos estados representam execuções de ações (atividades internas ao sistema). Portanto, ele é um gráfico de fluxo que evidencia os controles e as execuções das atividades. Seus principais elementos são:

- **início:** círculo preenchido;
- **estado ou atividade:** retângulo com bordas arredondadas;
- **transição:** seta;
- **decisão ou desvio:** losango - uma entrada e mais de uma saída;
- **intercalação ou merge:** losango - mais de uma entrada e uma saída;
- **separação ou fork:** barra horizontal - uma entrada e mais de uma saída;
- **junção ou join:** barra horizontal - mais de uma entrada e uma saída.

Um diagrama de atividades descreve uma sequência de atividades. Então, se ele fosse utilizado para descrever o fluxo de navegação descrito no enunciado, o foco ficaria nos eventos dos links e botões ao invés de nas páginas. Ou seja, dentre as alternativas que temos, o diagrama de atividades não é o mais adequado para o que se pede no enunciado.

Diagrama de Estados

Diagrama de estados também é comumente chamado de máquina de estados. Ele mostra os diferentes estados de um objeto durante sua vida, bem como os estímulos que acionam as mudanças de estado. Ou seja, o diagrama de estados vê o ciclo de vida dos objetos como máquinas de estados (automatos) finitos. O termo "finito" significa que existe um número finito de estados que o objeto pode assumir, bem como é finito o número de estímulos que acionam as mudanças de estado.

Na UML, o estado do objeto é definido pelos valores dos atributos de um objeto de uma determinada classe do modelo. No entanto, é importante ressaltar que nem todas as variações de valores de atributos devem ser representadas por estados exclusivos, mas apenas aquelas que podem afetar significativamente o trabalho do objeto no contexto da aplicação.

Existem dois tipos especiais de estados: inicial e final. Eles são especiais porque nenhum evento pode fazer com que um objeto retorne para seu estado inicial, bem como não existe nenhum evento capaz de tirar o objeto de seu estado final.

A Figura 10 mostra um exemplo de um diagrama de estados para um objeto do tipo servidor. Nele são mostrados 4 estados (pronto, escutando, trabalhando e desligando), além dos estados inicial e final. Também são mostrados os estímulos que acionam as mudanças de estado.

É este o tipo de diagrama adequado para modelar o fluxo de navegação entre páginas web. As páginas seriam encaradas como os objetos e os eventos de links e botões seriam modelados como os estímulos.

Diagrama de Componentes

Um diagrama de componente mostra os componentes do software (por exemplo, componentes CORBA, Java Beans ou seções do sistema que são claramente distintas) e os artefatos de que eles são feitos, como arquivos de código-fonte, bibliotecas de programação ou tabelas de bancos de dados relacionais. Sendo assim, nem de longe ele é útil para modelar algum tipo de fluxo de sistema.

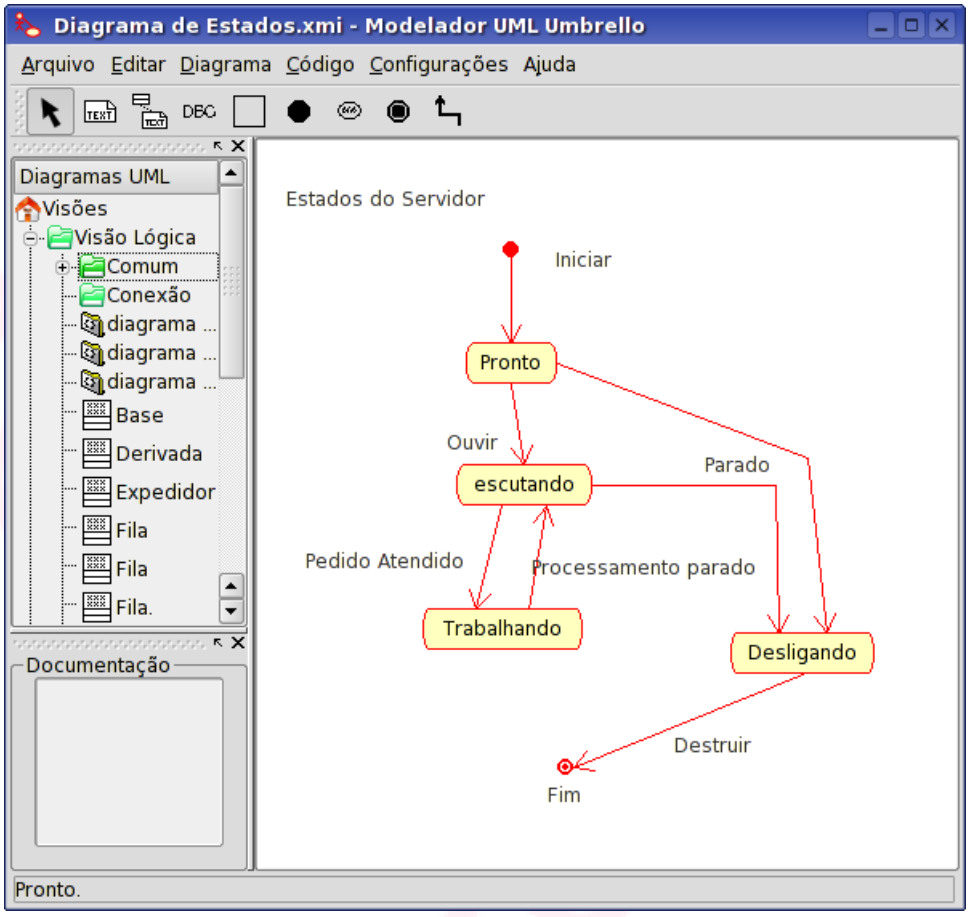
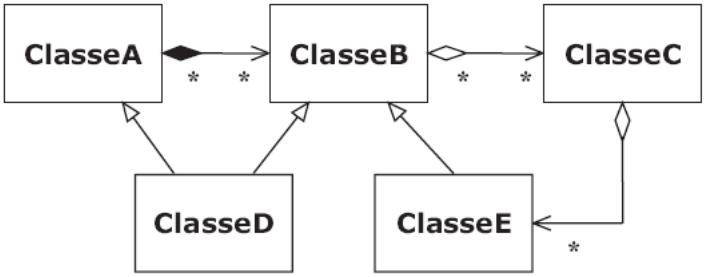


Figura 10: exemplo de Diagrama de Estados.

Diagrama de Sequência

Seu principal objetivo é descrever a sequência ao longo do tempo das comunicações entre objetos. A ênfase temporal é obtida com a linha vertical de tempo que está sempre presente neste tipo de diagrama. Assim como o diagrama de comunicação, o diagrama de sequência é um tipo de diagrama de interação. Ou seja, eles descrevem trocas de mensagens em situações dinâmicas. Perceba, portanto, que este tipo de diagrama não se mostra adequado a modelagem de fluxo de navegação.

32. **Assuntos relacionados:** *UML, Diagrama de Classes,*
Banca: *CESGRANRIO*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Eng. de Software*
Ano: *2008*
Questão: *54*



A figura acima mostra um diagrama de classes UML desenvolvido para um projeto em que ainda não se sabe em que linguagem será realizada a implementação. Sobre o diagrama, assinale a afirmação correta.

(a). Há um erro na cardinalidade da associação entre ClasseA e ClasseB, pois se trata de uma composição e, como tal, um objeto da ClasseB só pode estar associado a um objeto da ClasseA.

(b). Há uma dependência cíclica entre ClasseB, ClasseC e ClasseE, o que não é permitido pela UML.

(c). O fato de que ClasseD generaliza ClasseA e ClasseB se traduz em herança múltipla, o que não é permitido pela UML.

(d). Retirando a ClasseA, o diagrama resultante corresponde ao padrão de projeto *composite*.

(e). Invertendo o sentido de todas as generalizações, o diagrama resultante corresponde ao padrão de projeto *chain of responsability*.

Solução:

Lembre-se que diagramas de classe nos permitem identificar tanto o conteúdo de uma classe quanto o relacionamento entre várias classes. Em um diagrama de classe, podemos mostrar as variáveis e métodos membros de uma classe. Podemos também mostrar se uma classe herda de outra, ou se mantém uma referência para outra.

(A) CORRETA

O relacionamento "todo-parte", também conhecido como composição agregada, ou relacionamento "tem-um" ou "parte-de", indica que um objeto (o todo) é composto de outros objetos (as partes). Com a composição agregada, o relacionamento entre os objetos é muito mais forte que com a associação, pois nela o todo não pode existir sem suas partes e as partes não podem existir sem o todo. Vários pontos importantes são inerentes a este fato. São eles:

- remoção do todo implica na remoção das partes;
- existe apenas um todo, isto é, as partes não são compartilhadas com outros todos;
- as partes não podem ser acessadas "fora" do todo, ou seja, elas são particulares para o todo;
- uma mensagem destinada a uma parte deve ser enviada para o todo e retransmitida por ele à parte.

Isto significa que a agregação composta deve ser utilizada somente quando um objeto é considerado como uma parte de outro objeto e não apenas uma associação ocasional com existência e visibilidade independentes.

A fim de representar este tipo de relacionamento, utiliza-se uma linha que termina com um símbolo de diamante preenchido, símbolo este colocado contra o todo. Além disso, para evitar qualquer confusão possível, ao todo é atribuída, explicitamente, a multiplicidade de 1 (um), mesmo porque apenas um todo é possível.

A partir do diagrama UML e do que expomos, conseguimos classificar a Classe A como sendo o todo e a Classe B como a parte do relacionamento. Assim, podemos verificar que a alternativa **A** está correta, pois a multiplicidade 1 (um) deveria ser atribuída ao todo, e não uma multiplicidade denotada pelo símbolo * que representa nenhum ou mais objetos.

(B) ERRADA

Na verdade, o que não é permitido na UML é o que foge da sua semântica e sintaxe, pois lembremos que UML é uma linguagem de modelagem. Logo, a UML em sua semântica e sintaxe não impede a existência da dependência cíclica, pois essa não se preocupa com problemas da arquitetura do sistema ou da solução.

(C) ERRADA

Um relacionamento entre classes é chamado de relacionamento de generalização. Esse relacionamento também pode ser chamado de relacionamento de especialização, pois a generalização e a especialização são dois pontos de vista do mesmo relacionamento: as Classes A e B são generalizações da Classe D e esta é uma especialização das anteriores.

No diagrama de classes, uma generalização é representada por uma flecha partindo da subclasse (no caso, a Classe D) em direção à(s) superclasses (no caso, as Classes A e B).

É importante mencionar que, assim como a alternativa B, independentemente da linguagem de programação adotada permitir (C++) ou não (Java) herança múltipla, não será a UML que restringirá a modelagem de herança múltipla. Assim, em UML, uma classe pode ter mais de uma superclasse. No nosso caso, a Classe D possui tanto características da Classe A quanto da Classe B.

Concluímos, assim, que a alternativa está errada.

(D) ERRADA

Composite compõe objetos em estruturas do tipo árvore para representar hierarquias todo-parte. Faz também com que o tratamento dos objetos individuais e de suas composições

seja uniforme.
Aplicações gráficas como editores de desenho e sistemas de captura de esquema deixam os usuários construírem diagramas complexos a partir de Componentes simples. O usuário pode agrupar Componentes para formar Componentes maiores, que, por sua vez, podem ser agrupados para formar Componentes maiores ainda. O Padrão *Composite* descreve como usar composição recursiva, de modo que os clientes não tenham que fazer distinção entre componentes simples e composições.

A Figura 11 apresenta esta estrutura.

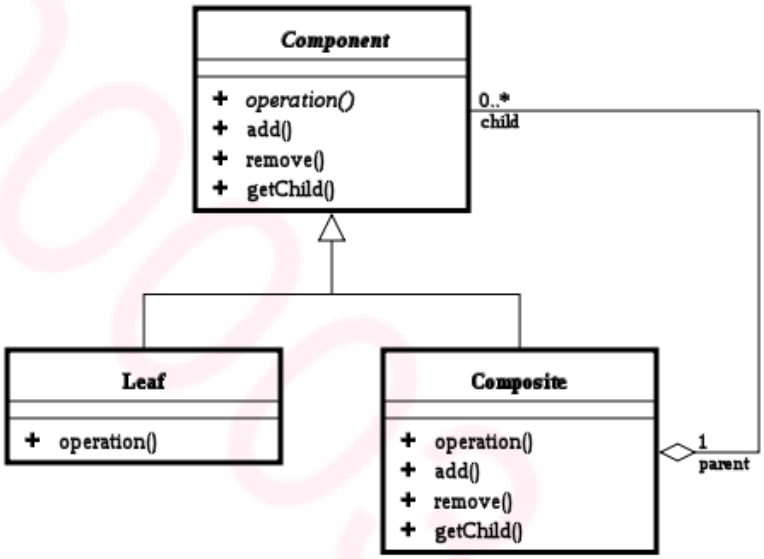


Figura 11: estrutura do padrão de projeto *Composite*.

Observe que, mesmo após a retirada da Classe A não conseguimos chegar à estrutura apresentada na figura: o *Composite* é composto por *componentes* e é um filho deste através da herança. Na questão não temos essa composição e herança ao mesmo tempo. Portanto, alternativa errada.

(E) ERRADA

O padrão de projeto de software *Chain of Responsibility* representa um encadeamento de objetos receptores para o processamento de uma série de solicitações diferentes. Esses objetos receptores passam a solicitação ao longo da cadeia até que um ou vários objetos a trate.

Cada objeto receptor possui uma lógica descrevendo os tipos de solicitação que é capaz de processar e como passar adiante aquelas que requeiram processamento por outros receptores. A delegação das solicitações pode formar uma árvore de recursão, com um mecanismo especial para inserção de novos receptores no final da cadeia existente.

Dessa forma, fornece um acoplamento mais fraco por evitar a associação explícita do remetente de uma solicitação ao seu receptor e dar a mais de um objeto a oportunidade de tratar a solicitação.

Um exemplo da aplicação desse padrão é o mecanismo de herança nas linguagens orientadas a objeto: um método chamado em um objeto é buscado na classe que implementa o objeto e, se não encontrado, na superclasse dessa classe, de maneira recursiva.

A Figura 12 apresenta esta estrutura.

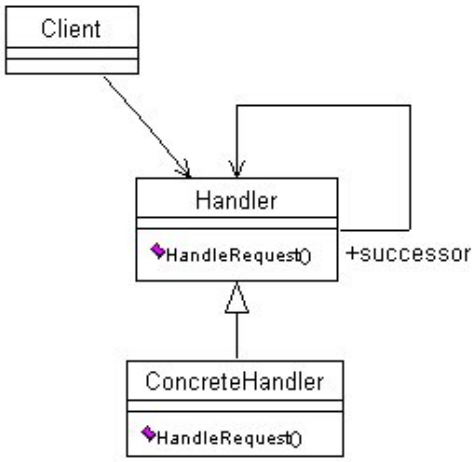
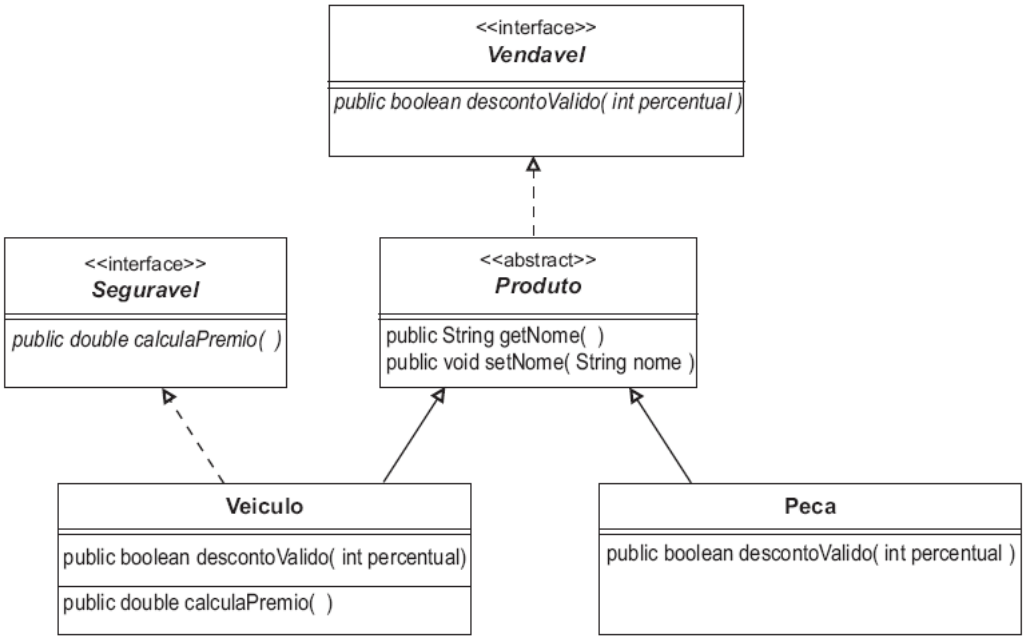


Figura 12: representação da estrutura do padrão *Chain of Responsability*.

Como podemos observar, a estrutura do padrão é bem simples e possui um auto-relacionamento (*successor*), o que nos permite descartar a alternativa.

33. **Assuntos relacionados:** *Desenvolvimento de Software*, *Diagrama de Classes*,
Banca: *CESGRANRIO*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Eng. de Software*
Ano: *2008*
Questão: *62*



Com base no diagrama acima, analise os trechos de código Java a seguir.

I - Produto p = new Produto();
p.setNome("Carro");

II - Seguravel s = new Veiculo();
s.setNome("Carro");
double p = s.calculaPremio();

III - Seguravel s = new Veiculo();
((Veiculo) s).setNome("Carro");

IV - Vendavel v = new Peca();
Produto p = (Produto) v;
p.setNome("Pneu");

V - Vendavel v = new Veiculo();
Produto p = (Peca)((Produto)((Veiculo) v));

Estão corretos **APENAS** os trechos de código

(a). I e III

(b). I e IV

(c). II e III

(d). III e IV
(e). IV e V

Solução:

(I) ERRADA

Observe pelo diagrama que Produto é uma classe abstrata, portanto não é possível aplicar um construtor sobre ela. Em outras palavras, ela não pode ser instanciada. Nós precisamos assegurar que o objeto a ser construído pertence a uma classe concreta da classe abstrata Produto, no caso, ou a classe Veículo ou a classe Peca. Uma solução de correção seria o seguinte código:

Produto p = new Veículo();

(II) ERRADA

Note que apesar da variável s ser uma instância da classe Veiculo, ela foi automaticamente convertida (*upcast*) no momento da sua declaração para a interface Seguravel. Desse modo, esse objeto poderá acessar apenas os métodos que foram definidos na interface, no caso, o método calculaPremio, e não o método setNome.

(III) CORRETA

Semelhante à alternativa anterior, porém com um conversão (*downcast*) explícita entre a interface Seguravel e a classe que a implementa (Veiculo). Este tipo de conversão permite a uma classe pai, no caso a interface Seguravel, acessar todos os métodos da classe filha.

Note, também, que s é uma instância da classe Veiculo, e por isso não foi necessário inserir mecanismos que verificam o tipo do objeto antes da conversão (RITT, do inglês *Run-Time Type Identification*).

Em Java isso é alcançado por meio do operador *instanceof*. O uso adequado desse operador garante que as conversões são sempre seguras, não permitindo que um objeto seja convertido para uma classe inválida.

(IV) CORRETA

No trecho de código são realizadas duas conversões:

1. *Upcast* entre uma instância da classe, a classe concreta Peca, e a interface Vendavel. Mais uma vez lembre-se que uma conversão para uma classe (ou interface) pai é feita implicitamente;
2. *Downcast* entre a interface Vendavel, que é uma instância da classe Peca, e a classe abstrata Produto. Observe, nesse caso, que a conversão é realizada para a classe pai (Produto) da classe instanciada (Peca), o que é perfeitamente aceitável uma vez que toda Peca é um Produto.

Após a realização do *downcast* o método setNome da classe Produto é apropriadamente invocado.

(V) ERRADA

No trecho de código, há uma tentativa de realizar um *cast* entre a instância de Veiculo, representada por v, para a classe Peca. Tal tipo de cast (entre classe irmãs) produzirá um erro de compilação, pois as referidas classes possuem métodos completamente diferentes e não relacionados.

34. **Assuntos relacionados:** *Java*, *UML*, *Diagrama de Classes*,
Banca: *CESGRANRIO*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Eng. de Software*
Ano: *2008*
Questão: *63*

```
public class Produto {

      private Fabricante mFabricante;

      public Produto () {
      }

      public Fabricante getFabricante () {
            return mFabricante;
      }

      public void setFabricante (Fabricante val) {
            this.mFabricante = val;
      }
}

import java.util.ArrayList;

public class Fabricante {

      private ArrayList<Produto> mProduto;

      public Fabricante () {
      }

      public ArrayList<Produto> getProduto () {
            return mProduto;
      }

      public void setProduto (ArrayList<Produto> val) {
            this.mProduto = val;
      }
}
```

Assinale o diagrama de classe que expressa corretamente a implementação mostrada acima, em Java, das classes Produto e Fabricante, bem como da associação entre as mesmas.

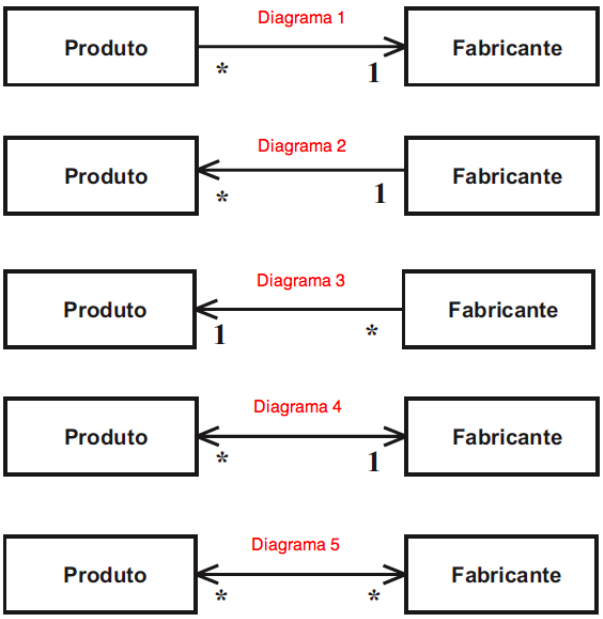
(a). Diagrama 1

(b). Diagrama 2

(c). Diagrama 3

(d). Diagrama 4

(e). Diagrama 5



Solução:

Lembre-se que diagramas de classe nos permitem identificar tanto o conteúdo de uma classe quanto o relacionamento entre várias classes. Em um diagrama de classes podemos mostrar as variáveis e métodos membros de uma classe. Podemos também mostrar se uma classe herda de outra, ou se mantém uma referência para outra. Em suma, podemos descrever todas as dependências do código-fonte entre classes. Para resolvermos esta questão precisamos saber:

1. associações entre classes muito frequentemente representam instâncias de variáveis que mantêm referência para outros objetos;
2. a direção da flecha nos informa que a classe mantém referência para outra classe;
3. o número próximo à cabeça da seta nos informa quantas referências são mantidas;
4. quando existem muitas conexões representamos por estrela (*). Em Java, isso é comumente implementado com um Vetor ou uma Lista.

De posse desse conhecimento, estamos aptos a revolver a presente questão.

Inicialmente, analisemos a declaração da classe Produto. Observe que nela é declarado um membro privado do tipo Fabricante. Em outras palavras, a classe Produto mantém uma única referência para classe Fabricante. Portanto, na associação existente entre Produto e Fabricante deve existir uma flecha apontando para a classe Fabricante com o valor numérico 1 (um) em sua ponta.

Agora, analisemos a declaração da classe Fabricante. Observe que nela é declarado um vetor (ArrayList) privado do tipo Produto, em outras palavras, a classe Fabricante mantém muitas referências para classe Produto. Portanto, na associação existente entre Fabricante e Fabricante também deve existir uma flecha apontando para a classe Produto com um asterisco (*) em sua ponta.

Concluímos, então, que a reposta correta é a letra **D**.

35. **Assuntos relacionados:** *Modelagem Funcional*, *Diagrama de Fluxo de Dados*,
Banca: *FCC*
Instituição: *TCE/CE*
Cargo: *Analista de Controle Externo - Auditoria de Tecnologia da Informação*
Ano: *2008*
Questão: *77*

NÃO é um elemento da modelagem funcional (DFD)

(a). o depósito de dados.

(b). a entidade associativa.

(c). o fluxo de dados.

(d). a entidade externa.

(e). a função ou processo.

Solução:

O Diagrama de Fluxo de Dados (DFD) é uma das ferramentas de modelagem funcional mais utilizadas no projeto de sistemas, sendo o elemento central da chamada análise estruturada.

O DFD pode ser entendido como uma ferramenta que permite modelar um sistema como uma rede de processos funcionais interligados por "dutos" e "tanques de armazenamento" de dados. Outra característica fundamental dessa ferramenta é o enfoque dado aos processos executados pelo sistema que está sendo modelado.

Portanto, o DFD tem por finalidade retratar o fluxo e o conteúdo das informações utilizadas pelo sistema, dividir o sistema em partições funcionais e comportamentais, além de descrever a essência daquilo do sistema. Para isso, o DFD faz uso dos quatro elementos básicos mostrados na Figura 13.

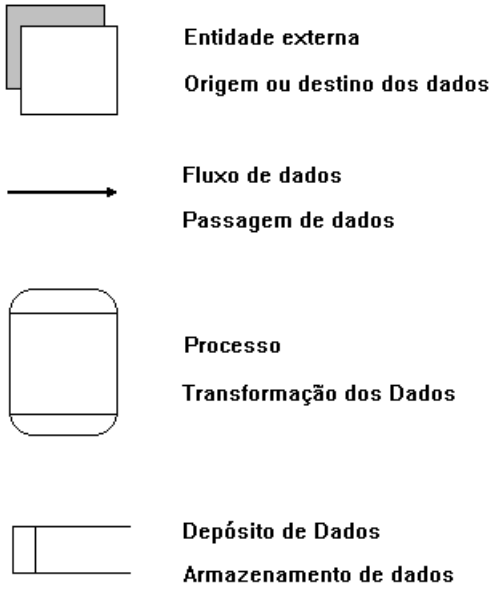


Figura 13: elementos do DFD.

Um **processo** representa as funções que são desempenhadas no sistema por meio da transformação de dados. Logo, o objetivo desse elemento é transformar dados por meio de uma função. A descrição de um processo é sempre uma sentença imperativa, constituindo um verbo no infinitivo, seguido por um objeto. Por exemplo: efetuar vendas mensais, incluir detalhes do novo cliente, averiguar crédito do cliente.

Uma **entidade externa** representa coisas ou pessoas que são origem ou destino dos dados representados no DFD. Exemplos: Pessoas: clientes, fornecedores, funcionários, contribuintes, departamentos de uma empresa, outros sistemas.

Um **depósito de dados** representa os dados que serão utilizados pelos processos no DFD. Esse elemento também é conhecido como repositórios de dados, e são responsáveis pelos dados em repouso.

Por fim, temos o elemento **fluxo de dados**, que pode ser considerado um canal que permite a passagem de dados entre os componentes do DFD. Os fluxos permitem as seguintes conexões: processo–processo, processo–entidade externa, entidade externa–processo, depósito de dados–processo e processo–depósito de dados.

36. **Assuntos relacionados:** *UML*,
Banca: *FCC*
Instituição: *TCE/CE*
Cargo: *Analista de Controle Externo - Auditoria de Tecnologia da Informação*
Ano: *2008*
Questão: *78*

Os mecanismos de extensibilidade da UML incluem

(a). estereótipos, apenas.

(b). valores atribuídos e restrições, apenas.

(c). estereótipos e valores atribuídos, apenas.

(d). estereótipos e restrições, apenas.

(e). estereótipos, valores atribuídos e restrições.

Solução:

A UML (Unified Modeling Language) é uma linguagem para especificação, documentação, visualização e desenvolvimento de sistemas orientados a objetos. Ela sintetiza os principais métodos existentes, sendo considerada uma das linguagens mais expressivas para modelagem de sistemas orientados a objetos.

Os mecanismos de extensibilidade da UML especificam como os elementos do modelo (por exemplo, as classes e os relacionamentos entre elas) podem ser personalizados e estendidos. Os três tipos de mecanismos de extensão do UML são:

- **Estereótipos:** permitem a classificação de um elemento de modelo estendido de acordo com um elemento de modelo base já existente na UML e a definição de novas restrições. Todos os elementos estendidos do modelo que possuírem um ou mais estereótipos terão valores adicionais, além dos atributos, associações e super classes que o elemento já possui na UML;
- **Restrições:** são regras de restrições bem definidas. Uma restrição permite que uma nova especificação semântica seja atribuída ao elemento de modelo, podendo ser adicionada tanto a um elemento de modelo diretamente quanto a um estereótipo;
- **Tagged Value:** As tagged values (valores atribuídos ou valores rotulados) permitem explicitar uma certa propriedade de um elemento através de um par: nome, valor. Essas informações podem ser adicionadas arbitrariamente a qualquer elemento do modelo, podendo ser determinado pelo utilizador ou por convenções das ferramentas de suporte.

Portanto, a resposta da questão é a alternativa E.

37. **Assuntos relacionados:** *UML, Classe Concreta, Classe Abstrata,*
Banca: *FCC*
Instituição: *TRT 2a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2008*
Questão: *32*

Na UML, a multiplicidade

(a). é aplicada aos atributos, somente.

(b). aplicada a uma classe é o número de instâncias que esta pode ter.

(c). indica a quantidade de atributos herdados por uma classe-filha em uma hierarquia de classes.

(d). aplicada nas associações entre classes indica o número de atributos comuns às classes envolvidas.

(e). aplicada a uma classe concreta é o número de operações que esta pode ter.

Solução:

UML (Unified Modeling Language), como o seu próprio nome revela, é uma linguagem de modelagem. Portanto, ela não é uma metodologia de desenvolvimento. Ou seja, ela não pretende apresentar quais são os passos necessários para que se obtenha os produtos de interesse. Ao invés disso, ela auxilia, por meio de diagramas, a visualização do projeto e das comunicações entre os seus objetos.

No contexto de UML, o termo multiplicidade pode estar associado a dois aspectos: atributos e associações entre classes.

Multiplicidade relacionada a atributo indica a dimensão desse atributo e, geralmente, é mais lembrada quando se trata de vetores e/ou matrizes. Por exemplo: se uma classe tem um atributo do tipo vetor "valores[5]", então a multiplicidade desse atributo é 5.

Multiplicidade também pode estar relacionada a associação entre classes. Nesse caso, ela tem um sinônimo: cardinalidade. Ela determina quantos objetos no sistema são possíveis em cada vértice da associação. A exibição da multiplicidade é feita por meio de um intervalo de valores não-negativos em cada vértice da associação [mín..máx]. O símbolo * pode ser utilizado para denotar o valor infinito. Vamos a um exemplo para facilitar o entendimento. Imagine uma associação entre as classes "Companhia" e "Empregado" que revela uma relacionamento em que "empregados trabalham em companhias". Se considerarmos que uma companhia sempre exige dedicação exclusiva de seus empregados, a multiplicidade mais sensata a ser aplicada nesse caso é [1] no lado da Companhia e [1..*] no lado da classe Empregado.

Figura 14: multiplicidade no UML.

Ou seja, um objeto da classe Companhia pode se relacionar com 1 ou mais objetos da classe Empregado (1 companhia tem 1 ou vários empregados) e um objeto da classe Empregado somente pode se relacionar com 1 objeto da classe Companhia (1 empregado somente pode

trabalhar em 1 companhia).

Com o que já foi exposto, não é complicada concluir que é a alternativa B a correta. Contudo, é importante se comentar cada alternativa incorreta.

(A) como já explicado acima, multiplicidade realmente se aplica a atributos, mas não SOMENTE a eles. Portanto, essa alternativa não é correta;

(C) sem dúvida, em hierarquia de classes, classes-filhas herdam determinados atributos. Contudo, o termo multiplicidade não indica quantos são esses atributos herdados;

(D) em associações entre classes, pode acontecer das classes envolvidas terem atributos comuns. O termo multiplicidade, porém, não indica quantos são esses atributos comuns;

(E) Classes abstratas são classes que não podem ser instanciadas, ou seja, não pode haver objetos que pertençam a elas. Elas servem para quando queremos generalizar operações ou definir interface comum a diversas subclasses. As operações (métodos) de uma classe abstrata podem conter implementações totalmente funcionais ou apenas a definição de um protocolo para a operação, ou seja, sua assinatura (nome da operação, parâmetros e retorno). No caso em que se provem apenas a assinatura, e não a implementação, a operação é declarada abstrata. Somente classes abstratas podem conter operações abstratas. Por outro lado, as classes concretas implementam de fato todas as suas operações e permitem a criação de instâncias. Uma classe concreta não possui métodos abstratos. O que muitas vezes acontece é uma classe concreta implementar os métodos abstratos assinados na classe abstrata (superclasse). Nesses casos, diz-se que a classe concreta é derivada da classe abstrata. De qualquer forma, o número de métodos (operações) de uma classe abstrata ou concreta não é representado pelo termo multiplicidade. Portanto, essa alternativa é errada.

38. **Assuntos relacionados:** *UML, Diagrama de Estados,*
Banca: *FCC*
Instituição: *TRT 2a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2008*
Questão: *33*

Dos diagramas definidos na UML 2.0, é aplicado na modelagem do comportamento de uma interface, classe ou colaboração, o Diagrama de

(a). Componente.

(b). Objeto.

(c). Estrutura Composta.

(d). Pacote.

(e). Estado de Máquina.

Solução:

A resposta da questão é a alternativa E, Diagrama de Estado de Máquina, que também é comumente chamado de Diagrama de Estados.

Os diagramas de estado mostram os diferentes estados de um objeto durante sua vida, bem como os estímulos que acionam as mudanças de estado. Ou seja, os diagramas de estado vêem o ciclo de vida dos objetos como máquinas de estados (autómatos) finitos. O termo "finito" significa que existe um número finito de estados que o objeto pode assumir, bem como um é finito o número de estímulos que acionam as mudanças de estado.

No UML, o estado do objeto é definido pelos valores dos atributos de um objeto de uma determinada classe do modelo. No entanto, é importante ressaltar que nem todas as variações de valores de atributos devem ser representadas por estados exclusivos, mas apenas aquelas que podem afetar significativamente o trabalho do objeto no contexto da aplicação.

Existem dois tipos especiais de Estados: inicial e final. Eles são especiais porque nenhum evento pode fazer com que um objeto retorne para seu estado inicial, bem como não existe nenhum evento capaz de tirar o objeto de seu estado final.

A Figura 15 mostra um exemplo de um diagrama de estados para um objeto do tipo servidor. Nele são mostrados 4 estados (pronto, escutando, trabalhando e desligando) além dos estados inicial e final. Também são mostrados os estímulos que acionam as mudanças de estado.

Com relação às demais alternativas da questão, são relevantes as seguintes informações.

O diagrama de componente mostram os componentes do software (por exemplo, componentes CORBA, java beans ou seções do sistema que são claramente distintas) e os artefatos de que eles são feitos, como arquivos de código fonte, bibliotecas de programação ou tabelas de bancos de dados relacionais.

Em UML, um pacote (package) é um mecanismo de agrupamento genérico. O Diagrama de pacotes comumente é utilizado para descrever como os pedaços do sistema são divididos e

agrupados, bem como mostrar as dependências entre eles. Este diagrama também é muito utilizado para ilustrar a arquitetura lógica do sistema, mostrando o agrupamento de suas classes. Nesse caso, é comum se falar em pacotes lógicos.

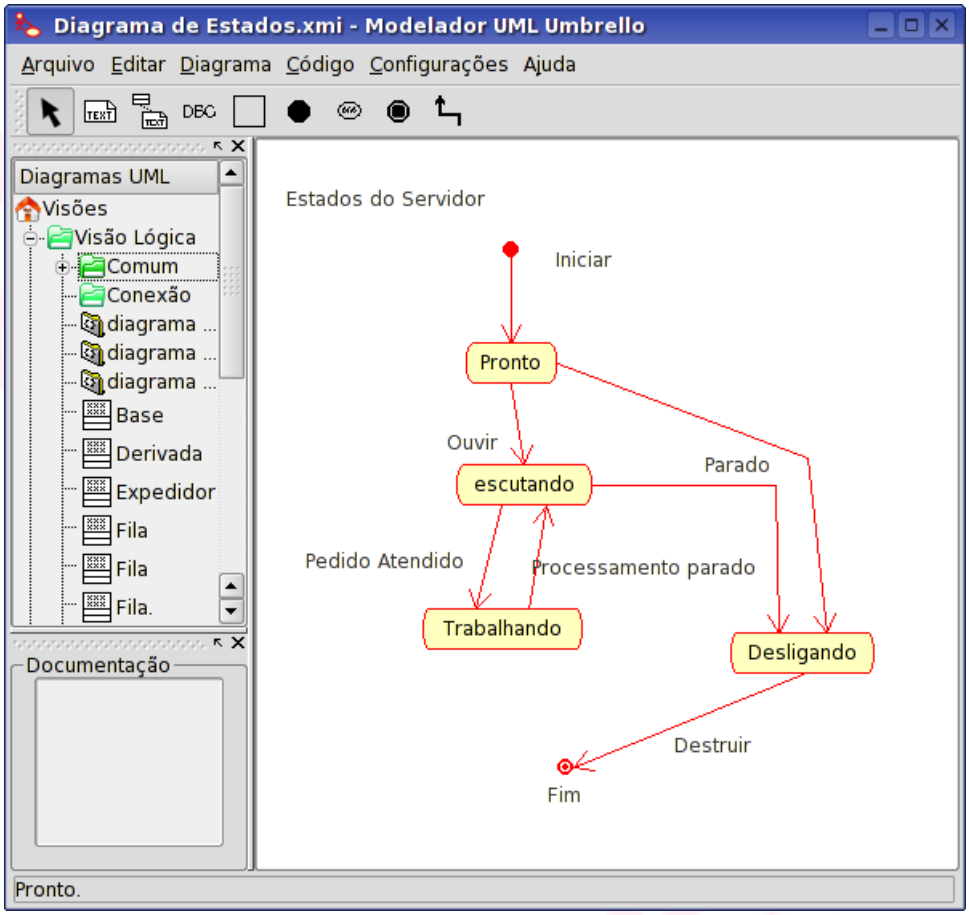


Figura 15: Diagrama de Estados.

O diagrama de estrutura composta foi definido no UML 2.0, e destina-se a descrição dos relacionamentos entre os elementos do modelo. Esse diagrama é utilizado para descrever a colaboração interna de classes, interfaces ou componentes, com o objetivo final de especificar uma funcionalidade.

Já o diagrama de objetos é uma variação do diagrama de classes, utilizando inclusive a mesma notação. A única diferença é que o diagrama de objetos mostra os objetos que foram instanciados das classes. Desse modo, o diagrama de objetos representa um perfil do sistema em um determinado momento de sua execução.

39. **Assuntos relacionados:** *UML, Diagrama de Caso de Uso,*
Banca: *FCC*
Instituição: *TRT 2a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2008*
Questão: *35*

Em um diagrama de Caso de Uso são admitidos os relacionamentos

(a). dependência, somente.

(b). dependência e generalização, somente.

(c). dependência, generalização e associação.

(d). associação, somente.

(e). dependência e associação, somente.

Solução:

Diagrama de Caso de Uso é um dos principais diagramas da UML (Unified Modeling Language). Ele descreve, de forma gráfica e intuitiva, relacionamentos e dependências entre um grupo de casos de uso e os atores que interagem com o sistema. O ponto de vista assumido para essa descrição é sempre o dos atores envolvidos.

Um ator é uma entidade externa ao sistema. Ele é representado graficamente por meio de um boneco e um rótulo com o seu nome. Geralmente, é um ator que inicia um caso de uso. Alguns exemplos de atores são: usuários, outros sistemas que fazem interface com o sistema que está sendo modelado e eventos externos.

Um caso de uso é um conjunto de atividades do sistema que produz um resultado concreto e tangível. Ou seja, ele define uma grande função do sistema. Graficamente, casos de uso são representados por elipses com seus respectivos rótulos.

Enfim, esse tipo de diagrama descreve interações entre os atores e o sistema em si. Veja um exemplo na Figura 16.

Os possíveis relacionamentos, que são representados no diagrama utilizando-se setas, são os seguintes:

- entre um ator e um caso de uso
  - **Relacionamento de Associação:** define do ponto de vista externo uma funcionalidade do sistema.
- entre atores
  - **Relacionamento de Generalização:** define que o ator que está na origem da seta tem seus próprios casos de uso. Já o ator que é apontado pela seta também tem os casos de uso do outro ator. Um exemplo típico desse tipo de relacionamento seria uma seta saindo do ator "Cliente On-line" e apontando para o ator "Cliente".
- entre casos de uso
  - **Relacionamento de Dependência-Inclusão:** um relacionamento de inclusão de um caso de uso A (origem da seta) para um caso de uso B (destino da seta) indica que B é essencial para o comportamento de A. Pode ser dito também que B é parte de A;

- **Relacionamento de Dependência-Exclusão:** um relacionamento de extensão de um caso de uso B para um caso de uso A indica que o caso de uso B pode ser acrescentado para descrever o comportamento de A (não é essencial). A extensão é inserida em um ponto de extensão do caso de uso A. Ponto de extensão em um caso de uso é uma indicação de que outros casos de uso poderão ser adicionados a ele. Quando o caso de uso for invocado, ele verificará se suas extensões devem ou não ser invocadas;
- **Relacionamento de Generalização ou Especialização (é-um):** um relacionamento entre um caso de uso genérico para um mais específico, que herda todas as características de seu pai. Quanto um caso de uso B é-um caso de uso A, A será uma generalização de B, ou seja, B será uma especialização de A.

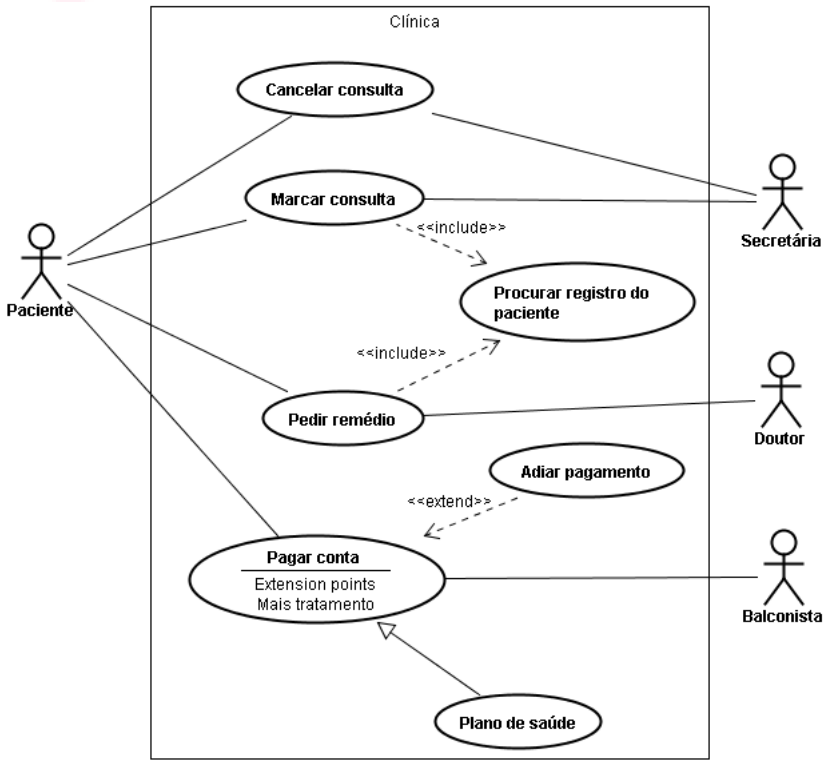


Figura 16: Diagrama de Caso de Uso.

Tendo em vista os possíveis tipos de relacionamento em Diagramas de Caso de Uso, pode-se concluir que a alternativa correta é a C.

40. **Assuntos relacionados:** *UML, Visões UML,*
Banca: *FCC*
Instituição: *TRT 2a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2008*
Questão: *36*

Um cubo, graficamente na UML, é um elemento físico existente em tempo de execução que representa um recurso computacional com pelo menos alguma memória, e, freqüentemente, com capacidade de processamento. Trata-se de

(a). pacote.

(b). nó.

(c). interface.

(d). objeto.

(e). nota.

Solução:

Em UML, um nó é um objeto físico de tempo de execução que representa um recurso computacional, possuindo, geralmente, memória e capacidade de processamento. Graficamente, um nó é representado pelo desenho de um cubo, como mostra a Figura 17.

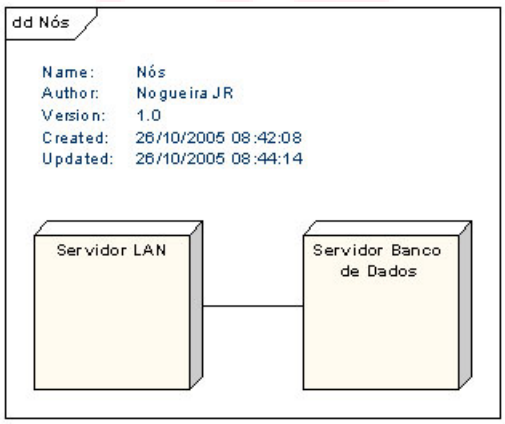


Figura 17: nó UML.

Os nós são utilizados na Visão de Implantação, que mostra a topologia de hardware em que o sistema é executado. Essa visão direciona principalmente a distribuição, o fornecimento e a instalação das partes que constituem o sistema físico.

Os aspectos estáticos dessa visão são capturados em diagramas de implantação, enquanto os aspectos dinâmicos são capturados pelos diagramas de interações, de estados e de atividades. Aproveitando o gancho, vamos falar um pouco sobre o conceito de Visões da UML.

Segundo Booch, visualizar, especificar, construir e documentar sistemas complexos de software são tarefas que requerem a visualização desses sistemas sob várias perspectivas, que também são conhecidas como visões. Segundo a UML, a arquitetura de sistema complexo de software pode ser descrita mais adequadamente por cinco visões interligadas. Cada visão constitui uma projeção na organização e estrutura do sistema.

Além da visão de implantação, que já foi mencionada, a UML ainda conta com mais outras quatro visões que são:

- **Visão do Caso de Uso:** abrange os casos de uso que descrevem o comportamento do sistema conforme é visto pelos seus usuários finais, analistas e pessoal do teste. Essa visão não especifica realmente a organização do sistema de um software. Porém, ela existe para especificar as forças que determinam a forma da arquitetura do sistema. Com a UML, os aspectos estáticos dessa visão são capturados em diagramas de caso de uso, enquanto os aspectos dinâmicos são capturados em diagramas de interação, diagramas de gráfico de estados e diagramas de atividades;
- **Visão de Projeto:** abrange as classes, interfaces e colaborações que formam o vocabulário do problema e de sua solução. Essa perspectiva proporciona principalmente um suporte para os requisitos funcionais do sistema, ou seja, os serviços que o sistema deverá fornecer a seus usuários finais;
- **Visão do Processo:** abrange as threads e os processos que formam os mecanismos de concorrência e de sincronização do sistema. Essa visão cuida principalmente de questões referentes ao desempenho, à escalabilidade e ao throughput do sistema. Com a UML, os aspectos estáticos e dinâmicos dessa visão são capturados nos mesmos tipos de diagramas da visão de projeto, mas com o foco voltado para as classes ativas que representam threads e processos;
- **Visão de Implementação:** abrange os componentes e os arquivos utilizados para a montagem e fornecimento do sistema físico. Essa visão envolve principalmente o gerenciamento da configuração das versões do sistema, composta por componentes e arquivos de alguma maneira independentes, que podem ser reunidos de diferentes formas para a produção de um sistema executável.

A literatura costuma definir o esquema de visões do UML como um esquema "4+1", no qual a visão de caso de uso é a visão central, sendo cercada pelas demais quatro visões apresentadas. A Figura 18 ilustra esse conceito.

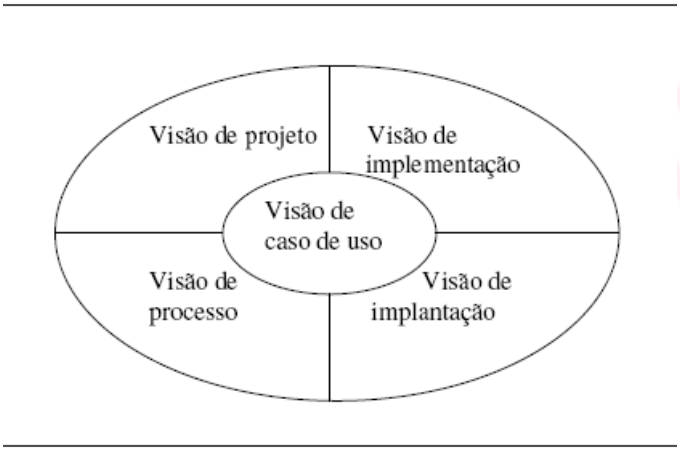


Figura 18: visões UML.

41. **Assuntos relacionados:** *UML*, *Diagrama de Objetos*, *Diagrama de Atividades*, *Diagrama de Comunicação*, *Diagrama de Sequência*, *Diagrama de Tempo*,
Banca: *FCC*
Instituição: *TRT 15a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2009*
Questão: *55*

Cobre um conjunto de instâncias dos itens encontrados nos diagramas de classe, expressa a parte estática de uma interação composta pelos objetos que colaboram entre si, mas sem qualquer uma das mensagens passadas entre eles e, também, congela um momento no tempo. Na UML, trata-se do diagrama de

(a). atividade.

(b). comunicação.

(c). sequência.

(d). tempo.

(e). objetos.

Solução:

A UML (Unified Modeling Language) é uma linguagem não-proprietária que permite modelagem de sistemas por meio de diagramas padronizados. Ela especifica tanto o significado gráfico quanto semântico de cada diagrama. A versão 2.0 da UML contempla os seguintes diagramas:

- Diagramas Estruturais
  - Diagrama de Classes
  - Diagrama de Objetos
  - Diagrama de Componentes
  - Diagrama de Instalação
  - Diagrama de Pacotes
  - Diagrama de Estrutura
- Diagramas Comportamentais
  - Diagrama de Caso de Uso
  - Diagrama de Transição de Estados
  - Diagrama de Atividade
- Diagramas de Interação
  - Diagrama de Sequência
  - Diagrama de Interatividade
  - Diagrama de Colaboração ou Comunicação
  - Diagrama de Tempo

Para se obter, com segurança, a alternativa correta, é importante conhecer cada um dos quatro diagramas apresentados como alternativas.

(A) Diagrama de Atividade

Um diagrama de atividades é um diagrama de estado em que todos ou a grande maioria dos estados representam execuções de ações (atividades internas ao sistema). Portanto, ele é um gráfico de fluxo que evidencia os controles e as execuções das atividades. Seus principais elementos são: início (círculo preenchido); estado ou atividade (retângulo com bordas arredondadas); transição (seta); decisão ou desvio (losango - uma entrada e mais de uma saída); intercalação ou merge (losango - mais de uma entrada e uma saída); separação ou fork (barra horizontal - uma entrada e mais de uma saída) e junção ou join (barra horizontal - mais de uma entrada e uma saída). Claramente este tipo de diagrama não se encaixa ao perfil descrito no enunciado. Portanto, esta não é a alternativa correta.

(B) Diagrama de Comunicação

Até a versão 1.5 da UML, este tinha o nome de diagrama de colaboração. Ele apresenta as interações existentes entre objetos em uma situação específica. Ele é um pouco similar ao diagrama de sequência. Enquanto um diagrama de comunicação apresenta mais claramente quais objetos se relacionam entre si, um diagrama de sequência enfatiza como as interações entre objetos ocorrem ao longo do tempo. Não é difícil concluir também que este tipo de diagrama também não se encaixa ao perfil descrito no enunciado. Ou seja, esta não é a alternativa buscada.

(C) Diagrama de Sequência

Como explicado no item anterior, um diagrama de sequência é similar ao de comunicação. Seu principal objetivo é descrever a sequência ao longo do tempo das comunicações entre objetos. A ênfase temporal é obtida com a linha de tempo vertical que está sempre presente neste tipo de diagrama. Assim como o diagrama de comunicação, o diagrama de sequência é um tipo de diagrama de interação. Ou seja eles descrevem trocas de mensagens em situações dinâmicas. Esse fato já bastaria para concluirmos que essa não é a alternativa correta.

(D) Diagrama de Tempo

Este diagrama pode ser entendido como uma fusão entre o diagrama de sequência e o de estado. Ele apresenta o comportamento dos objetos e suas interações em uma escala de tempo, juntamente com as mensagens que modificam os estados desses objetos. Como o próprio nome do diagrama sugere, ele não expressa parte estática alguma. Por isso, fica claro que essa também não é a alternativa certa.

(E) Diagrama de Objetos

Dentre os quatro diagramas apresentados como alternativas, este é o único estrutural. Sem dúvida, este fator é o atalho para a resolução consciente desta questão. Enquanto os diagramas estruturais modelam o sistema, ou parte dele, do ponto de vista estático, os diagramas comportamentais e de interação o fazem com ênfase à dinâmica envolvida.

O diagrama de objetos é muito similar ao diagrama de classes. Os dois seguem praticamente a mesma notação. A principal diferença é que em um diagrama de objetos, os atributos dos objetos possuem valores. Isso acontece porque ele evidencia o estado de um sistema em um determinado ponto do tempo. De forma simplificada, podemos entender o diagrama de objetos como uma instância do diagrama de classes. Por este diagrama apresentar todas as

características do enunciado, esta é a alternativa correta.

42. **Assuntos relacionados:** *UML*, *Diagrama de Comunicação*, *Diagrama de Sequência*,
Banca: *FCC*
Instituição: *TRT 15a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2009*
Questão: *56*

Duas características distinguem os diagramas de sequência dos de comunicação:

(a). linha do tempo de comunicação e mensagem.

(b). linha de vida do objeto e mensagem.

(c). linha de vida do objeto e foco de controle.

(d). linha de vida da mensagem e bifurcação.

(e). linha de vida da classe e nós de comunicação.

Solução:

Ambos os diagramas envolvidos nesta questão são ferramentas da UML (Unified Modeling Language). Tanto um quanto o outro, se propõem a revelar as interações entre objetos. Por isso eles são classificados como diagramas de interação. Apesar dos seus objetivos finais apontarem na mesma direção, cada diagrama apresenta uma ênfase diferente. Fundamentalmente, um diagrama de sequência é utilizado para evidenciar a cronologia entre as interações (trocas de mensagem) entre os objetos. Já um diagrama de comunicação é utilizado para enfatizar quais objetos se comunicam com quais, sem a informação cronológica das trocas de mensagens.

Uma boa estratégia para resolvermos esta questão é a de eliminação.

(A) e (B) ERRADAS

"Mensagem" não é uma característica que distingue os dois diagramas. Isso porque ambos trazem a informação de quais objetos trocam mensagens com quais objetos, cada um ao seu próprio estilo. Portanto, podemos eliminar estas duas alternativas.

(D) ERRADA

"Bifurcação", apesar de não ser a melhor tradução, se refere ao controle de execução fork (separação - não necessariamente em dois) do diagrama de atividade. Como essa não é uma característica de nenhum dos dois diagramas à prova, ela não oferece nenhuma distinção entre esses diagramas. Isso já basta para eliminarmos esta alternativa.

(E) ERRADA

Nenhum dos dois diagramas tem a característica "linha de vida da classe". Por conta disso, não se pode dizer que tal característica distingue os tipos de diagrama em questão. Portanto, eliminamos também esta alternativa. OBS: "linha de vida do objeto" sim é uma característica do diagrama de sequência que não aparece no diagrama de comunicação. Não confunda objeto com classe!

Por eliminação, concluímos que a alternativa correta é a letra c.

43. **Assuntos relacionados:** *UML*, *Diagrama de Sequência*, *Operadores de Controle Estruturados*,
Banca: *FCC*
Instituição: *TRT 15a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2009*
Questão: *57*

Os operadores de controle estruturados, aplicados aos diagramas de sequência (região retangular que circunscreve o diagrama e que serve para mostrar modos de execução), NÃO têm o objetivo de mostrar execução

(a). iterativa.
(b). condicional.
(c). paralela.
(d). opcional.
(e). comunicacional.

Solução:

O diagrama de sequência é uma das principais ferramentas da UML (Unified Modeling Language). Como o próprio enunciado desta questão sugere, esse tipo de diagrama pode conter operadores de controle estruturados que servem para evidenciar modos de execução dentro do diagrama. Esses operadores são representados por tags e retângulos que delimitam suas coberturas. Veja um exemplo na Figura 19.

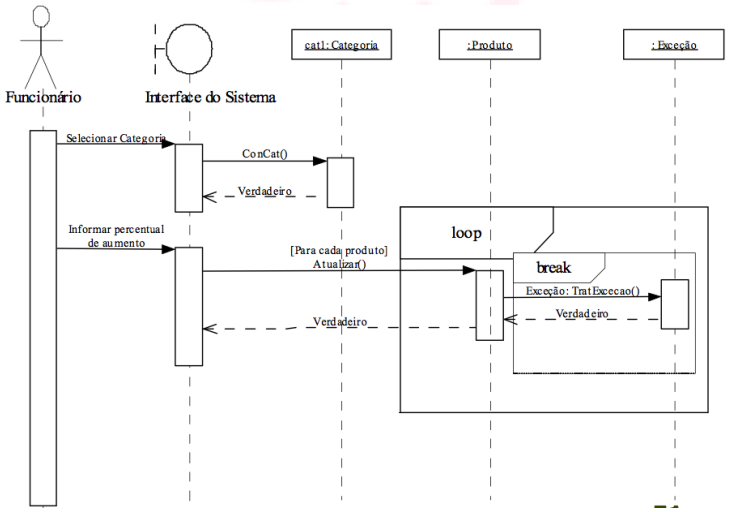


Figura 19: exemplo de operadores de controle estruturados.

Em resumo, os tipos de operadores de controle são:

- **Execução Opcional (tag opt):** o corpo do operador (área interna ao retângulo) somente é executado se uma condição de proteção for verdadeira. Essa condição é uma expressão booleana que pode aparecer entre colchetes acima de qualquer linha de vida no corpo;
- **Execução Condicional (tag alt):** o corpo do operador é subdividido por linhas horizontais tracejadas. Cada subdivisão tem uma condição de proteção. O corpo de uma subdivisão somente é executado se a sua condição for verdadeira;

- **Execução Paralela (tag par):** o corpo do operador também é subdividido por linhas horizontais tracejadas. Todas as subdivisões são executadas em paralelo;
- **Execução Iterativa (tag loop):** o corpo do operador é executado enquanto a condição de proteção, que aparece na parte superior de uma linha de vida no corpo, for verdadeira;
- **Quebra (tag break):** ele representa uma "quebra" na execução do processo. Por conta disso, ele é especialmente utilizado para modelar tratamentos de exceções;
- **Região Crítica (tag critical):** ele identifica uma operação atômica, ou seja, aquela que não pode ser interrompida.

Pelo o exposto, é fácil concluir que a única alternativa que traz um tipo de execução NÃO coberto pelos operadores de controle estruturados dos diagramas de sequência é a letra e.

44. **Assuntos relacionados:** *MER*, *DER*, *Entidade Associativa*, *Entidade Fraca*,
Banca: *FCC*
Instituição: *TRT 15a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2009*
Questão: *58*

Considere:

I Um relacionamento do tipo "material compõe material".

II Um relacionamento que necessita ser relacionado a outro relacionamento.

III Entidade cuja vida depende de outra.

No MER, I, II e III são, respectivamente, representados por:

(a). entidade associativa, auto relacionamento e entidade fraca.

(b). entidade associativa, entidade fraca e auto relacionamento.

(c). auto relacionamento, entidade fraca e entidade associativa.

(d). auto relacionamento, entidade associativa e entidade fraca.

(e). entidade fraca, entidade associativa e auto relacionamento.

Solução:

MER é o acrônimo de Modelo Entidade-Relacionamento. Ele é um modelo conceitual de alto-nível projetado para ser compreensível também pelos demandantes dos sistemas modelados. A principal ferramenta desse modelo é o DER (Diagrama Entidade-Relacionamento). Esse diagrama é utilizado para apresentar graficamente a modelagem elaborada pelo projetista. Atualmente não há uma única notação padrão de DER adotada pelo mercado. Contudo, todas elas são similares. Algumas delas foram propostas por Peter Chen, Bachman e James Martin.

São abordados abaixo, cada um dos três conceitos relacionados a MER que aparecem nas alternativas.

- **entidade associativa:** é uma entidade definida a partir da simplificação de um relacionamento do tipo N:M (muitos-para-muitos) entre duas ou mais entidades. Sua chave primária sempre é composta, no mínimo, pelas chaves primárias das entidades que participam do relacionamento. Imagine um relacionamento "vende", do tipo 1:N (um-para-muitos), entre as entidades PEDIDO e PRODUTO. Ou seja, um PEDIDO vende N PRODUTOS. Pode-se ter uma entidade associativa ITEM_DE_PEDIDO que teria a chave primária composta pelas chaves primárias das entidades PEDIDO e PRODUTO. Poderia ser, por exemplo, "número do pedido" e "código do produto". Perceba que o principal interesse em se criar entidades associativas se deve ao fato delas possuírem atributos que nem sempre fazem sentido nas entidades geradoras. Esse é o caso dos atributos "quantidade" e "desconto", que podem ser modelados para se relacionarem diretamente apenas com a entidade associativa ITEM_DE_PEDIDO. Perceba também que é o item II que representa o significado de entidade associativa;
- **auto relacionamento:** é um caso especial de entidade em que ela se relaciona com si mesma. Elas não são muito comuns, mas em alguns casos elas se mostram bastante úteis para a modelagem desejada do sistema. Um exemplo típico é o relacionamento "supervisiona" e a entidade EMPREGADO. Ou seja, um EMPREGADO supervisiona

N EMPREGADOS. É fácil perceber que o item I traz outro exemplo de auto relacionamento;

- **entidade fraca:** é justamente definida pelo item III. Um bom exemplo de entidade fraca é a entidade DEPENDENTE. Em um relacionamento "depende de", a existência da entidade DEPENDENTE é condicionada a existência da entidade EMPREGADO. Nesse caso, EMPREGADO é a entidade forte.

Tendo em vista o exposto, podemos concluir que a alternativa correta é a letra d.

45. **Assuntos relacionados:** *Engenharia de Software*, *UML*, *Diagrama de Caso de Uso*,
Banca: *FCC*
Instituição: *TRT 16a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2009*
Questão: *46*

Considere um Caso de Uso Base - UCB - que represente o atendimento a um trabalhador para uma reclamação trabalhista. Entretanto, na ocorrência de uma determinada condição como, por exemplo, "o reclamante tem precedentes judiciais", um outro Caso de Uso - UCS - envia um comportamento ao UCB. Nessa situação, a UML representa o relacionamento de UCB com UCS como

(a). exceção.

(b). extensão.

(c). generalização.

(d). agregação.

(e). inclusão.

Solução:

A UML (Unified Modeling Language) é uma linguagem visual para especificação, construção e documentação de softwares baseados na modelagem orientada a objeto. A UML não é um método de desenvolvimento, mas um modelo de linguagem utilizada para que a modelagem de sistemas seja consistência, simples, compreensível e de fácil comunicação com outros sistemas.

A UML é composta por elementos que modelam diferentes partes de um sistema. Os elementos ajudam a criar os diagramas, que ajudam a entender a arquitetura do sistema. A UML possui basicamente nove diagramas que são utilizados em combinação para fornecer os diferentes aspectos de sistemas, tais: diagrama de caso de uso, de classe, de objeto, de estado, de seqüência, de colaboração, de atividade, de componente e o de implantação.

Por meio dos diagramas, a UML aspectos estático (estrutura estática), dinâmico (comportamento dinâmico) e funcional. A estrutura estática é suportada pelos diagramas de classes e de objetos, que consiste nas classes e seus relacionamentos. O comportamento dinâmico é suportado pelos diagramas de caso de uso, de seqüência, de atividade, estado e colaboração. E a funcionalidade é suportada pelos diagramas de componente e implantação.

O diagrama de casos de uso descreve o comportamento de um sistema do ponto de vista dos usuários, possibilitando a definição dos limites de um sistema e das relações entre o sistema e o ambiente. Ou seja, os casos de uso são descrições de interações entre os usuários de um sistema e o sistema propriamente dito.

O diagrama de casos de uso é composto por atores, por casos de uso e por relacionamentos. Os atores representam o papel de uma entidade externa ao sistema como um usuário, um hardware, ou outro sistema que interage com o sistema modelado. Os casos de uso são processos ou funções que o sistema deve realizar de forma automática ou manual. Os relacionamentos podem ocorrem entre casos de usos, entre atores e casos de uso e entre atores. Os tipos de relacionamentos entre casos de uso são a extensão, a inclusão e a generalização. Os tipos de relacionamento entre atores e casos de usos é a associação. E para relacionamento

entre atores é a generalização.

Um caso de uso é representado por uma elipse conectada a símbolos de atores. Um retângulo mostra os limites (fronteiras) do sistema, contendo em seu interior o conjunto de casos de uso e, do lado externo, os atores interagindo com o sistema.

A associação é a participação de um ator em um caso de uso, mostrando a comunicação entre o ator com o caso de uso. A interação neste tipo de relacionamento ocorre por meio de mensagens e, no UML, é representado por uma linha entre ator e uma elipse.

A seguir, analisamos as alternativas desta questão:

(A) ERRADA

De acordo com explicado anteriormente, não existe o tipo de relacionamento exceção. Na especificação de casos de uso, existem os fluxos de exceção que descrevem os procedimentos que devem ser adotados no caso de erros durante o fluxo principal e alternativo. Logo, alternativa está errada.

(B) CORRETA

Um relacionamento de extensão indica que um deles terá seu procedimento acrescido, em um ponto de extensão, de outro caso de uso, identificado como base. Este tipo de relacionamento é utilizado para expressar rotinas de exceção e para descrever cenários opcionais de um caso de uso. Os casos estendidos descrevem cenários que somente ocorrerão em uma situação específica, se uma determinada condição for satisfeita.

Este relacionamento tem relação de dependência entre caso de uso de um mesmo diagrama.

Conforme o enunciado, o caso de uso "o reclamante tem precedentes judiciais" descreve uma situação específica do caso de uso base "o atendimento a um trabalhador para uma reclamação trabalhista". Logo, alternativa está correta.

(C) ERRADA

Um relacionamento do tipo generalização ocorre quando temos dois elementos semelhantes, mas com um deles realizando algo a mais. Este relacionamento especifica que um caso de uso herda as características de um caso de uso pai, e pode sobrepor algumas delas ou adicionar novas. Este relacionamento é semelhante à herança entre classes.

O relacionamento relatado no enunciado desta questão não é um relacionamento de generalização. Logo, alternativa errada.

(D) ERRADA

De acordo com explicado anteriormente, não existe relacionamento nos diagramas de caso de uso do tipo agregação. A agregação ocorre no relacionamento entre classes que ocorre no diagrama de classes do UML. Logo, alternativa errada.

(E) ERRADA

Um relacionamento do tipo inclusão indica que um dos casos de uso terá seu procedimento copiado num local especificado no outro caso de uso. Este tipo de relacionamento é utilizado quando existem cenários cujas ações servem a mais de um caso de uso. Este relacionamento é utilizado para rotinas de fluxo alternativo.

O relacionamento relatado no enunciado desta questão não é um relacionamento de inclusão. Logo, alternativa errada.

46. **Assuntos relacionados:** *Engenharia de Software*, *UML*, *Diagrama de Classes*,
Banca: *FCC*
Instituição: *TRT 16a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2009*
Questão: *47*

Considere diversas agências (classe Agencia) de atendimento a reclamações trabalhistas espalhadas em vários pontos do Estado. Uma delas, a central (classe AgenciaCentral), tem atributos diferenciados, porém herda os demais atributos e operações de Agencia. O relacionamento entre essas classes é definido na UML como

(a). inclusão.

(b). composição.

(c). específico.

(d). generalização.

(e). encapsulamento.

Solução:

A UML (Unified Modeling Language) é uma linguagem visual para especificação, construção e documentação de softwares baseados na modelagem orientada a objeto. A UML não é um método de desenvolvimento, mas um modelo de linguagem utilizada para que a modelagem de sistemas seja consistência, simples, compreensível e de fácil comunicação com outros sistemas.

A UML é composta por elementos que modelam diferentes partes de um sistema. Os elementos ajudam a criar os diagramas, que ajudam a entender a arquitetura do sistema. A UML possui basicamente nove diagramas que são utilizados em combinação para fornecer os diferentes aspectos de sistemas, tais: diagrama de caso de uso, de classe, de objeto, de estado, de seqüência, de colaboração, de atividade, de componente e o de implantação.

Por meio dos diagramas, a UML aspectos estático (estrutura estática), dinâmico (comportamento dinâmico) e funcional. A estrutura estática é suportada pelos diagramas de classes e de objetos, que consiste nas classes e seus relacionamentos. O comportamento dinâmico é suportado pelos diagramas de caso de uso, de seqüência, de atividade, estado e colaboração. E a funcionalidade é suportada pelos diagramas de componente e implantação.

As classes podem ter modelos, um valor que é usado para uma classe ou tipo não especificado. O tipo de modelo é especificado quando uma classe é iniciada (isto é, um objeto é criado).

Os diagramas de classe descrevem a estrutura estática de um sistema, mostrando as classes dos sistemas, os atributos delas, os objetos e os relacionamentos entre elas. Este diagrama não mostra a troca de mensagens entre as classes.

No diagrama de classes, temos os relacionamentos entre as classes e os objetos, que podem ser divididos em nível de instância e em nível de classe. Em nível de instância são a ligação (link), a associação e a agregação. Em nível de classes são a generalização, a especialização e a realização. Existem também os relacionamentos genéricos que são a dependência (uma classe depende ou usa outra classe) e a multiplicidade (número de instâncias de uma

classe que se relaciona a uma instância de outra classe).

A associação é uma conexão entre classes, e também significa que é uma conexão entre objetos daquelas classes. Em UML, uma associação é definida com um relacionamento que descreve uma série de ligações, onde a ligação é definida como a semântica entre as duplas de objetos ligados. As associações descrevem a conexão entre diferentes classes e podem ter uma regra que especifica o propósito da associação e pode ser uni ou bidirecional (indicando se os dois objetos participantes do relacionamento podem mandar mensagens para o outro, ou se apenas um deles sabe sobre o outro). Cada ponta da associação também possui um valor de multiplicidade, que dita como muitos objetos neste lado da associação pode relacionar-se com o outro lado.

As agregações são um tipo especial de associação no qual as duas classes participantes não possuem em nível igual, mas fazem um relacionamento todo-parte. Uma agregação descreve como a classe que possui a regra do todo é composta de outras classes, que possuem a regra das partes. Para agregações, a classe que age como o todo sempre tem uma multiplicidade de um.

A seguir, analisamos as alternativas desta questão:

(A) ERRADA

Um relacionamento do tipo inclusão indica que um dos casos de uso do diagrama de casos de uso terá seu procedimento copiado num local especificado no outro caso de uso. Este tipo de relacionamento é utilizado quando existem cenários cujas ações servem a mais de um caso de uso. Este relacionamento é utilizado para rotinas de fluxo alternativo.

Este tipo de relacionamento ocorre entre casos de uso e não entre classes. Logo, alternativa errada.

(B) ERRADA

Composições são associações que representam agregações muito fortes. Isto significa que composições formam também relacionamentos todo-parte, mas neste relacionamento as classes não podem existir independentes.

O enunciado trata do caso de um relacionamento entre classe com herança, onde a classe AgenciaCentral herda os atributos da classe Agencia, e não um relacionamento do tipo todo-parte. Logo, alternativa errada.

(C) ERRADA

A especialização ocorre quando uma classe é uma especialização de outra classe. Ou seja, uma ou mais subclasses são definidas a partir de uma classe existente. Às novas subclasses podem ser adicionadas propriedades e associações específicas.

De acordo com o enunciado, a classe AgenciaCentral possui apenas atributos diferenciados da classe Agencia, e não especifica novas subclasses. Logo, alternativa errada.

(D) CORRETA

Uma generalização entre duas classes coloca-as numa hierarquia, representando o conceito de herança de uma classe derivada de uma classe base. A generalização abstrai classes genéricas, a partir de classes com propriedades (atributos e operações) semelhantes. Em uma hierarquia de generalização, as subclasses herdam todas as propriedades de sua superclasse.

Conforme o enunciado, a classe AgenciaCentral herda propriedades da superclasse Agencia, caracterizando um relacionamento do tipo generalização. Logo, alternativa correta.

(E) ERRADA

O encapsulamento consiste na separação dos aspectos externos de um objeto, acessíveis por outros objetos, de seus detalhes internos de implementação, que ficam ocultos dos demais objetos. Este mecanismo é utilizado para impedir o acesso direto aos atributos de um objeto, disponibilizando métodos que alterem esses atributos. A utilização de encapsulamento é possível garantir a proteção aos atributos e as operações das classes.

De acordo com o enunciado, os atributos das classes AgenciaCentral e Agencia não estão encapsulados. Logo, alternativa está errada.

47. **Assuntos relacionados:** *UML*,
Banca: *FCC*
Instituição: *TRT 18a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2008*
Questão: *35*

Na notação original da UML 2.0, os símbolos + (mais) e # (jogo da velha), antecedendo as operações de uma classe, caracterizam tais operações, respectivamente, como

(a). pública e protegida.

(b). protegida e privada.

(c). pública e privada.

(d). pacote e protegida.

(e). pública e pacote.

Solução:

Em UML, a Visibilidade é uma enumeração cujos valores indicam se o elemento de modelagem ao qual se referem podem ser vistos fora de seu espaço de nome (namespace). Um namespace, por sua vez, é a parte do modelo na qual o nome pode ser definido ou usado. Dentro de um namespace, cada nome tem um significado único.

A visibilidade pode ser aplicada aos atributos e às operações em relação a uma classe ou entre classes e o pacote onde ela foi definida (seu contêiner). A UML define 4 níveis de visibilidade, mostrados na Tabela 1.

Símbolo	Visibilidade	Descrição
+	Público	Qualquer classe que possa ver o container também pode ver e usar as classes.
-	Privado	Apenas classes do mesmo container podem ver e usar a classe.
#	Protegido	Apenas classes do mesmo container ou de containers descendentes podem ver usar as classes.
∼	Pacote	Apenas classes do mesmo package que o container podem ver e usar as classes.

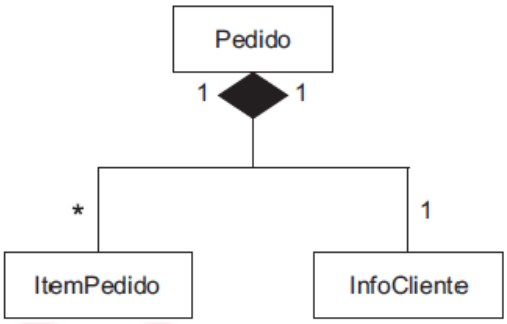
Tabela 1: níveis de visibilidade UML.

Atributos e operações com visibilidade pública em uma classe, por exemplo, podem ser vistos e utilizados por outras classes, enquanto atributos e operações com visibilidade privada podem ser vistos e utilizados apenas pela classe que os contém.

Portanto, a resposta da questão é a alternativa A.

48. **Assuntos relacionados:** *UML*, *Diagrama de Classes*, *Composição Agregada*,
Banca: *Cesgranrio*
Instituição: *BNDES*
Cargo: *Analista de Sistemas - Desenvolvimento*
Ano: *2008*
Questão: *35*

Considere o relacionamento de "todo-parte" ilustrado no diagrama UML abaixo.



É correto afirmar que

(a). um objeto da classe InfoCliente pode participar de mais de um relacionamento de composição desempenhando o papel de "parte".

(b). um objeto da classe ItemPedido pode participar de mais de um relacionamento de composição desempenhando o papel de "parte".

(c). uma instância da classe InfoCliente pode existir antes mesmo que a instância da classe Pedido com que se relacionará tenha sido criada.

(d). o relacionamento ilustrado acima é ternário.

(e). a cardinalidade do pedido no relacionamento com ItemPedido igual a 1 não precisaria ser apresentada, uma vez que não poderia assumir outro valor.

Solução:

Lembre-se que diagramas de classe nos permitem identificar tanto o conteúdo de uma classe quanto o relacionamento entre várias classes. Em um diagrama de classe, podemos mostrar as variáveis e métodos membros de uma classe. Podemos também mostrar se uma classe herda de outra, ou se mantém uma referência para outra.

O relacionamento "todo-parte", também conhecido como composição agregada, ou relacionamento "tem-um" ou "parte-de", indica que um objeto (o todo) é composto de outros objetos (as partes). Com a composição agregada, o relacionamento entre os objetos é muito mais forte que com a associação, pois nela o todo não pode existir sem suas partes e as partes não podem existir sem o todo. Vários pontos importantes são inerentes a este fato. São estes:

- remoção do todo implica na remoção das partes;
- existe apenas um todo, isto é, as partes não são compartilhadas com outros todos;
- as partes não podem ser acessadas "fora" do todo, ou seja, elas são particulares para o todo;

- uma mensagem destinada a uma parte deve ser enviada para o todo e retransmitida por ele à parte.

Isto significa que a agregação composta deve ser utilizada somente quando um objeto é considerado como uma parte de outro objeto e não apenas uma associação ocasional com existência e visibilidade independentes.

A fim de representar este tipo de relacionamento, utiliza-se uma linha que termina com um símbolo de diamante preenchido, símbolo este colocado contra o todo. Além disso, para evitar qualquer confusão possível, ao todo é atribuída, explicitamente, a multiplicidade de 1 (um), mesmo porque apenas um todo é possível.

A partir do diagrama UML e do que expomos, conseguimos classificar a classe Pedido como sendo o todo e as classes ItemPedido e InfoCliente como as partes do relacionamento. Assim, podemos eliminar as alternativas A e B, pois as partes não podem participar de mais de um relacionamento na composição agregada. Podemos eliminar, também, a alternativa C uma vez que uma parte não pode existir sem o todo. Lembre-se que seria necessário que as 3 (três) entidades estivessem associadas simultaneamente para que tivéssemos um relacionamento ternário, logo, a alternativa D também está errada. Portanto, a alternativa **E** é a correta, pois a multiplicidade 1 (um) atribuída ao todo é utilizada apenas para evitar qualquer confusão possível.

49. **Assuntos relacionados:** *Desenvolvimento de Sistemas*, *Modelagem Funcional*, *Diagramas UML*,
Banca: *Cesgranrio*
Instituição: *BNDES*
Cargo: *Analista de Sistemas - Desenvolvimento*
Ano: *2008*
Questão: *36*

O diagrama UML mais indicado para representar o passo a passo do fluxo de eventos principal de um caso de uso de um software orientado a objetos é o diagrama de

(a). casos de uso.

(b). atividades.

(c). eventos e transições.

(d). classes.

(e). componentes.

Solução:

A UML (Unified Modeling Language) é uma linguagem que permite modelagem de sistemas por meio de diagramas padronizados. Ela especifica tanto o significado gráfico quanto semântico de cada diagrama. A versão 2.0 da UML contempla os seguintes diagramas:

- Diagramas Estruturais
  - Diagrama de Classes
  - Diagrama de Objetos
  - Diagrama de Componentes
  - Diagrama de Instalação
  - Diagrama de Pacotes
  - Diagrama de Estrutura
- Diagramas Comportamentais
  - Diagrama de Caso de Uso
  - Diagrama de Transição de Estados
  - Diagrama de Atividade
- Diagramas de Interação
  - Diagrama de Sequência
  - Diagrama de Interatividade
  - Diagrama de Colaboração ou Comunicação
  - Diagrama de Tempo

Para determinar a alternativa correta da questão, é importante conhecer cada um dos diagramas apresentados como alternativas.

(A) Diagrama de Caso de Uso

Segundo Ivan Jacobson, podemos dizer que um caso de uso é um documento narrativo que descreve a sequência de eventos de um ator que usa um sistema para completar um processo em um sistema. Em outras palavras, o caso de uso é uma técnica de modelagem

usada para descrever como que o sistema deve se comportar mediante a interação com seus diversos usuários que, nesse contexto, também são conhecidos como atores. Os diagramas de caso de uso, por sua vez, são representações gráficas que ilustram o relacionamento entre os três integrantes básicos de um caso de uso, que são os atores, os casos de uso e o sistema.

(B) Diagrama de Atividades

Os diagramas de atividade podem ser vistos como variações dos diagramas de estado, e têm objetivo capturar ações – **passos** que serão executados – e seus resultados em termos das mudanças de estados dos objetos. Em outras palavras, os diagramas de atividades mostram o **fluxo** sequencial das atividades executadas por uma operação específica do sistema. Os principais elementos de um diagrama de atividades são as atividades em si, os pontos de entrada e saída, as decisões, as subdivisões de atividades paralelas (fork), e as junções de atividades (join).

(C) Diagrama de Eventos e Transições

Os diagramas estados (também conhecidos como diagramas de eventos e transições) mostram os diferentes estados de um objeto durante sua vida, bem como os estímulos que acionam as mudanças de estado. Ou seja, os diagramas de estado vêem o ciclo de vida dos objetos como máquinas de estados (autômatos) finitos. O termo finito significa que existe um número finito de estados que o objeto pode assumir, bem como um é finito o número de estímulos que acionam as mudanças de estado.

(D) Diagrama de Classes

O objetivo dos diagramas de classe é descrever os vários tipos de objetos no sistema, bem como o relacionamento entre eles. Sobre os diagramas de classes, é importante ressaltar que eles podem oferecer três perspectivas distintas que são: (i) Conceitual: destinada aos clientes, representa os conceitos do domínio em estudo; (ii) Especificação: destinada às pessoas que não precisam saber detalhes de desenvolvimento, foca as principais interfaces da arquitetura e métodos, porém não como eles serão implementados; (iii) Implementação: destinada ao time de desenvolvimento, aborda vários detalhes de implementação.

(E) Diagrama de Componentes

O diagrama de componente mostram os componentes do software (por exemplo, componentes CORBA, java beans ou seções do sistema que são claramente distintas) e os artefatos de que eles são feitos, como arquivos de código fonte, bibliotecas de programação ou tabelas de bancos de dados relacionais.

Feitas as devidas apresentações dos diagramas apresentados como alternativas, podemos concluir que a resposta correta é a letra B. As palavras passos e fluxos, em destaque na apresentação do diagrama de atividades, foram a chave para a resolução dessa capciosa questão.

50. **Assuntos relacionados:** *UML*, *Diagramas UML*, *OCL - Object Constraint Language*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas Pleno - Processos*
Ano: *2006*
Questão: *38*

Sobre os diagramas da UML 2.0 e as extensões que têm sido aplicadas a ela, são feitas as seguintes afirmativas:

I Na modelagem de negócios são utilizados os seguintes diagramas da UML: de classes, de processos de negócio, de atividades e estados.

II Extensões na UML referem-se a inclusões de elementos nos diagramas existentes que representem os objetivos de um processo, suas entradas e saídas, os eventos que direcionam o processo, os recursos consumidos e a ordem de execução de suas atividades.

III A UML possibilita a implementação de extensões à linguagem através do uso de estereótipos.

IV A descrição de regras de negócios na UML é feita de modo textual e, para implementar regras entre casos de uso podem ser utilizadas constraints, que podem ser expressas via OCL (Object Constraint Language).

Estão corretas apenas as afirmativas:

(a). I e II, apenas.

(b). III e IV, apenas.

(c). I, II e III, apenas.

(d). II, III e IV, apenas.

(e). I, II, III e IV.

Solução:

A Linguagem Unificada de Modelagem (UML – Unified Modelling Language) é uma linguagem de modelagem padronizada de propósito geral na campo da Engenharia de Software. A UML inclui um conjunto de técnicas de notação gráfica pra criar modelos visuais de sistemas de software.

A UML não é uma metodologia de desenvolvimento. Ela é usada para especificar, visualizar, modificar, construir e documentar os artefatos de um sistema de software orientado a objetos. Oferece uma forma padronizada de visualizar a arquitetura de um sistema, incluindo elementos tais como:

- Atores;
- Processos de negócio;
- Componentes (lógicos);
- Atividades;
- Definições de linguagem de programação;
- Esquemas de banco de dados; e
- Componentes de software reutilizáveis.

Os diagramas UML podem representar duas visões de um modelo de sistema: a Visão Estática (ou Estrutural) e a Visão Dinâmica (ou Comportamental). A primeira enfatiza as estruturas estáticas do sistema utilizando objetos, atributos, operações e relacionamentos. A segunda concentra-se no comportamento dinâmico, apresentando colaborações entre os objetos, bem como suas mudanças de estado. Em sua versão 2.0, a UML possuía os seguintes diagramas:

- **Diagramas Estruturais**
  - Diagrama de Classes
  - Diagrama de Objetos
  - Diagrama de Componentes
  - Diagrama de Estrutura Composta
  - Diagrama de Distribuição
- **Diagramas de Comportamento**
  - Diagrama de Caso de Uso
  - Diagrama de Estado (ou Diagrama de Transição de Estados)
  - Diagrama de Atividade
  - Diagramas de Interação
  - Diagrama de Sequência
  - Diagrama de Comunicação (ou Diagrama de Colaboração)
  - Diagrama de Visão Geral de Interação
  - Diagrama Temporal

A UML 2.2 apresenta, adicionalmente, o Diagrama de Perfil (Profile Diagram) e o Diagrama de Pacotes (Package Diagram), ambos diagramas estruturais.

Existem mecanismos de extensão disponibilizados pela UML com o objetivo de proporcionar a personalização de características. São eles: perfis e estereótipos. Um perfil provê um mecanismo genérico de extensão para personalizar modelos UML para alguma plataforma ou algum domínio específico. Tais perfis são definidos utilizando-se estereótipos, definições de tags e constraints que são aplicados para especificar elementos do modelo, tais como classes, atributos, operações (métodos) e atividades. Assim, um perfil é uma coleção de tais extensões que coletivamente personalizam a UML para um domínio (por exemplo, aeroespacial, financeiro) ou plataforma (por exemplo, J2EE, .net) em particular.

A OCL (Object Constraint Language) é uma linguagem declarativa para descrever regras aplicadas a modelos UML. Atualmente, faz parte do padrão UML. A OCL complementa a UML proporcionando expressões livres da ambiguidade característica das linguagens naturais e livres da dificuldade inerente de se utilizar a complexidade matemática.

Apesar de os processos de negócio de um determinado domínio serem modelados pela UML, não existe um diagrama especificamente intitulado “Diagrama de Processos de Negócio”, o que, efetivamente, torna falsa a assertiva do item I da presente questão.

A assertiva II está de acordo com a teoria exposta.

Conforme explanado, a UML utiliza estereótipos para implementar extensões à si mesma. A assertiva III está correta.

A assertiva IV apresenta corretamente o formato de descrição de regras de negócios na UML (modo textual) e também apresenta o componente chave do novo padrão da OMG para transformação de modelos, a OCL.

Portanto, a letra D indica acertadamente os itens II, III e IV como corretos, sendo a resposta para a questão.

51. **Assuntos relacionados:** *UML*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas Pleno - Processos*
Ano: *2006*
Questão: *55*

Fazendo uma comparação entre os recursos disponíveis na UML 1.4 e UML 2.0, conclui-se que a UML 2.0:

(a). acrescentou dois novos diagramas de Interação: o diagrama Visão Geral da Interação e o diagrama de Tempo, sendo que o primeiro combina o fluxo de controle de um diagrama de Atividades com interações e ocorrências de interação e o segundo modela as mudanças de estado por uma linha de tempo para cada objeto em uma interação.

(b). acrescentou três novos mecanismos de extensibilidade: estereótipos, que identificam um conjunto de qualidades que podem aparecer em diversos elementos dos diagramas; tagged values, que permitem acrescentar novos recursos a um elemento do diagrama; restrições, que definem regras para proteger a integridade de um elemento do diagrama.

(c). substituiu o diagrama de Máquina de Estados de Protocolo e o diagrama Statechart, ambos da UML 1.4, pelo diagrama Máquina de Estados, que contém uma série de estados, as transições conectando os estados, o trigger para as transições, as atividades realizadas na execução das transições e as atividades realizadas no decorrer da duração de cada estado.

(d). substituiu o diagrama de Pacotes da UML 1.4 por uma visão combinada dos diagramas de Componentes e de Implantação, em que os componentes são representados por artefatos e os nós e as conexões oferecem os locais de implantação e execução para os artefatos.

(e). substituiu o diagrama de Componentes da UML 1.4 pelo diagrama Estrutura de Composição que modela as partes de uma classe, componente ou colaboração, incluindo os pontos de interação (portas) usados para acessar recursos da estrutura.

Solução:

Esta é uma questão um tanto quanto específica. Contudo, gostando ou não, mesmo que se acredite que ela não mede adequadamente o conhecimento dos candidatos, o fato é que esse tipo de questão quase sempre aparece nas provas. Cabe ao concurseiro buscar uma boa estratégia para aumentar suas chances.

A UML (Unified Modeling Language) é uma linguagem visual e não-proprietária destinada a especificação, documentação e modelagem de sistemas por meio de diagramas padronizados. Vários diagramas são utilizados, sendo que cada um dá ênfase a determinados aspectos, mas todos modelam o mesmo sistema (ou parte dele). Ela especifica tanto o significado gráfico quanto semântico de cada diagrama. Essa linguagem é considerada uma das mais expressivas para modelagem de sistemas. A UML não é um método de desenvolvimento, mas um modelo de linguagem utilizada para que a modelagem de sistemas seja consistente, simples, compreensível e de fácil comunicação com outros sistemas.

Há 14 diagramas na versão 2.0 da UML, sendo que 4 deles são novos. São eles:

- **Diagramas Estruturais**
  - Diagrama de Classes: representa uma vasta gama de elementos estáticos de um modelo, assim como classes e seus tipos, o seu conteúdo e as suas relações;
  - Diagrama de Objetos: descreve objetos, as suas relações em um ponto espcecífico do tempo, tipicamente no caso especial de um Diagrama de Classes ou de um Diagrama de Comunicações;
  - Diagrama de Componentes: descreve os elementos que compõem uma aplicação, sistema ou empresa. São descritos os componentes, as suas relações, interações e as suas interfaces públicas;
  - Diagrama de Instalação: representa a arquitetura de execução do sistema, incluindo nós, ambientes de execução de hardware ou software, assim como o middleware que os liga;
  - Diagrama de Pacotes: representa a forma como os elementos dos modelos estão organizados em pacotes, assim como as dependências entre eles;
  - Diagrama de Estrutura de Composições **(NOVO)**: descreve a estrutura interna de uma classe, componente ou caso de uso, incluindo os pontos de interação destes com outras partes do sistema. É semelhante ao Diagrama de Classes, mas representa partes e conectores. As partes não são necessariamente classes no modelo e não representam instâncias particulares, mas podem representar papéis que os objetos ou instâncias desempenham. As partes são descritas de maneira semelhante aos objetos.
- **Diagramas Comportamentais**
  - Diagrama de Caso de Uso: demostra casos de uso, atores e relações entre eles;
  - Diagrama de Transição de Estados: descreve os estados em que um objeto ou interação pode se encontrar, assim como as transições entre eles;
  - Diagrama de Atividade: descreve processos de negócio de alto nível, incluindo fluxos de dados ou modelagem lógica da complexidade dentro do sistema.
- **Diagramas de Interação**
  - Diagrama de Sequência: modela a lógica sequencial, em especial a ordenação temporal de mensagens entre objetos em um determinado contexto;
  - Diagrama de Interatividade (ou de Vista Geral de Interação) **(NOVO)**: uma variante do Diagrama de Atividades que fornece uma visão geral do fluxo de controle dentro de um processo de sistema ou de negócio;
  - Diagrama de Colaboração ou Comunicação **(NOVO)**: representa instâncias de classes as suas relações e o fluxo de mensagens entre elas. São semelhantes aos Diagramas de Sequência, focam-se principalmente na organização estrutural de objetos que enviam e recebem mensagens. Frequentemente é necessário efetuar uma escolha entre o Digrama de Sequência e o Digrama de Comunicação. Se representar o tempo ou sequência de eventos for o mais importante, deve ser usado o Diagrama de Sequência. Por outro lado, se o objetivo for mostrar o contexto deve ser usado o Diagrama de Comunicação;
  - Diagrama de Tempo **(NOVO)**: descreve as mudanças de um estado ou condição de uma instância de um objeto ou papel ao longo do tempo. Tipicamente usado para mostrar a mudança de estados de um objeto com o decorrer do tempo em resposta a eventos externos. Ele pode ser usado para definir componentes de software e hardware-driven. Esse tipo de diagramas pode ser desenhado com base em um valor ou baseado temporalmente em um ponto de vista específico.

Vamos, de forma bem pragmática, a cada uma das alternativas.

(A) CORRETA

Como se pode observar na explicação acima, os dois diagramas citados nesta alternativa (Visão Geral da Interação e de Tempo) foram realmente acrescentados na versão 2.0 da UML. Perceba também que as descrições feitas estão de acordo. Portanto, é esta a alternativa correta.

(B) ERRADA

Todos os três mecanismos de extensibilidade citados já faziam parte da versão 1.4. Portanto, esta alternativa está errada. Pode-se descrever tais mecanismos da seguinte forma:

- **estereótipos:** permitem a classificação de um elemento de modelo estendido de acordo com um elemento de modelo base já existente na UML e a definição de novas restrições. Todos os elementos estendidos do modelo que possuírem um ou mais estereótipos terão valores adicionais, além dos atributos, associações e super classes que o elemento já possui na UML;
- **restrições:** são regras de restrições bem definidas. Uma restrição permite que uma nova especificação semântica seja atribuída ao elemento do modelo, podendo ser adicionada tanto a um elemento de modelo diretamente quanto a um estereótipo;
- **tagged values:** as tagged values (valores atribuídos ou valores rotulados) permitem explicitar uma certa propriedade de um elemento através de um par: nome, valor. Essas informações podem ser adicionadas arbitrariamente a qualquer elemento do modelo, podendo ser determinado pelo utilizador ou por convenções das ferramentas de suporte.

(C) ERRADA

Não há nem na versão 1.4 nem na versão 2.0 da UML um tipo de diagrama chamado "Máquina de Estados de Protocolo". Isso já é suficiente para que esta alternativa esteja errada. Muito embora a descrição do diagrama de Máquina de Estados (Transição de Estados) esteja correta, dizer que ele substituiu um diagrama inexistente e outro que na verdade é o próprio diagrama de Máquina de Estados (diagrama Statechart) é um grande equívoco.

(D) ERRADA

Como o diagrama de Pacotes faz parte da versão 2.0, a afirmação feita nesta alternativa não faz sentido.

(E) ERRADA

Esta é outra alternativa que se referente a um diagrama que não deixou de fazer parte da UML na sua versão 2.0: diagrama de Componentes. Ou seja, se trata de uma afirmação errada.

52. **Assuntos relacionados:** *Engenharia de Software*, *UML*, *Generalização*,
Banca: *ESAF*
Instituição: *Agência Nacional de Águas (ANA)*
Cargo: *Analista Administrativo - Tecnologia da Informação e Comunicação / Desenvolvimento de Sistemas e Administração de Banco de Dados*
Ano: *2009*
Questão: *16*

A restrição UML aplicada a um conjunto de generalizações, especificando que uma instância pode ter apenas um dos subtipos determinados como tipo daquela instância, é conhecida como

(a). Associação.

(b). Auto.

(c). Global.

(d). Local.

(e). Disjunção.

Solução:

A Unified Modeling Language (UML) é uma linguagem de modelagem e serve para ser utilizada no desenho de software orientado a objetos, sendo o seu uso bastante limitado em outros paradigmas de programação. A fonte principal sobre o UML encontra-se em seu sítio oficial (http://www.uml.org/). Lá é possível ter acesso às especificações atualizadas. Muitos elementos compõem o modelo UML, eles são usados para a criação de diagramas, que representam uma determinada parte ou ponto de vista do sistema.

A UML, na versão 2.0, é composta por treze tipos de diagramas. Eles podem ser classificados como estáticos e dinâmicos. Nos estáticos, ao contrário dos dinâmicos, as características do sistema não mudam com o tempo. Na Figura 20, podemos verificar essa classificação.

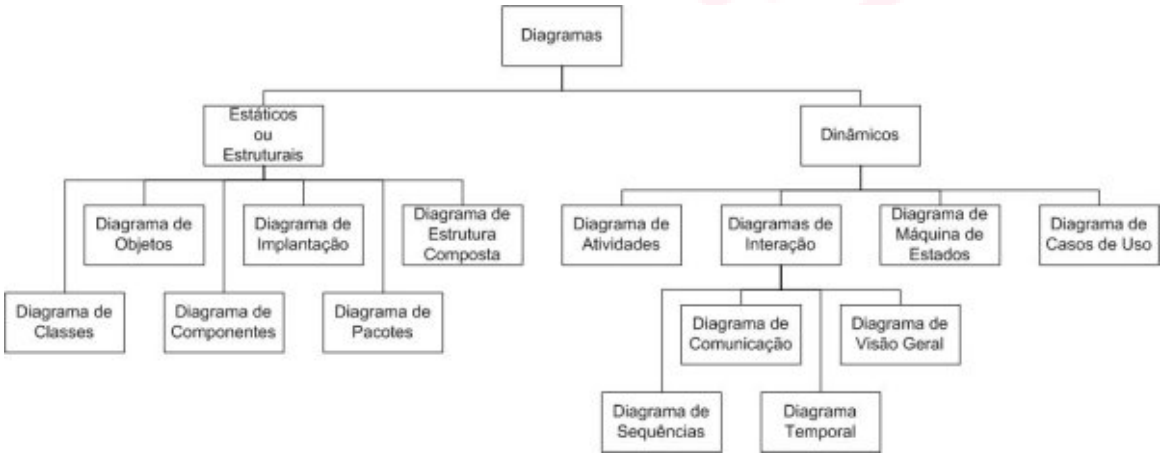


Figura 20: classificação dos diagramas UML.

Os principais conceitos de orientação a objetos (classes, agregação, herança etc.) são representados no diagrama de classes. Ele tem por objetivo descrever, de maneira estática, o escopo da aplicação que está sendo desenvolvida.

Em um diagrama de classe, as associações podem ter quatro tipos de operações:

- **Generalização**: o elemento mais geral (superclasse) se relaciona com um elemento mais específico (subclasse);
- **Agregação**: também conhecido como "parte de" ou composição. Descreve se uma das classes associadas é parte de outra classe;
- **Associação**: duas classes não correlatas se relacionam, representado uma conexão semântica entre os objetos;
- **Dependência**: relação de subordinação entre as classes, onde a mudança na classe independente afetará a classe dependente.

No caso particular da generalização, ponto central da questão, se aplica o conceito de herança. Duas classes são colocadas numa hierarquia sendo uma mais genérica (superclasse) e a outra (subclasse) que herda os atributos e as operações da superclasse, podendo sobrepor ou modificar algumas delas.

A herança pode ser simples ou múltipla. Na múltipla, uma subclasse está associada a mais de uma superclasse. Já na simples, uma subclasse tem uma e somente uma superclasse associada via generalização.

A UML permite que, uma herança possa ser dada em mais detalhes em quatro situações:

- **Superposição**: um objeto da superclasse pode pertencer simultaneamente a mais do que uma subclasse;
- **Disjunção**: não permite a superposição, ou seja, um objeto da superclasse pode pertencer a somente uma subclasse;
- **Completa**: um objeto da superclasse deve pertencer a alguma das subclasses;
- **Incompleta:** um objeto da superclasse pode não pertencer a nenhuma das subclasses.

Vamos exemplificar alguns dos casos acima para que a situação fique mais clara para o leitor. Imagine que temos um tipo chamado *Veículo* com dois subtipos *Carro* e *Caminhão*. Se *Veículo* for uma generalização disjunta de *Carro* e *Caminhão*, uma instância do tipo *Veículo* só pode assumir um dos seus subtipos (*Carro* ou *Caminhão*). Portanto, a alternativa correta é a letra (E). Outro detalhe é que se for uma herança do tipo completa, um veículo necessariamente deve ser um dos seus subtipos.

Já a superposição, podemos exemplificar da seguinte maneira: um tipo chamado *Peça* com dois subtipos chamados *Manufaturada* e *Torneada*. Dado que há um generalização com superposição, será possível que uma peça manufaturada e torneada seja representada como um objeto.

Os outros termos citados nas alternativas (A), (B), (C) e (D) também possuem relação com as representações em UML, mas nada tem a ver com o enunciado da questão.

53. **Assuntos relacionados:** *Linguagem de Modelagem*, *UML*,
Banca: *ESAF*
Instituição: *Controladoria-Geral da União (CGU)*
Cargo: *Analista de Finanças e Controle - Tecnologia da Informação / Desenvolvimento de Sistemas de Informação*
Ano: *2008*
Questão: *31*

A linguagem de Modelagem Unificada (UML) emergiu como notação de diagramação de padrão, de fato e de direito, para a modelagem orientada a objetos. Desta forma, a sentença que conceitua apropriadamente a UML, segundo o OMG-Object Management Group, é

(a). um método para especificar e modelar os artefatos dos sistemas.

(b). um processo de especificação e modelagem de sistemas orientados a objeto.

(c). uma linguagem para implementar os conceitos da orientação a objetos.

(d). uma linguagem visual para especificar, construir e documentar os artefatos dos sistemas.

(e). um método comum para a representação da orientação a objetos.

Solução:

Como o nome sugere, a UML é uma linguagem de modelagem. Oriunda da junção de três métodos de modelagem (Booch, OMT e OOSE), é mantida pela OMG (Object Management Group) e permite que os desenvolvedores visualizem o resultado de seus trabalhos em diagramas padronizados, através de uma notação gráfica bem definida.

A questão envolve, basicamente, a avaliação de terminologias. A UML não pode ser considerada um processo, pois não define etapas de execução. Nem tão pouco pode-se entendê-la como um método ou uma metodologia, já que não existem diretrizes de como projetar ou especificar sistemas. Essa primeira avaliação exclui as assertivas a), b) e e) como alternativas para a questão. A opção c) descreve uma linguagem de programação, o que não é o caso da UML.

O item d) define precisamente a UML: uma linguagem visual para especificar, construir e documentar os artefatos dos sistemas. Deve-se ter em mente que a UML é uma linguagem composta por elementos gráficos a serem utilizados na modelagem dos conceitos do paradigma orientado a objetos.

54. **Assuntos relacionados:** *UML*, *Diagrama de Componentes*, *Diagrama de Colaboração*, *Diagrama de Objetos*, *Diagrama de Atividades*, *Diagrama de Caso de Uso*,
Banca: *ESAF*
Instituição: *Secretaria do Tesouro Nacional (STN)*
Cargo: *Analista de Finanças e Controle - Tecnologia da Informação / Desenvolvimento de Sistemas de Informação*
Ano: *2008*
Questão: *5*

Em UML (Unified Modeling Language), o diagrama cujo foco é a organização estrutural de objetos que enviam e recebem mensagens, exibindo assim, tais objetos e as ligações entre eles, bem como as respectivas mensagens, é o diagrama de

(a). componentes.

(b). colaboração.

(c). objetos.

(d). atividades.

(e). caso de uso.

Solução:

A Linguagem Unificada de Modelagem (UML - Unified Modelling Language) é uma linguagem de modelagem padronizada de propósito geral no campo da Engenharia de Software. A UML inclui um conjunto de técnicas de notação gráfica pra criar modelos visuais de sistemas de software.

A UML não é uma metodologia de desenvolvimento. Ela é usada para especificar, visualizar, modificar, construir e documentar os artefatos de um sistema de software orientado a objetos. Oferece uma forma padronizada de visualizar a arquitetura de um sistema, incluindo elementos tais como:

- Atores;
- Processos de negócio;
- Componentes (lógicos);
- Atividades;
- Definições de linguagem de programação;
- Esquemas de banco de dados;
- Componentes de software reutilizáveis.

Os diagramas UML podem representar duas visões de um modelo de sistema: a Visão Estática (ou Estrutural) e a Visão Dinâmica (ou Comportamental). A primeira enfatiza as estruturas estáticas do sistema utilizando objetos, atributos, operações e relacionamentos. A segunda concentra-se no comportamento dinâmico, apresentando colaborações entre os objetos, bem como suas mudanças de estado. Em sua versão 2.0, a UML possuía os seguintes diagramas:

- Diagramas Estruturais:
  - Diagrama de Classes;
  - Diagrama de Objetos;

  - Diagrama de Componentes;
  - Diagrama de Estrutura Composta;
  - Diagrama de Distribuição.
- Diagramas de Comportamento:
  - Diagrama de Caso de Uso;
  - Diagrama de Estado (ou Diagrama de Transição de Estados);
  - Diagrama de Atividade;
  - Diagramas de Interação;
  - Diagrama de Sequência;
  - Diagrama de Comunicação (ou Diagrama de Colaboração);
  - Diagrama de Visão Geral de Interação;
  - Diagrama Temporal;

Bom, chegou a hora de abordarmos cada diagrama apresentado nas alternativas.

(A) ERRADA

Na verdade, um diagrama de componente mostra os componentes do software (por exemplo, componentes CORBA, Java Beans ou seções do sistema que são claramente distintas) e os artefatos de que eles são feitos, como arquivos de código-fonte, bibliotecas de programação ou tabelas de bancos de dados relacionais.

(B) CORRETA

Este diagrama mudou de nome, agora (UML 2.0) ele é chamado de diagrama de comunicação (o nome já nos sugere que a alternativa está correta).

Ele apresenta as interações existentes entre objetos em uma situação específica, o que o torna um pouco similar ao diagrama de sequência. Enquanto um diagrama de colaboração apresenta mais claramente quais objetos se relacionam entre si, um diagrama de sequência enfatiza como as interações entre objetos ocorrem ao longo do tempo.

Por este diagrama apresentar todas as características do enunciado, esta é a alternativa CORRETA.

(C) ERRADA

O diagrama de objetos é uma variação do diagrama de classes, utilizando inclusive a mesma notação. A única diferença é que o diagrama de objetos mostra os objetos que foram instanciados (os atributos dos objetos possuem valores) das classes. Desse modo, o diagrama de objetos representa um perfil do sistema em um determinado momento de sua execução.

De forma simplificada, podemos entender o diagrama de objetos como uma instância do diagrama de classes.

(D) ERRADA

Um diagrama de atividades é um diagrama de estados em que todos ou a grande maioria dos estados representam execuções de ações (atividades internas ao sistema). Portanto,

ele é um gráfico de fluxo (lembre-se disso, Candidato!) que evidencia os controles e as execuções das atividades. Seus principais elementos são:

- **início**: círculo preenchido;
- **estado ou atividade**: retângulo com bordas arredondadas;
- **transição**: seta;
- **decisão ou desvio**: losango - uma entrada e mais de uma saída;
- **intercalação ou merge**: losango - mais de uma entrada e uma saída;
- **separação ou fork**: barra horizontal - uma entrada e mais de uma saída;
- **junção ou join**: barra horizontal - mais de uma entrada e uma saída.

(E) ERRADA

Diagrama de Caso de Uso é um dos principais diagramas da UML (Unified Modeling Language). Ele descreve, de forma gráfica e intuitiva, relacionamentos e dependências entre um grupo de casos de uso e os atores que interagem com o sistema. O ponto de vista assumido para essa descrição é sempre o dos atores envolvidos.

Lembre-se que um ator é uma entidade externa ao sistema. Ele é representado graficamente por meio de um boneco e um rótulo com o seu nome. Geralmente, é um ator que inicia um caso de uso. Alguns exemplos de atores são: usuários, outros sistemas que fazem interface com o sistema que está sendo modelado e eventos externos.

Já um caso de uso é um conjunto de atividades do sistema que produz um resultado concreto e tangível. Ou seja, ele define uma grande função do sistema. Graficamente, casos de uso são representados por elipses com seus respectivos rótulos.

Resumindo, esse tipo de diagrama descreve interações entre os atores e o sistema em si.

55. **Assuntos relacionados:** *Engenharia de Software*, *UML*, *Hierarquia de Generalização*,
Banca: *ESAF*
Instituição: *Secretaria do Tesouro Nacional (STN)*
Cargo: *Analista de Finanças e Controle - Tecnologia da Informação / Desenvolvimento de Sistemas de Informação*
Ano: *2008*
Questão: *6*

Em hierarquias de generalização UML, a característica na qual uma classe herda tanto propriedades e relacionamentos de sua superclasse imediata quanto de suas superclasses não-imediatas (aquelas em um nível mais alto na hierarquia) é denominada

(a). transitividade.

(b). simetria.

(c). assimetria.

(d). herança múltipla.

(e). associação.

Solução:

A UML (*Unified Modeling Language*) é uma linguagem visual para especificação, construção e documentação de softwares baseados na modelagem orientada a objeto. A UML não é um método de desenvolvimento, mas um modelo de linguagem utilizada para que a modelagem de sistemas seja consistência, simples, compreensível e de fácil comunicação com outros sistemas.

O diagrama de casos de uso é composto por atores, por casos de uso e por relacionamentos. Os atores representam o papel de uma entidade externa ao sistema como um usuário, um hardware, ou outro sistema que interage com o sistema modelado. Os casos de uso são processos ou funções que o sistema deve realizar de forma automática ou manual.

Os relacionamentos podem ocorrem entre casos de usos, entre atores e casos de uso e entre atores. Os tipos de relacionamentos entre casos de uso são a extensão, a inclusão e a generalização. Os tipos de relacionamento entre atores e casos de usos é a associação. E para relacionamento entre atores é a generalização.

No UML, as classes podem ser organizadas em uma hierarquia onde uma classe (superclasse) é uma generalização de uma ou mais classes (subclasses). Uma subclasse herda os atributos e operações da superclasse e pode adicionar novos métodos.

Em uma hierarquia de generalização, as subclasses herdam todas as propriedades de sua superclasse.

A seguir analisamos as alternativas da questão:

(A) CORRETA

Uma das características de hierarquia de generalização é a transitividade, onde uma classe em uma hierarquia herda propriedades e relacionamentos de todos os seus ancestrais. Isto é, herda tanto propriedades e relacionamentos de sua superclasse imediata quanto de suas

superclasses não-imediatas (aquelas em um nível mais alto na hierarquia). Portanto, a alternativa está correta.

(B) ERRADA

O termo simetria não está relacionado a UML. Portanto, esta alternativa está errada.

(C) ERRADA

Outra característica da hierarquia de generalização é a assimetria, onde dadas duas classes A e B, se A for uma generalização de B, então B não pode ser uma generalização de A. Ou seja, não pode haver ciclos em uma hierarquia de generalização.

Esta alternativa está errada, pois não confere com a descrição solicitada no enunciado.

(D) ERRADA

Em uma herança múltipla, uma classe pode ter mais de uma superclasse. Tal classe herda de todas a suas superclasses. O uso de herança múltipla pode levar a conflitos quando atributos e operações de superclasses diferentes tem o mesmo nome e semânticas diferentes.

Esta alternativa está errada, pois não confere com a descrição solicitada no enunciado.

(E) ERRADA

A associação é um mecanismo que permite objetos comunicarem entre si, isto é, um relacionamento que descreve uma série de ligações, onde a ligação é definida como a semântica entre as duplas de objetos ligados.

Esta alternativa está errada, pois não confere com a descrição solicitada no enunciado.

56. **Assuntos relacionados:** *Engenharia de Software*, *Orientação a Objeto*, *Padrões de Projeto*,
Banca: *CESGRANRIO*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Eng. de Software*
Ano: *2008*
Questão: *45*

"Classes devem estar abertas para extensão e fechadas para modificação" é um princípio de projeto de modelos orientados a objetos. Tal princípio pode ser aplicado através do padrão de projeto

(a). decorator.

(b). flyweight.

(c). prototype.

(d). singleton.

(e). builder.

Solução:

A Análise Orientada a Objetos identifica e define classes que refletem diretamente o domínio do problema e as responsabilidades do sistema dentro dele. Normalmente, a análise ocorre com a suposição de que existe uma tecnologia "perfeita" disponível (capacidade ilimitada de armazenamento, custo zero e não passível de falha). Em suma, a análise se interessa pelo que o sistema deve fazer.

O Projeto Orientado a Objetos (*Object-Oriented Design - OOD*) transforma o modelo de análise em um modelo de projeto que serve de base para a construção do software, ou seja, envolve a modelagem de como o sistema será implementado com adição dos requisitos tecnológicos ou não funcionais. Ao contrário dos projetos de softwares convencionais, no projeto orientado a objetos, os componentes principais do sistema são divididos em módulos, chamados subsistemas ou pacotes, e os dados e as operações que os manipulam são encapsulados em classes. Então, o projeto se preocupa como os requisitos serão implementados e, portanto, pressupõe uma infra-estrutura de implementação e é fortemente influenciado pela análise.

O Padrão de Projeto Orientado a Objetos (*Object-Oriented Design Patterns - OOD*) é um dos componentes do projeto orientado a objetos e tem como objetivo registrar uma experiência no projeto de software, na forma de um padrão passível de ser efetivamente utilizado. Isso tem o intuito de reutilizar soluções que funcionaram no passado, evitando que um sistema seja desenvolvido do zero. Ou seja, padrão de projeto está relacionado com a reusabilidade de software.

Um padrão é um par nomeado problema/solução com orientações sobre como utilizá-lo em novas situações e possui quatro elementos essenciais:

- **Nome:** uma identificação que representa o problema, suas soluções e consequências;
- **Problema:** explica o problema de projeto, seu contexto e quando aplicar o padrão;
- **Solução:** descreve os elementos que compõem o projeto, seus relacionamentos, responsabilidades e colaborações. Não descreve um particular projeto concreto ou implementação;

- **Consequências:** são os resultados e os comprometimentos feitos ao se aplicar o padrão.

Um dos principais padrões de projeto é o chamado "Gangue dos Quatro" (*Gang of Four - GoF*). Os padrões GoF são organizados nos grupos:

- **Criativo:** está relacionado ao processo de instanciação (criação) dos objetos, ajudando a tornar um sistema independente de como seus objetos são criados;
- **Estrutural:** trata de como as classes e os objetos são compostos para formar estruturas maiores;
- **Comportamental:** diz respeito a algoritmos e a atribuição de responsabilidades entre objetos.

Um padrão GoF também é classificado segundo o seu escopo: de classe ou de objeto. Nos padrões com escopo de classe os relacionamentos que definem este padrão são definidos através de herança e em tempo de compilação. Nos padrões com escopo de objeto o padrão é encontrado no relacionamento entre os objetos definidos em tempo de execução. A Tabela 2 apresenta os padrões GoF organizados em grupos com os respectivos escopos.

	Grupos		
Escopo	Criativo	Estrutural	Comportamental
Classe	Método-Fábrica	Adaptador (classe)	Interpretador Método Modelo
Objeto	Construtor Fábrica Abstrata Protótipo Singular	Adaptador (objeto) Composto Decorador Fachada Peso-Mosca Ponte Procurador	Cadeia de Responsabilidade Comando Iterador Mediador Memorial Observador Estado Estratégia Visitador

Tabela 2: padrões com os respectivos grupos e escopo.

O princípio "Classes devem estar abertas para extensão e fechadas para modificação" é um dos principais princípios da orientação a objetos, e é conhecido como o princípio Aberto-Fechado (*Open Closed Principle – OCP*). Este princípio quer dizer que devemos estruturar um aplicativo de forma que seja possível adicionar novas funcionalidades com modificações mínimas no código existente. Por exemplo, ao criarmos uma subclasse, nós não devemos alterar o comportamento da classe base, mas apenas completá-lo.

A seguir, analisamos as alternativas da questão.

(A) CORRETA

O padrão de objeto decorator (decorador) do grupo Estrutural anexa responsabilidades adicionais a um objeto dinamicamente, permitindo estender sua funcionalidade. Provê uma alternativa flexível ao uso de herança como modo de estender funcionalidade e permite adicionar responsabilidades a objetos e não a uma classe inteira.

O decorator pode ser utilizado para adicionar responsabilidades dinamicamente a objetos

individuais e transparentemente (sem afetar outros objetos), quando há responsabilidades que podem ser retiradas, quando a herança geraria uma explosão de subclasses e quando a herança seria uma boa alternativa, mas a definição da classe está escondida ou não disponível para herança.

Pode-se pensar também em termos de interface gráfica, por exemplo, adicionar uma borda a um botão, frame, etc. Ao invés de construir uma subclasse para cada objeto que se deseja adicionar a funcionalidade, ele permite uma extensão transparente de funcionalidade durante execução. Para isso ele deve manter a mesma interface do objeto sendo estendido, e deve aceitá-lo como argumento na inicialização para criar uma referência que será mantida como variável membro privada.

O padrão decorator está relacionado em estender funcionalidades do objeto de modo a não modificar a classe inteira. O que mostra que o princípio Aberto-Fechado é aplicado a este padrão. Portanto, alternativa correta.

(B) ERRADA

O padrão de objeto flyweight (peso-mosca) do grupo Estrutural utiliza compartilhamento para suportar um grande número de pequenos objetos de forma eficiente. O flyweight é apropriado quando vários objetos devem ser manipulados, e esses não suportam dados adicionais. No padrão flyweight não existem ponteiros para os métodos do dado, pois isto consome muita memória. Em contrapartida são chamadas sub-rotinas diretamente para acessar o dado.

Um exemplo é o processador de texto. Cada caractere representa um objeto que possui uma família de fonte, um tamanho de fonte e outras informações. Um documento grande com tal estrutura de dados facilmente ocuparia toda a memória disponível no sistema. Para resolver o problema, como muitas dessas informações são repetidas, o flyweight é usado para reduzir os dados, isto é, cada objeto de caractere contém uma referência para outro objeto com suas respectivas propriedades.

O padrão flyweight não está relacionado a estender funcionalidades de uma classe base sem modificá-la, o que mostra que este padrão não é aplicado ao princípio Aberto-Fechado. Portanto, alternativa errada.

(C) ERRADA

O padrão de objeto prototype (protótipo) do grupo Criativo especifica os tipos de objetos que podem ser criados a partir de um modelo original (protótipo) e cria novos objetos copiando este protótipo. Ele utiliza protótipos ao invés de classes e instâncias, reunindo os dois em um só, e ao invés de herança, utiliza decorator para estender funcionalidades.

O padrão prototype pode ser utilizado em sistemas que precisam ser independentes da forma como os seus componentes são criados, compostos e representados. O padrão prototype pode ser útil em sistemas que utilizam classes definidas em tempo de execução.

O padrão prototype não está relacionado a estender funcionalidades de uma classe base sem modificá-la, o que mostra que este padrão não é aplicado ao princípio Aberto-Fechado. Portanto, alternativa errada.

(D) ERRADA

O padrão de objeto singleton (singular) do grupo Criativo garante que uma classe possui uma única instância e provê um ponteiro global para acessá-la. Por exemplo, em uma aplicação que precisa de uma infra-estrutura de log de dados, pode-se implementar uma classe no padrão singleton. Desta forma existe apenas um objeto responsável pelo log em toda a aplicação que é acessível unicamente através da classe singleton.

O singleton é parecido com o flyweight, ambos permitem somente uma instância de um objeto. A diferença é que o singleton permite somente uma instância de uma classe durante o projeto, enquanto o flyweight utiliza factories (fábricas) para produzir somente uma instância em tempo de execução.

O padrão singleton não está relacionado a estender funcionalidades de uma classe base sem modificá-la, o que mostra que este padrão não é aplicado ao princípio Aberto-Fechado. Portanto, alternativa errada.

(E) ERRADA

O padrão de objeto builder (construtor) do grupo Criativo separa a construção de um objeto complexo de sua representação de modo que o mesmo processo de construção pode criar diferentes representações.

Por exemplo, o padrão Builder pode ser utilizado em uma aplicação que converte o formato RTF para uma série de outros formatos e que permite a inclusão de suporte para conversão para outros formatos, sem a alteração do código fonte do leitor de RTF. A solução para este problema é criar uma classe de leitura (director) associada a outra classe capaz de converter o formato RTF para outra representação (builder).

O padrão builder é muitas vezes comparado com o padrão Abstract Factory, pois ambos podem ser utilizados para a construção de objetos complexos. A principal diferença entre eles é que o builder constrói objetos complexos passo a passo e o Abstract Factory constrói famílias de objetos, simples ou complexos, de uma só vez.

O padrão builder não está relacionado a estender funcionalidades de uma classe base sem modificá-la, o que mostra que este padrão não é aplicado ao princípio Aberto-Fechado. Portanto, alternativa errada.

57. **Assuntos relacionados:** *Orientação a Objeto*, *Herança Simples*, *Herança Múltipla*, *Agregação entre Classes*, *Associação entre Classes*,
Banca: *Cesgranrio*
Instituição: *BR Distribuidora*
Cargo: *Analista de Sistemas - Desenvolvimento*
Ano: *2008*
Questão: *63*

Entre os conceitos da modelagem de sistemas orientados a objeto, NÃO se inclui:

(a). herança simples.

(b). herança múltipla.

(c). agregação.

(d). normalização.

(e). associação.

Solução:

São expostos a seguir cada conceito apresentado nas alternativas.

(A) e (B) ERRADAS

Herança é um princípio em que classes compartilham entre si atributos e métodos. Ela permite criação de hierarquia de subclasses (classes derivadas) que herdam características das classes pai (super-classes ou classes base). O mecanismo de herança permite ainda que a subclasse inclua ou sobreponha novos atributos e métodos da superclasse. Os mecanismos existentes são: especialização, especificação, generalização, extensão, limitação e combinação. Três dos grandes benefícios alcançados por esse recurso são: reusabilidade de código, menos custo com manutenção de código e simplificação de modelagem.

Não há mistério na diferenciação entre a herança simples e herança múltipla. Na herança simples, as subclasses podem herdar características somente de uma super-classe. Já na herança múltipla, as subclasses podem herdar atributos e métodos de mais de uma super-classe.

Como sem dúvida nenhuma herança simples e múltipla são conceitos relacionados a modelagem de sistemas orientados a objeto, estas alternativas não devem ser marcadas.

(C) e (E) ERRADAS

Associação, agregação e composição são três tipos de relacionamento entre classes.

Uma ligação representa um relacionamento entre objetos específicos. Já uma associação diz respeito a conexão entre classes. Em uma associação, não há relacionamento de posse entre as classes. As classes são independentes. A Figura 21 ajuda na compreensão desses dois conceitos.

Agregação e composição são variações especiais de associação que modelam relacionamentos do tipo todo-parte. Classes mais complexas agregam ou são compostas de classes mais simples. Esses conceitos são similares. A diferença básica é que uma composição é um tipo mais forte de agregação. Enquanto na composição o todo somente existe se todas as partes existirem, na agregação o todo pode existir com partes inexistentes. Vamos a dois exemplos.

Uma classe TREM pode ser modelada como uma composição entre as classes VAGAO e LOCOMOTIVA. Nessa modelagem, embutiu-se a ideia de que para existir um objeto da classe TREM, é necessário a existência de um objeto da classe LOCOMOTIVA e um objeto da classe VAGAO. Já uma classe FAMILIA pode ser modelada como uma agregação das classes PAI, MAE e FILHO. Nesse caso, a ideia embutida é a de que um objeto da classe FAMILIA pode existir sem a necessidade da existência dos três objetos das classes agregadas. Ou seja, entendeu-se que uma família pode ser composta por pai e filhos ou por mãe e pai, por exemplo.

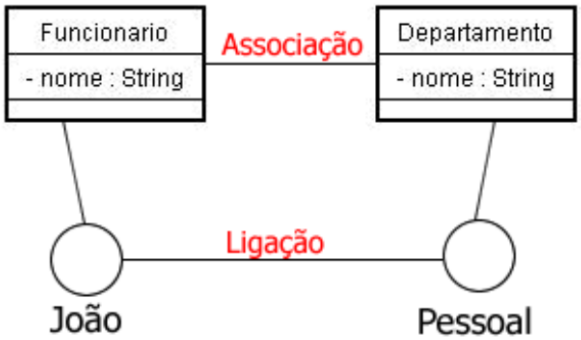


Figura 21: exemplo de ligação e associação.

Como as alternativas C e D trazem de fato conceitos relacionados a modelagem de sistemas orientados a objeto, elas não se encaixam no que se pede no enunciado.

(D) CORRETA

Normalização é um conceito relacionado a banco de dados. Como ele não se encaixa no âmbito de modelagem de sistemas orientados a objeto, é esta a alternativa a ser marcada.

58. **Assuntos relacionados:** *Redes de Computadores*, *Qualidade de Serviço (QoS)*,
Banca: *ESAF*
Instituição: *Superintendência de Seguros Privados (SUSEP)*
Cargo: *Analista Técnico da SUSEP - Tecnologia da Informação*
Ano: *2010*
Questão: *19*

Qualidade de Serviço (Quality of Service - QoS) refere-se

(a). à qualidade do desenvolvimento de sistemas de rede antes de sua entrada em operação.

(b). às garantias de interface estatística entre multiplexadores para evitar perda, retardo, vazão e jitter.

(c). às garantias de estatísticas de roteamento que um sistema de rede pode apresentar com relação a perdas.

(d). às garantias de desempenho estatístico que um sistema de rede pode dar com relação a perda, retardo, vazão e jitter.

(e). à auditoria de desempenho estatístico que um sistema assíncrono pode dar com relação a avanço, retardo, concepção e jitter.

Solução:

Um fluxo de dados é uma sequência de pacotes desde uma origem até um destino. Em uma rede sem conexões (comutação de pacotes), esses pacotes podem individualmente seguir rotas distintas (diferentemente do que ocorre em redes orientadas a conexão, onde a rota é única para todos os pacotes). Quatro parâmetros básicos definem as necessidades de qualidade de transmissão exigidas pelos fluxos: confiabilidade (resiliência a perda), retardo (ou delay), flutuação (ou jitter) e largura de banda (ou vazão). Assim, entende-se por **Qualidade de Serviço** um requisito das diversas aplicações de rede para as quais exige-se a manutenção dos valores destes parâmetros entre níveis mínimos e máximos pré-estabelecidos.

O parâmetro **confiabilidade** refere-se à garantia de transmissão sem perda dos pacotes. Aplicações como correio eletrônico e transferência de arquivos exigem altos níveis de confiabilidade, já que a ausência de alguns pacotes poderá gerar perda de informação. Por outro lado, aplicações de áudio e vídeo por demanda são menos rigorosas nesse ponto, na medida em que a ausência de alguns pacotes pode não prejudicar a experiência em tempo-real desejada pelo usuário. Contudo, o mesmo pode não ocorrer com essa experiência caso o nível de flutuação na transmissão do fluxo cresça. A flutuação refere-se à chegada de pacotes com intervalos de tempo irregulares entre si, o que pode degradar bastante a execução de fluxo de áudio, por exemplo, causando interrupções indesejadas. Aplicações de VoIP (Voz sobre IP) e videoconferência sofrem com a flutuação, porém, em menor nível. O **retardo** (atraso uniforme na transmissão dos pacotes), contudo, prejudica enormemente estas duas últimas categorias de aplicações, pois a sensação de interrupção provocada pela demora na recepção do fluxo áudio-visual prejudica a experiência em tempo-real desejada pelo usuário. As aplicações de correio eletrônico e de transferência de arquivos são "imunes" à flutuação e ao retardo. A **largura de banda**, parâmetro que indica o volume máximo de transmissão de dados por unidade de tempo, é um parâmetro importantíssimo para a transmissão de vídeos (devido à quantidade de informação a ser transmitida por unidade de tempo), sendo um pouco menos expressivo na transmissão de áudio e menos ainda em aplicações de correio eletrônico.

Além disso, como em uma rede sem conexões (caso da Internet) os pacotes podem seguir caminhos distintos desde a origem até o destino, pode ocorrer uma chegada desordenada dessas unidades de transmissão. Essa situação exige um reordenamento no usuário final para a correta entrega de informação à aplicação. O parâmetro **entrega desordenada** (out-of-order ou desequencing) refere-se a essa situação, extremamente prejudicial à aplicações de áudio e vídeo por demanda, por exemplo.

Nas redes de comutação de pacotes, em geral, são definidos três **níveis de serviço**, a saber: melhor esforço, onde não há garantia na entrega de fluxos; serviço diferenciado, que permite definir níveis de prioridade para diferentes tipos de fluxos; e serviço garantido, onde há reserva de recursos de redes para determinados tipos de fluxo.

Pela teoria exposta, observa-se que a opção d) é a única que apresenta-se coerentemente de acordo com o solicitado pela questão, sendo, portanto, a resposta para a mesma.

59. **Assuntos relacionados:** *Programação*, *Orientação a Objeto*, *Análise Orientada a Objetos*,

Banca: *ESAF*
Instituição: *Superintendência de Seguros Privados (SUSEP)*
Cargo: *Analista Técnico da SUSEP - Tecnologia da Informação*
Ano: *2010*
Questão: *26*

São características usadas para seleção de objetos a serem considerados na Análise Orientada a Objetos:

(a). informação retida, múltiplos atributos, requisitos essenciais.

(b). informação produzida, múltiplos atributos, requisitos periféricos.

(c). informações essenciais, múltiplos processos, requisitos essenciais.

(d). informação retida, múltiplos referenciamentos, requisitos top-down.

(e). informação referenciada, atributos numéricos, requisitos essenciais.

Solução:

A **Análise Orientada a Objetos** consiste na definição das classes que representam o problema a ser resolvido. Também deve ser definido a maneira pela qual as classes se relacionam umas com as outras, o funcionamento interno dos objetos (atributos e operações) e os mecanismo de comunicação (mensagens). As características que descrevem um sistema ou produto, devem ser tomadas como base para a definição das características estáticas e dinâmicas dessas classes.

Primeiramente, os casos de uso devem ser criados. Os casos de uso descrevem os cenários de como atores (pessoas, agentes, outros sistemas) interagem com o sistema a ser construído.

A modelagem Classe-Responsabilidade-Colaboração (CRC) transforma as informações contidas nos casos de uso em classes e suas colaborações com outras classes. Uma linguagem unificada de modelagem é utilizada para descrever as características estáticas e dinâmicas das classes.

Basicamente, as tarefas a serem realizadas pela Análise Orientada a Objetos são:

- Identificação de Classes;
- Especificação de Hierarquias de Generalização/Especialização;
- Identificação de Associações e Atributos;
- Modelagem do Comportamento;
- Definição das Operações.

Entre os principais métodos de Análise Orientada a Objetos, nos quais não entraremos em detalhes, podemos citar:

- Método de Booch;
- OMT;
- OOSE (Jacobson);
- Método de Coad & Yourdon;

- Rumbaugh;
- Wirfs-Brock.

Houveram várias tentativas em se definir um método padrão que acabaram sendo dificultadas devido, principalmente, à dificuldade em se definir um processo padrão de análise e projeto. Entretanto, um passo importante já foi dado: a criação da Linguagem de Modelagem Unificada (UML), que é uma notação padrão que pode ser utilizada em diversos métodos de modelagem.

Dada a especificação de requisitos e os diagramas de casos de uso e suas descrições, a modelagem do sistema pode ser iniciada. O processo mais importante em um método orientado a objetos é a descoberta de quais classes devem ser incluídas no modelo.

A expressão inicial do sistema é representada pelas classes de um modelo. O passo inicial na identificação de classes é a revisão e discussão da especificação de requisitos. Como estratégia, a especificação de requisitos pode ser lida procurando por substantivos.

Nem todas as classes devem ser incluídas no modelo. Mo caso da identificação de classes, alguns critérios são essenciais para a inclusão das mesmas durante a Análise Orientada a Objetos:

- informação retida: os atributos são relevantes, ou seja, precisam ser lembrados para o funcionamento do sistema;
- múltiplos atributos: se existe somente um atributo no objeto, provavelmente, o candidato é atributo de outra classe;
- requisitos essenciais: a classe candidata é válida independentemente da tecnologia utilizada;
- serviços necessários: o objeto precisa exibir algum comportamento (processamento);
- vários objetos na classe: deve ser possível identificar vários objetos para a classe candidata;
- atributos comuns: os atributos aplicam-se a todos os objetos da classe;
- serviços comuns: os serviços aplicam-se a todos os objetos da classe;
- resultados não derivados: a classe candidata não pode ser derivada de atributos e serviços de outras classes.

Esse é somente o começo de uma Análise Orientada a Objetos e, no momento, não vamos nos estender às etapas seguintes, pois a questão envolve especificamente a Identificação de Classes (ou Objetos).

De acordo com os três primeiros itens da lista acima, concluímos que a alternativa correta é a letra (A).

60. **Assuntos relacionados:** *Engenharia de Software*, *Testes de Software*, *Complexidade Ciclomática*,
Banca: *CESGRANRIO*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Eng. de Software*
Ano: *2008*
Questão: *55*

Considere o seguinte código de um método de uma classe Java:

```
public boolean primo(int x) {
        if (x == 1 || x == 2) {
            return true;
        }
        int raiz = (int) Math.sqrt((double)x);
        for (int i = 2; i <= raiz; i++) {
            if (x % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

Qual a complexidade ciclomática do método?

(a). 2

(b). 3

(c). 4

(d). 5

(e). 6

Solução:

A complexidade ciclomática, conhecida também como complexidade condicional, é uma métrica de software que fornece uma medida quantitativa da complexidade lógica de um programa estruturado (cíclico). Ou seja, essa métrica mede o número de caminhos linearmente independentes do conjunto básico de um programa, indicando um limite máximo para o número de casos de teste que dever ser executados para garantir que todas as instruções do programa sejam executadas pelo menos uma vez. Um caminho independente é qualquer caminho que introduz pelo menos um novo conjunto de instruções ou uma nova condição.

Existem várias formas de medir a complexidade ciclomática de um programa. Uma forma rápida e prática, que é o que nos interessa em uma prova, é contabilizar o número total de predicados lógicos (comparações expressões booleanas) que aparecem em uma rotina. Seja n esse número, a complexidade ciclomática é dada então por $n + 2$.

No nosso caso, o número total de predicados lógicos é 3 (o primeiro *if (x == 1 || x == 2)*, a condição no *for (int i = 2; i <= raiz; i++)* e o segundo if *(x % i == 0)*). Então, a complexidade ciclomática do procedimento *primo* é igual 5.

61. **Assuntos relacionados:** *Teste de Mesa*, *Java*,
Banca: *CESGRANRIO*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Eng. de Software*
Ano: *2008*
Questão: *65*

```
public class Ponto {
   private int x;
   private int y;

   public Ponto(int x, int y) {
      setCoordenadas(x,y);
   }

   public void setCoordenadas(int x, int y) {
      this.x = x;
      this.y = y;
   }

   public String toString() {
      return "(" + x + "," + y + ")";
   }

   public static void main(String[] args) {
      int a = 1;
      int b = 2;
      int c = 3;
      int d = 4;
      Ponto p = new Ponto(a,b);
      Ponto q = new Ponto(c,d);
      Ponto r = p;
      c = 5;
      p.setCoordenadas(c,d);
      System.out.print(p);
      System.out.print(q);
      System.out.print(r);
      r.setCoordenadas(a,b);
      a = b;
      q.setCoordenadas(b,c);
      System.out.print(p);
      System.out.print(q);
      System.out.println(r);
   }
}
```

Qual será a saída da execução da classe Java acima?

(a). (5,4)(3,4)(5,4)(1,2)(2,5)(1,2)

(b). (5,4)(3,4)(1,2)(5,4)(2,5)(1,2)

(c). (5,4)(5,4)(5,4)(2,2)(2,5)(2,2)

(d). (3,4)(3,4)(5,4)(2,5)(2,5)(1,2)

(e). (3,4)(3,4)(5,4)(2,2)(2,5)(2,2)

Solução:

A seguir, o passo a passo da execução da classe em questão. Na verdade, serão abordados os principais pontos da execução.

- após a execução da 5a linha (Ponto p = new Ponto(a,b);), teremos o seguinte cenário: a=1; b=2; c=3; d=4; p.x=1; p.y=2;
- após a execução da 6a linha (Ponto q = new Ponto(c,d);), teremos o seguinte cenário: a=1; b=2; c=3; d=4; p.x=1; p.y=2; q.x=3; q.y=4;
- após a execução da 7a linha (Ponto r = p;), teremos o seguinte cenário: a=1; b=2; c=3; d=4; p.x=1; p.y=2; q.x=3; q.y=4; r <-> p

Perceba que não foi criado um objeto que seria atribuído à variável r. Na execução dessa última linha, a variável r foi atribuída ao objeto já existente p. Ou seja, a partir deste ponto, em qualquer referência às variáveis r e p serão utilizadas as propriedades do mesmo objeto.

- após a execução da 9a linha (p.setCoordenadas(c,d);), teremos o seguinte cenário: a=1; b=2; c=5; d=4; p.x=5; p.y=4; q.x=3; q.y=4; r <-> p
- após as execuções das linhas de número 10, 11 e 12 (System.out.print(p); System.out.print(q); System.out.print(r);), teremos a seguinte saída: (5,4)(3,4)(5,4)
- após a execução da 13a linha (r.setCoordenadas(a,b);), teremos o seguinte cenário: a=1; b=2; c=5; d=4; p.x=1; p.y=2; q.x=3; q.y=4; r <-> p
- após a execução da 14a linha (a = b;), teremos o seguinte cenário: a=2; b=2; c=5; d=4; p.x=1; p.y=2; q.x=3; q.y=4; r <-> p
- após a execução da 15a linha (q.setCoordenadas(b,c);), teremos o seguinte cenário: a=2; b=2; c=5; d=4; p.x=1; p.y=2; q.x=2; q.y=5; r <-> p
- após as execuções das linhas de número 16, 17 e 18 (System.out.print(p); System.out.print(q); System.out.println(r);), teremos a seguinte saída: (5,4)(3,4)(5,4)(1,2)(2,5)(1,2)

62. **Assuntos relacionados:** *Engenharia de Software*, *Norma de Qualidade de Software*,
Banca: *FCC*
Instituição: *MPU*
Cargo: *Analista de Desenvolvimento de Sistemas*
Ano: *2007*
Questão: *56*

Considere as informações abaixo em relação ao desenvolvimento de sistemas:

I. executar um software com o objetivo de revelar falhas, mas que não prova a exatidão do software.
II. correta construção do produto.
III. construção do produto certo.

Correspondem corretamente a I, II e III, respectivamente,

(a). validação, verificação e teste.
(b). verificação, teste e validação.
(c). teste, verificação e validação.
(d). validação, teste e verificação.
(e). teste, validação e verificação.

Solução:

Os processos de verificação e validação devem ser aplicados ao longo de todo o desenvolvimento do software para o efetivo gerenciamento da qualidade. Esses dois processos, em conjunto com os processos de revisão e auditoria, podem ser consideradas técnicas do processo de garantia da qualidade.

Validação é uma atividade que tem como objetivo assegurar que o produto final corresponda aos requisitos do software (Estamos construindo o produto certo?). Já **Verificação** busca assegurar consistência, completitude e corretitude do produto em cada fase e entre fases consecutivas do ciclo de vida do software (Estamos construindo corretamente o produto?). Finalmente, o **Teste** é uma atividade que tem como objetivo examinar o comportamento do produto através de sua execução. Concluímos, aqui, que a alternativa correta é a letra (C).

Os testes são também conhecidos como **Análise Dinâmica**. Já a **Análise Estática** não envolve a execução propriamente dita do produto. É interessante que a análise estática seja aplicada em qualquer artefato intermediário como, por exemplo, em revisões técnicas e na inspeção de código.

A norma ISO/IEC 12207 foi criada em 1995 com o objetivo de estabelecer uma estrutura comum para os processos do ciclo de vida de um software. Esta norma é dividida em três diferentes classes de processos, que são:

- Processos fundamentais;
- Processo de apoio;
- Processos organizacionais.

Os processos de validação e verificação são considerados processos de apoio na norma ISO/IEC 12207. Eles são usados para garantir a qualidade e não são considerados fundamentais, pois auxiliam outro processo.

63. **Assuntos relacionados:** *Engenharia de Software*, *Testes de Software*,
Banca: *FCC*
Instituição: *TRT 15a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2009*
Questão: *34*

Os testes de integração têm por objetivo verificar se

(a). os módulos testados produzem os mesmos resultados que as unidades testadas individualmente.

(b). os módulos testados suportam grandes volumes de dados.

(c). as funcionalidades dos módulos testados atendem aos requisitos.

(d). os valores limites entre as unidades testadas individualmente são aceitáveis.

(e). o tempo de resposta dos módulos testados está adequado.

Solução:

Uma vez implementado o código de uma aplicação, o mesmo deve ser testado para descobrir tantos defeitos quanto possível, antes da entrega do produto de software ao seu cliente.

O teste é uma atividade de verificação e validação do software e consiste na análise dinâmica do mesmo, isto é, na execução do produto de software com o objetivo de verificar a presença de defeitos no produto e aumentar a confiança de que o mesmo está correto.

A idéia básica dos testes é que os defeitos podem se manifestar por meio de falhas observadas durante a execução do software. Essas falhas podem ser resultado de uma especificação errada ou falta de requisito, de um requisito impossível de implementar considerando o hardware e o software estabelecidos, o projeto pode conter defeitos ou o código pode estar errado. Assim, uma falha é o resultado de um ou mais defeitos.

Um teste, dependendo da estratégia adotada, pode envolver vários estágios. Basicamente há 3 (três) grandes estágios de teste

1. **Teste de Unidade**: tem o objetivo de validar individualmente itens menores (classes ou métodos básicos) da implementação de um caso de uso por meio de testes de caixa-preta (orientado a entrada e saída) e caixa-branca (trabalha diretamente sobre o código fonte). O objetivo é garantir que o item a ser testado cumpre a função para o qual ele foi projetado, adequando-se ao papel que ele desempenha no caso de uso. É interessante observar que o item a ser testado, apesar de menor magnitude, pode depender ou estar relacionado a outros itens. Neste caso, é papel do desenvolvedor testar a integração de seu item com os demais, não se limitando a executar apenas testes isolados;
2. **Teste de Integração**: na fase de teste de integração o objetivo é encontrar falhas provenientes da integração interna dos componentes de um sistema. Geralmente os tipos de falhas encontradas são de envio e recebimento de dados. Por exemplo, um objeto A pode estar aguardando o retorno de um valor X ao executar um método do objeto B, porém este objeto B pode retornar um valor Y, gerando, desta forma, uma falha. Não faz parte do escopo dessa fase de teste o tratamento de interfaces com outros sistemas (integração entre sistemas). Essas interfaces são testadas na fase de teste de sistema (veremos a seguir);

O Teste de Integração pode ser realizado por meio das seguintes abordagens:

- **Integração ascendente ou bottom-up**: nessa abordagem, primeiramente, cada módulo no nível inferior da hierarquia do sistema é testado individualmente. A seguir, são testados os módulos que chamam esses módulos previamente testados. Esse procedimento é repetido até que todos os módulos tenham sido testados;
- **Integração descendente ou top-down**: A abordagem, neste caso, é precisamente o contrário da anterior. Inicialmente, o nível superior (geralmente um módulo de controle) é testado sozinho. Em seguida, todos os módulos chamados por pelo módulo testado são combinados e testados como uma grande unidade. Essa abordagem é repetida até que todos os módulos tenham sido incorporados. Por fim, o sistema inteiro seria testado;
- **Integração sanduíche**: considera uma camada alvo no meio da hierarquia e utiliza as abordagens ascendente e descendente, respectivamente para as camadas localizadas abaixo e acima da camada alvo;
- ***Big-bang***: testa individualmente cada módulo e só depois de testados individualmente os integra.

3. **Teste de Sistema**: tem por objetivo identificar erros de funções (requisitos funcionais) e características de desempenho (requisito não funcional) que não estejam de acordo com as especificações. Os testes de sistema incluem diversos tipos de teste, realizados na seguinte ordem:
   - **Teste funcional**: verifica se o sistema integrado realiza as funções especificadas nos requisitos;
   - **Teste de desempenho**: verifica se o sistema integrado atende aos requisitos não funcionais do sistema (eficiência, segurança, confiabilidade etc);
   - **Teste de aceitação**: os clientes testam o sistema a fim de garantir que o mesmo satisfaz suas necessidades;
   - **Teste de instalação**: ocorre quando o teste de aceitação é feito em um ambiente de teste diferente do local em que será instalado.

De posse deste conhecimento, podemos descarta a alternativa **A**, pois se um módulo produzisse o mesmo valor que as suas unidades, a sua existência seria desnecessária. Podemos descartar, também, as alternativas **B** e **E**, pois dizem respeito ao Teste de Sistema. Além disso, uma vez que o teste de valor limite é efetuado no Teste de Unidade, podemos descartar a alternativa **D** também. Portanto, ao final do teste de integração, é possível assegurar que as funcionalidades do módulo testado atendem aos requisitos (alternativa **C** está correta).

64. **Assuntos relacionados:** *Engenharia de Software*, *Testes de Software*,
Banca: *FCC*
Instituição: *TRT 18a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2008*
Questão: *36*

Uma sistemática para construção da arquitetura do software enquanto, ao mesmo tempo, conduz ao descobrimento de erros associados às interfaces é a estratégia de teste de software denominada de

(a). sistema.

(b). unidade.

(c). validação.

(d). arquitetura.

(e). integração.

Solução:

O primeiro passo para a resolução da questão é compreender do que se trata o termo "interfaces". As interfaces são os pontos do software em que um componente interage com outro, ou pontos em que o sistema como um todo interage com outros sistemas. Portanto, os testes das interfaces do software testam a integração entre os diversos componentes do sistema, e a integração do sistemas com outros.

Com isso, a resposta da questão é a alternativa E, testes de integração. Agora, vamos saber um pouco mais sobre os tipos de testes apresentados nas demais alternativas.

- Testes de Sistema: O objetivo dos testes de sistema é executar o sistema sob ponto de vista de seu usuário final em busca de falhas. Os testes são executados em condições similares - de ambiente, interfaces e de dados - àquelas as quais o usuário estará submetido ao utilizar o sistema;
- Testes de Unidade: As "unidades" podem ser entendidas como sendo as menores unidades de software desenvolvidas, ou como partes ou módulos do sistema. Por este motivo, os testes unitários também são conhecidos como testes de módulo;
- Testes de Validação: De acordo com a definição apresentada pelo CMMI, a validação é o processo de avaliar se o software satisfaz os requisitos especificados. A validação está intimamente ligada a outra etapa, também definida pelo CMMI, chamada Verificação. A verificação, por sua vez, visa assegurar que o software está sendo construído de acordo com a especificações.

Por fim, vamos discutir um pouco a alternativa D, arquitetura. O termo arquitetura de software remete, primordialmente, aos estudo dos componentes do software, das suas propriedades externas, e de seus relacionamentos com outros softwares.

Os "componentes" do software, como já vimos, são testados individualmente nos testes unitários. A integração entre eles, por sua vez, é testada nos testes de integração. Ou seja, testar a arquitetura deverá envolver, minimamente, testar os componentes do software e sua integração.

Nesse ponto, esperamos que o candidato já tenha despertado para a amplitude do termos

arquitetura. Porém, para reforçar ainda mais este fato, vamos ver o que as normas oficiais falam sobre o tema.

Na Ontologia estabelecida pela ANSI/IEEE 1471-2000, a arquitetura de software é normalmente organizada em "visões". As visões são definidas como instâncias de pontos de vista, servindo para descrever a arquitetura na perspectiva de um conjunto de stakeholders. Exemplos de visões são:

- Visão funcional (relacionada às funcionalidades e lógica do software)
- Visão de código (relacionada aos códigos fonte do software)
- Visão de desenvolvimento (relacionada a estrutura do software)
- Visão de concorrência (relacionada aos processos e threads do software)
- Visão física (relacionada a evolução do software)
- Visão de ação do usuário (relacionada ao feedback dos usuários)

Concluindo, a arquitetura de software é uma disciplina bem ampla e genérica que permeia inúmeros aspectos do processo de desenvolvimento e uso do software. Além disso, o caráter genérico e abstrato do assunto é reforçado pelo fato do tema ser abordado por uma Ontologia que, ainda por cima, pode ser considerada recente.

65. **Assuntos relacionados:** *Engenharia de Software*, *Testes de Software*,
Banca: *FCC*
Instituição: *TRT 18a Região*
Cargo: *Analista Judiciário - Tecnologia da Informação*
Ano: *2008*
Questão: *37*

NÃO se trata de uma categoria de erros encontrados por meio de teste caixa-preta:

(a). Conjunto básico de caminhos de execução.

(b). Funções incorretas ou omitidas.

(c). Acesso à base de dados externa.

(d). Comportamento ou desempenho.

(e). Iniciação e término.

Solução:

Os métodos de teste caixa-preta concentram-se nos requisitos funcionais do software. Em outras palavras, nos testes caixa-preta o testador não está interessado em como as entradas são processadas, mas apenas se as saídas são coerentes com os dados fornecidos na entrada.

Os principais objetivos dos testes caixa-preta são descobrir erros de interface, erros nas estruturas de dados, erros de acesso a banco de dados externos, erros de desempenho, erros de inicialização e de término, e também detectar funções incorretas ou ausentes.

Já os erros relacionados aos caminhos de execução só podem ser detectados por testes caixa-branca. Com isso, a resposta da questão é a alternativa A.

Sobre os testes de caixa-preta, ainda é importante ressaltar que eles geralmente são aplicados durante as últimas etapas da atividade de teste, concentrando-se no domínio das informações. Alguns exemplos de métodos de testes caixa-preta são mostrados a seguir:

- Particionamento de equivalência: Esse método consiste em dividir o domínio de entrada de um programa em classes de dados a partir das quais os casos de teste podem ser derivados. O particionamento de equivalência procura definir casos de testes que descubram classes de erros, reduzindo o número total de teste que devem ser realizados;
- Análise de valor limite: O objetivo do análise do valor limite é por a prova os valores fronteiriços das entradas de dados, tendo em vista o fato de que a maior parte dos erros tende a ocorrer nas fronteiras do domínio de entrada;
- Técnicas de grafo de causa-efeito: O grafo de causa-efeito é uma técnica de projetos de caso de teste que oferece uma uma representação concisa das condições lógicas e das ações correspondentes;
- Testes de comparação: Os testes de comparação, também conhecidos como testes *back-to-back*, são utilizados quando a confiabilidade do software é absolutamente crítica. Para aplicar a técnica, defende-se que versões independentes de software sejam desenvolvidas para as aplicações. O método de comparação não é infalível, pois se a especificação a partir da qual todas as versões foram desenvolvidas estiver errada, provavelmente todas as versões refletirão o erro. Além disso, se cada uma das versões independentes produzir resultados idênticos, mas incorretos, os testes de condições deixarão de detectar o erro.

66. **Assuntos relacionados:** *Testes de Software*, *Complexidade Ciclomática*,
Banca: *Cesgranrio*
Instituição: *BR Distribuidora*
Cargo: *Analista de Sistemas - Desenvolvimento*
Ano: *2008*
Questão: *60*

Para garantir a cobertura de todos os comandos de um programa, é possível encontrar o limite superior para o número de testes que precisam ser projetados e executados, realizando a(o)

(a). contagem do número de linhas de um programa.

(b). cálculo do total dos ciclos do programa.

(c). cálculo do total de resultados diferentes de testes de caixa preta.

(d). cálculo da complexidade ciclomática do grafo que representa a estrutura lógica do programa.

(e). cálculo do número de partições de equivalência do grafo que representa a estrutura lógica do programa.

Solução:

A resposta da questão é a alternativa D. Introduzida em 1976 por Thomas McCabe, a complexidade ciclomática mede o número de caminhos linearmente independentes ao longo de um grafo. Em outras palavras, a complexidade ciclomática pode ser interpretada como uma estimativa do número de decisões sobre um grafo.

No contexto da ciência computação, a complexidade ciclomática é uma métrica utilizada para medir a complexidade de processos e de unidades lógicas (métodos, classes, funções etc) de um software. Quanto maior for a complexidade ciclomática, maior será a complexidade do software.

No âmbito dos testes de software, medir a complexidade ciclomática significa determinar o número de casos de teste mínimo para testar adequadamente todos os caminhos independentes do programa. Existem inúmeras variantes da técnica de complexidade ciclomática, porém, vamos nos ater a essência da técnica.

Para compreender melhor como calcular a complexidade ciclomática, vamos apresentar primeiro uma ferramenta de apoio ao cálculo, chamada grafos de fluxos de execução. Os grafos de fluxo de execução são representações gráficas que descrevem o fluxo de controle de um programa, como ilustrado na Figura 22.

Nesses grafos, cada desvio possível no programa é mostrado como um caminho separado, e os loops são representados com o uso de setas que retornam ao nó de condição do loop. A partir do grafo, é possível calcular a complexidade ciclomática pela seguinte fórmula:

$$\text{Complexidade Ciclomática} = \text{Número de Arestas} - \text{Número de Nós} + 2$$

Obviamente, pelo fato do exemplo mostrado na figura ser bem simples, foi possível desenhar o grafo manualmente e medir a complexidade ciclomática. Na prática, em softwares maiores, é impossível calcular a complexidade ciclomática sem a ajuda de uma ferramenta especializada.

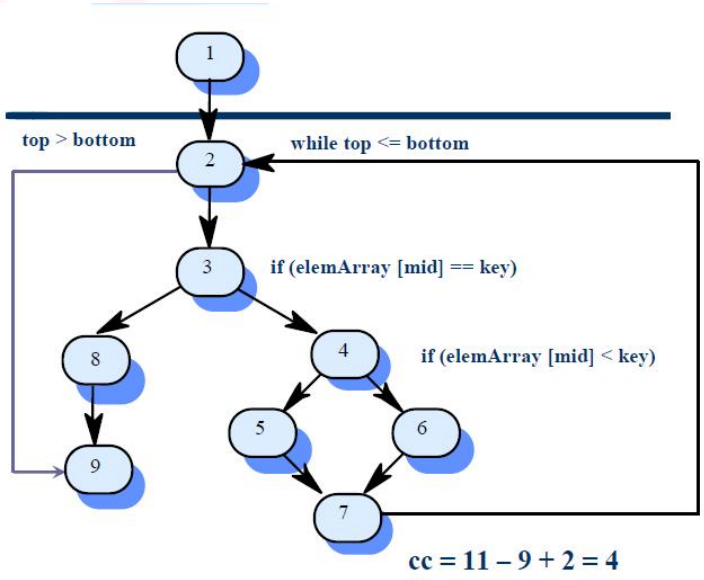


Figura 22: grafo de fluxo de execução.

67. **Assuntos relacionados:** *Engenharia de Software*, *Casos de Testes*, *Partições de Equivalência*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas - Processos de Negócio*
Ano: *2008*
Questão: *54*

Um importante aspecto da elaboração de casos de testes para um sistema em desenvolvimento é a escolha dos valores de entrada e das saídas previstas dos casos de teste. Escolhas baseadas apenas em valores típicos, em geral, são incapazes de revelar todas as falhas da implementação. É necessário identificar conjuntos de valores que possuam características comuns, do ponto de vista das funcionalidades a serem testadas, como, por exemplo, "números negativos", "números com mais dígitos do que o previsto", "strings sem branco", "arrays de um são elemento", além de prever casos de teste cobrindo a totalidade destes conjuntos, e projetar, para cada conjunto, casos de teste com valores nos limites e próximos ao ponto médio do conjunto. Esses conjuntos são denominados

(a). partições de equivalência.

(b). grupos de controle.

(c). espaços amostrais.

(d). classes características.

(e). intervalos de testes.

Solução:

Na engenharia de software, casos de teste é um conjunto de condições usadas para teste de software. Ele pode ser elaborado para identificar defeitos na estrutura interna do software, através de situações que exercitem adequadamente todas as estruturas utilizadas na codificação; ou ainda, garantir que os requisitos do software que foram construídos sejam plenamente atendidos. Podemos utilizar a ferramenta de casos de uso para criar e rastrear um caso de teste, facilitando assim a identificação de possíveis falhas. O caso de teste deve especificar a saída esperada e os resultados esperados do processamento. Numa situação ideal, no desenvolvimento de casos de teste, se espera encontrar o subconjunto dos casos de testes possíveis com a maior probabilidade de encontrar a maioria dos erros.

As partições de equivalência são um método de teste que divide o domínio de entrada de um programa em classes de equivalência a partir das quais os casos de teste podem ser derivados. O particionamento de equivalência procura definir um caso de teste que descubra classes de erros, assim reduzindo o número total de casos de teste que devem ser desenvolvidos. O projeto de casos de teste para particionamento de equivalência baseia-se numa avaliação de classes de equivalência para uma condição de entrada. Uma classe de equivalência representa um conjunto de estados válidos ou inválidos para condições de entrada. Uma condição de entrada é um valor numérico, um intervalo de valores, um conjunto de valores relacionados ou uma condição booleana. Ou seja, o modelo deve compreender partições de valores de entrada e saída. Cada partição deve conter um intervalo de valores, escolhidos de tal maneira, que todos os valores nessa classe têm o mesmo resultado, daí o nome equivalência. Podem ser usados valores válidos e inválidos para serem candidatos à classe de equivalência. É desejável se ter pelo menos um caso de teste para exercitar cada uma das classes de equivalência para se ter uma máxima cobertura da aplicação.

68. **Assuntos relacionados:** *Engenharia de Software*, *Testes de Software*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas Pleno - Processos*
Ano: *2006*
Questão: *54*

Uma estratégia de teste de software integra métodos de projeto de casos de teste em uma série bem planejada de passos, que resultam na construção bem sucedida de um software. O objetivo principal do projeto de casos de teste é originar um conjunto de testes que tenha a maior probabilidade de detectar erros no software. Sobre as estratégias e técnicas de teste de software, assinale a afirmativa correta.

(a). O teste de caixa-preta enfoca a estrutura de controle do programa, tendo como exemplos o teste de caminho básico, que faz uso de grafos de para originar um conjunto de testes linearmente independentes que vão garantir a cobertura e a análise de valor-limite, que investiga a habilidade do programa de manipular dados no limite de aceitabilidade.

(b). O teste de caixa-branca são projetados para validar os requisitos funcionais de funcionamento interno de um programa, tendo como exemplos o particionamento de equivalência, que divide o domínio de entrada em classes de dados que provavelmente exercitam função específica do software e o teste de matriz ortogonal, que fornece um método eficiente e sistemático para testar sistemas com pequeno número de parâmetros de entrada.

(c). O teste de integração focaliza o esforço de verificação na menor unidade de projeto do software e, usando a descrição de projeto no nível de componente como guia, caminhos de controle importantes são testados para descobrir erros dentro dos limites do módulo. O teste fumaça é um exemplo de abordagem de teste de integração.

(d). O teste de recuperação é um teste de sistema que força o software a falhar de diversos modos e verifica se a recuperação é adequadamente realizada, seja ela feita de forma automática (realizada pelo próprio sistema) ou requerendo intervenção humana.

(e). A fase alpha de testes é realizada ainda no processo de desenvolvimento, nas instalações do desenvolvedor com os usuários finais e utilizando um ambiente controlado, enquanto a beta é realizada entre o término do desenvolvimento e a entrega do produto, nas instalações do desenvolvedor com um ambiente controlado pelo usuário final. Na fase gama são gerados pela equipe de desenvolvimento casos de teste que são realizados por grupos restritos de usuários finais.

Solução:

O objetivo principal das técnicas de software é encontrar falhas no software. As técnicas mais conhecidas são os testes de caixa-branca, de caixa-preta, de performance, de usabilidade, de carga, de stress, de confiabilidade e de recuperação.

Nos testes de caixa-branca, o testador tem acesso ao código fonte da aplicação e os casos de testes são desenvolvidos analisando-se o código fonte e procuram cobrir todas as possibilidades do componente de software. A técnica de teste de caixa-branca é recomendada para as fases de testes unitários e de testes de integração. O teste do caminho básico, o teste de loops (laços) e o teste de estruturas de controles são tipos de teste da caixa branca.

Já o teste de caixa-preta não leva em consideração o comportamento interno do componente. Neste caso, os dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado a um resultado esperado. Tal técnica é recomendada para qualquer fase de testes: testes unitários, testes de integração, testes de sistema e testes de aceitação. Exemplos de técnicas de teste de caixa-preta são os grafos de causa-efeito, testes baseados em tabela de decisão e teste do valor limite.

Os testes de recuperação têm o objetivo validar o comportamento do software após a ocorrência de erros ou condições anormais, realizando a recuperação dos dados ou ainda a permanência do software em execução.

Os testes também são divididos em fases. Na fase de testes unitários, as menores unidades de software desenvolvidas são testadas. Na fase de testes de integração, o objetivo é encontrar falhas provenientes da integração dos componentes de um sistema. Já os testes de sistema, visam cobrir as funcionalidades do sistema, mas sob ponto de vista do usuário final. Nos testes de aceitação, um grupo reduzido de usuários finais do sistema simulam operações do mesmo para verificar se o funcionamento está de acordo com o que foi pedido.

Em alguns casos, os testes requerem que fases de testes especiais sejam executadas antes do produto ser disponibilizado a todos os usuários. O período entre o término do desenvolvimento e a entrega é conhecido como fase alpha e os testes executados nesse período, como testes alpha. PRESSMAN afirma que o teste alpha é conduzido pelo cliente no ambiente do desenvolvedor, com este "olhando sobre o ombro" do usuário e registrando erros e problemas de uso.

Já ao final da fase alpha, as versões beta podem ser lançadas a grupos restritos de usuários. Os testes da fase beta também podem ser considerados como testes de aceitação, mas voltados para softwares cuja distribuição atingirá um grande número de usuários.

Analisando cada uma das alternativas, podemos notar que a alternativa (A) está errada. O teste de caminho básico é um teste de caixa-branca, pois o seu passo inicial consiste em determinar todos os caminhos possíveis através do fluxo de controle do programa, obtido através de uma análise do código-fonte. Já a análise de valor-limite procura estabelecer casos de testes nos quais os valores de limites extremos serão avaliados, tanto os da entrada quanto os da saída. Além disso, essa análise não leva em conta a estrutura interna do programa e pode ser considerada um técnica de teste de caixa-preta. Entretanto, a alternativa continua invalidada, visto que a primeira técnica de exemplo exemplo é um teste de caixa-branca.

Analisando a alternativa (B), podemos notar que, no particionamento de equivalência, a entrada de dados é dividida em classes que serão testadas a partir de um caso de uso específico. Sendo assim, muitos casos de testes redundantes podem ser eliminados. Entretanto, é mais um exemplo de técnica de teste caixa-preta, visto que não procura analisar o funcionamento interno do programa. A técnica de teste de matriz ortogonal também é uma técnica de teste de caixa-preta. Ela pode ser aplicada a problemas nos quais o domínio de entrada é relativamente pequeno, mas grande demais para acomodar teste exaustivo. A abordagem de teste de matriz ortogonal nos possibilita obter boa cobertura de teste com muito menos casos de teste que a estratégia exaustiva. Os dois exemplos de técnicas de teste são de caixa-preta, portanto a alternativa (B) está incorreta.

O teste de integração nada tem a ver com o que foi dito na alternativa (C), pois são os testes unitários que são responsáveis em verificar as menores unidades do projeto e, por isso, a alternativa (C) também está errada. Entretanto, o teste fumaça é um exemplo de teste de integração, pois consiste em um teste rápido, executando as principais funcionalidades do sistema e sua execução é recomendada na frequência diária durante o desenvolvimento, visto que as modificações em unidades de projeto podem afetar as funcionalidades principais.

Nos testes de recuperação, o software é forçado a falhar de diversas maneiras para que seja verificado o seu comportamento. Se a recuperação for automática, os processos de recuperação e reinício são avaliados quanto à correção. Já se a recuperação exigir intervenção humana, o tempo de reparo é avaliado para determinar se ele se encontra fora dos limites aceitáveis. A alternativa (D) está correta.

Podemos notar as seguintes falhas na alternativa (E): os testes da fase beta não são realizados no ambiente do desenvolvedor, mas, sim, no ambiente do usuário e para um grupo restrito. O teste gama é um termo utilizado de forma sarcástica referindo-se aos produtos que são mal testados e são entregues aos usuários finais para que estes encontrem os defeitos já em fase de produção e não o que foi dito na alternativa (E).

Podemos concluir, que a alternativa (D) deve ser marcada, pois é a única correta.

69. **Assuntos relacionados:** *Engenharia de Software*, *Teste de Usabilidade*,
Banca: *Cesgranrio*
Instituição: *Petrobras*
Cargo: *Analista de Sistemas Júnior - Processos de Negócio*
Ano: *2008*
Questão: *61*

Uma das técnicas empregadas no projeto de interfaces de sistemas é a condução de testes de usabilidade, cujos resultados fornecem importantes indicadores para melhorar a qualidade da interface. Os testes de usabilidade consistem em

(a). apresentar o sistema para um grupo de foco e coletar a opinião dos participantes sobre a interface, os conceitos e as metáforas utilizadas na mesma.

(b). apresentar, para um usuário por vez, um protótipo da interface do sistema ou o próprio sistema, e solicitar que o usuário realize algum tipo de tarefa, observando suas reações à interface, erros cometidos, dificuldades e eficiência no cumprimento da tarefa.

(c). distribuir um questionário para os usuários iniciais do sistema com perguntas sobre a interface e mapear as respostas.

(d). submeter o sistema a um software robot de teste e verificar os tempos de resposta a cada padrão de navegação, comparando-os com os requisitos não funcionais do sistema.

(e). utilizar um software robot de teste para fazer acessos aleatórios ao sistema, tentando usar elementos da interface e registrando os erros encontrados.

Solução:

Segundo a definição da norma ISO 9241-11, usabilidade é "a extensão em que um produto pode ser usado por usuários específicos para alcançar objetivos específicos com eficácia, eficiência e satisfação num contexto específico de uso". Ou seja, a usabilidade é um conceito utilizado para descrever a qualidade da interação (eficácia, eficiência) de uma interface de um sistema diante de seus usuários (reação do usuário). Essa qualidade está associada aos seguintes princípios:

- facilidade de aprendizado;
- facilidade de memorização de tarefas no caso de uso constante;
- produtividade dos usuários na execução de tarefas;
- prevenção, visando a redução de erros por parte do usuário;
- satisfação subjetiva do usuário.

Conforme a estrutura mostrada na Figura 23, para verificar a usabilidade é preciso identificar os objetivos, e decompor a usabilidade (eficácia, eficiência e satisfação) em atributos passíveis de serem verificados e mensurados, assim como o contexto de uso (usuário, tarefa, equipamento e ambiente).

O teste de usabilidade é um processo onde participantes representativos (usuários representando uma população alvo) avaliam, de acordo com critérios específicos de usabilidade, em que grau uma interface de um sistema se encontra.

O teste de usabilidade busca encontrar problemas de usabilidade e fazer recomendações

no sentido de eliminar os problemas, melhorando a usabilidade da interface. Os testes de usabilidade são mais eficientes quando implementados como parte de um ciclo de vida de desenvolvimento de um sistema.

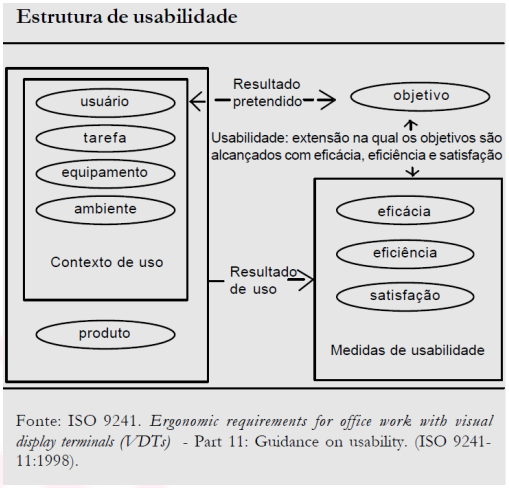


Figura 23: estrutura de usabilidade.

Os tipos de teste de usabilidade são:

- Teste de Exploração: realizado quando uma interface encontra-se em estágio pré-liminar de definição e desenho. O objetivo é avaliar a efetividade do desenho pré-liminar e conhecer a concepção do usuário ou modelo mental da interface.
- Teste de Avaliação: o mais comum, podendo ser empregado no início ou meio do ciclo de desenvolvimento do sistema. O objetivo é expandir o que foi alcançado no teste de exploração. Neste teste, o usuário executa tarefa bastante simples caminhando entre telas, onde é dado mais ênfase ao comportamento;
- Teste de Validação: realizado mais tarde no ciclo de desenvolvimento. Este teste certifica a usabilidade da interface bem próximo a sua liberação, verificando padrões de usabilidade, de desempenho e históricos;
- Teste de Comparação: não é realizado dentro um ciclo de desenvolvimento do sistema. Este teste pode ser utilizado para comparar as diferenças entre estilos de interface, medir a efetividade de um elemento integrante da interface, e como a liberação de um sistema atinge um produto concorrente.

Com base nas explicações anteriores, a seguir analisamos as alternativas:

(A) ERRADA

O teste de usabilidade envolve coletar opiniões dos participantes por meio de questionários a respeito da facilidade de aprendizado, da facilidade de memorização de tarefas, da produtividade dos usuários na execução de tarefas, da prevenção de erros e da satisfação usuário em uma interface, isto é, opiniões dos participantes sobre os princípios da usabilidade.

Os conceitos utilizados e as metáforas na interface são utilizados como meio para proporcionar os princípios de usabilidade, e não para serem avaliados diretamente pelo usuário. Portanto, a alternativa está errada.

(B) CORRETA

O teste de usabilidade pode ser aplicado a um usuário por vez, apesar de ter um custo elevado. No teste pode ser apresentado um desenho inicial ou uma versão pré-liminar do produto, ou um protótipo ou como um sistema como todo.

No teste de usabilidade, o usuário pode realizar tarefas a fim de se verificar as reações à interface, os erros cometidos, a dificuldades e eficiência da tarefa no sentido de eliminar os problemas encontrados e melhorar a usabilidade do produto. Logo, esta afirmativa está correta.

(C) ERRADA

A elaboração de um questionário para ser respondido pelos usuários pode ser aplicado no teste de usabilidade. Entretanto, o teste de usabilidade é aplicado a um grupo representativo de usuários que farão uso do produto, isto é, o teste é realizado com usuários finais do produto, e não os usuários iniciais. Portanto, esta alternativa está errada.

(D) ERRADA

O teste de usabilidade tem como objetivo avaliar os princípios de usabilidade de uma interface utilizando participantes representativos. Como o teste existe a presença de um ou mais participantes representativos do sistema, em testes de usabilidade não é possível a utilização de software robot. O software robot pode ser utilizado para verificar a acessibilidade de um produto.

Os testes para verificar os tempos de resposta a cada padrão de navegação, comparando-os com os requisitos não funcionais do sistema estão relacionados a teste de acessibilidade. Logo, esta alternativa está errada.

(E) ERRADA

Esta alternativa está errada. Vide explicação da alternativa (D).

Questão	Resposta
1	A
2	B
3	D
4	B
5	E
6	E
7	A
8	C
9	D
10	B
11	C
12	B
13	C
14	D
15	E
16	E
17	E
18	A
19	D
20	D
21	B
22	D
23	B
24	D
25	E
26	C
27	B
28	B
29	B
30	E
31	C
32	A
33	D
34	D
35	B
36	E
37	B
38	E
39	C
40	B
41	E
42	C
43	E
44	D
45	B
46	D
47	A
48	E
49	B
50	D
51	A
52	E
53	D
54	B
55	A

Questao	Resposta
56	A
57	D
58	D
59	A
60	D
61	A
62	C
63	C
64	E
65	A
66	D
67	A
68	D
69	B

Índice Remissivo