UNIP
UNIVERSIDADE PAULISTA

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 8

CQA - COMISSÃO DE QUALIFICAÇÃO E AVALIAÇÃO

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

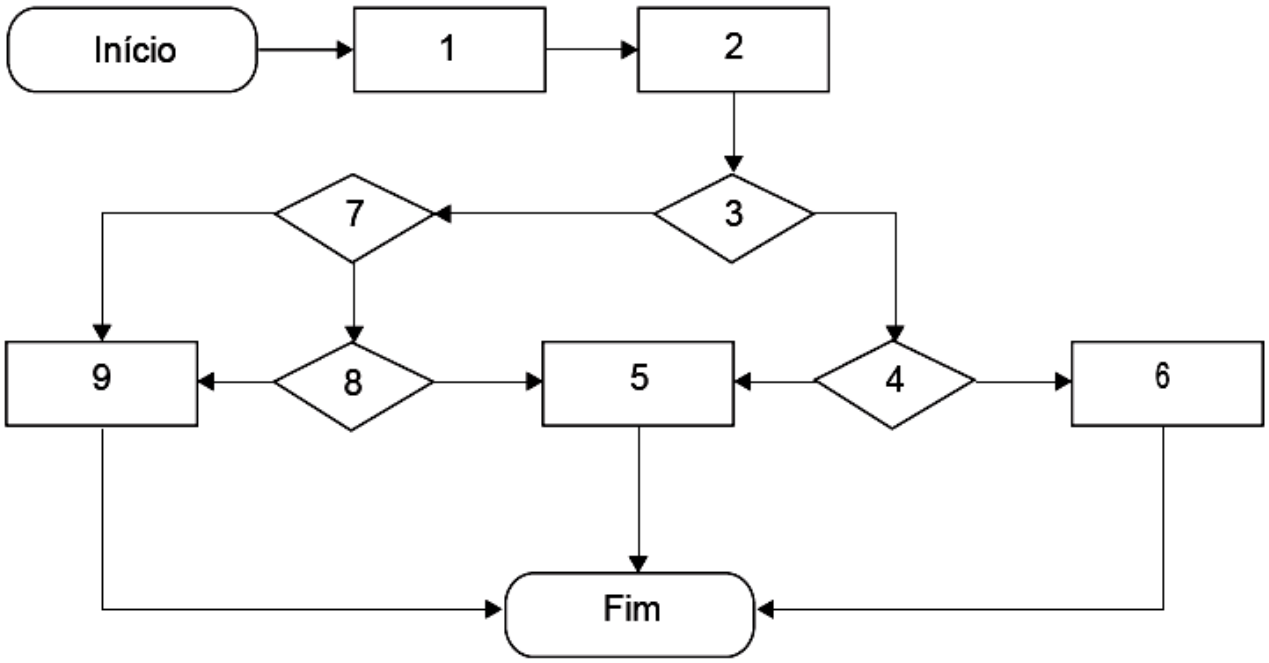
MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 8

Questão 1

Questão 1.[1]

Analise o fluxograma a seguir.



Em relação à execução de testes de caixa branca sobre este fluxograma, avalie as afirmativas a seguir.

I. Os caminhos 1-2-3-4-5, 1-2-3-4-6, 1-2-3-7-9, 1-2-3-7-8-9 formam um conjunto de caminhos de execução independentes.

II. O cálculo de complexidade ciclomática fornece a quantidade de caminhos independentes a testar.

III. Testes sobre caminhos independentes podem ser executados em programas procedurais, mas não podem ser executados em programas orientados a objetos.

É correto o que se afirma em

A. I, apenas.

B. III, apenas.

C. I e II, apenas.

D. II e III, apenas.

E. I, II e III.

[1]Questão 31 – Enade 2014.

1. Introdução teórica

Testes de caixa-preta e de caixa-branca. Complexidade ciclomática.

Para testarmos um programa ou um módulo de um programa, podemos utilizar, segundo Pressman (2011), os testes a seguir.

- **Teste de caixa-preta**. Nesse tipo de teste, são testadas apenas as interfaces externas ou apenas a funcionalidade desejada e especificada, sem haver conhecimento direto de como o programa funciona.
- **Teste de caixa-branca.** Nesse tipo de teste, são testados o programa e a sua estrutura interna, ou seja, o seu código fonte.

Ao fazermos um teste de caixa-branca, queremos ter a certeza de que estamos exercitando todas as linhas de código de um programa ao menos uma vez. Podemos visualizar a execução de um programa na forma de um fluxograma (figura 1) ou na forma de um grafo (figura 2). A representação em forma de grafo costuma ser mais adequada para a visualização de programas cujo fluxo de execução é complexo (PRESSMAN, 2011).

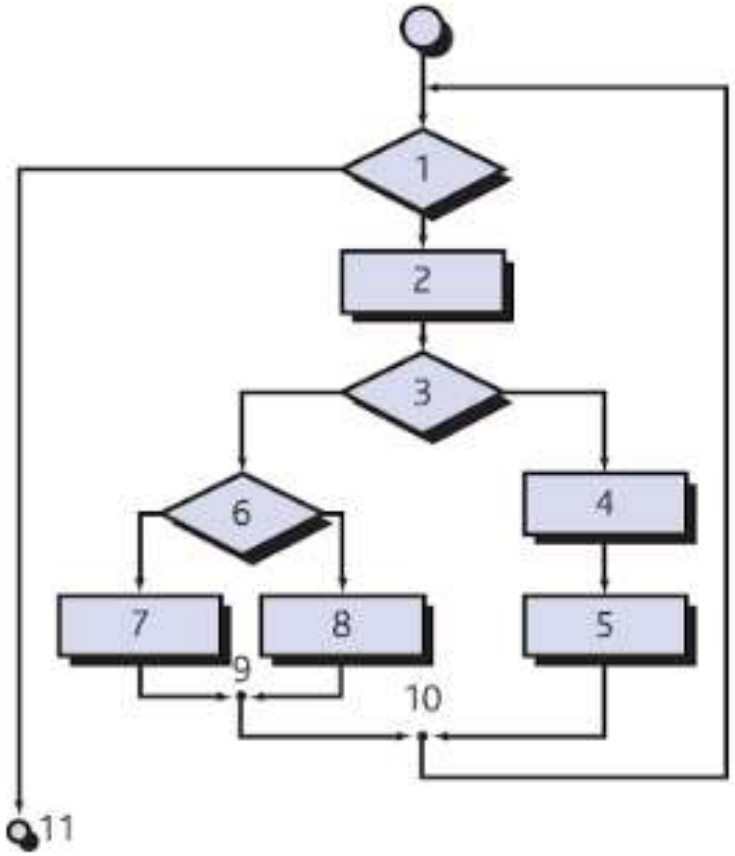


Figura 1. Exemplo de fluxograma da estrutura de um programa.
Fonte. PRESSMAN (2011).

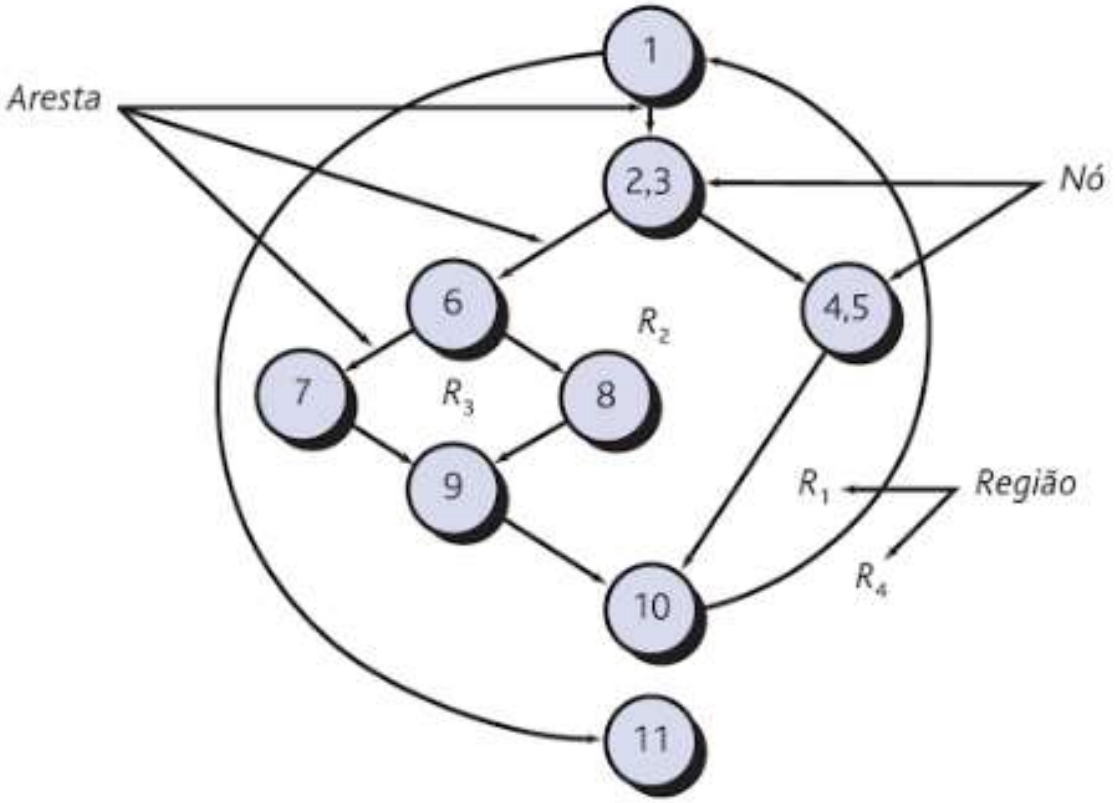


Figura 2. Exemplo de fluxograma da estrutura de um programa.
Fonte. PRESSMAN (2011).

Em um programa, a decisão corresponde à definição entre duas possibilidades diferentes do fluxo de execução: uma para o caso verdadeiro e outra para o caso falso. Um exemplo de estrutura de decisão é o comando "if", presente em várias linguagens de programação.

Vale notar que cada estrutura de decisão inserida a um programa aumenta o número de possíveis fluxos de execução.

Ao observarmos a figura 2, podemos verificar diversos possíveis fluxos de execução. Por exemplo, o caminho 1-11 é um possível fluxo de execução. Outra possibilidade é o caminho 1-2-3-4-5-10-1-11. Esses dois caminhos são independentes, ou seja, o segundo caminho introduz novos nós que não estão presentes no primeiro (PRESSMAN, 2011).

Outros dois caminhos independentes possíveis são: 1-2-3-6-8-9-10-1-11 e 1-2-3-6-7-9-10-1-11. Esses quatro caminhos formam um conjunto base para o grafo da figura 2. Já o caminho 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 não é independente, pois não introduz novos nós e é apenas uma combinação dos caminhos anteriores (PRESSMAN, 2011).

O número total de caminhos possíveis no grafo da figura 2 é chamado de complexidade ciclomática, indicada por V(G). Ela pode ser calculada a partir do número E de

arestas de um grafo de fluxo e do número N de nós desse grafo, conforme apresentado a seguir.

$$V(G)=E-N+2$$

2. Análise das afirmativas

I - Afirmativa correta.
JUSTIFICATIVA. Os caminhos são independentes, ou seja, cada caminho introduz uma nova aresta.

II - Afirmativa correta.
JUSTIFICATIVA. A complexidade ciclomática fornece uma medida da complexidade de um programa e enumera os diversos fluxos de execução de um programa.

III - Afirmativa incorreta.
JUSTIFICATIVA. A complexidade ciclomática pode ser aplicada tanto a programas procedurais quanto a programas orientados a objetos.

Alternativa correta: C.

3. Indicação bibliográfica

- PRESSMAN, R. S. *Engenharia de software: uma abordagem profissional*. 7. ed. Porto Alegre: AMGH, 2011.

Questão 2

Questão 2.[2]

O gerenciamento de projetos de sistemas é essencial para a engenharia de sofware. Um mau gerenciamento acarreta atraso na entrega do programa, custo maior do que o programado e falha no atendimento de requisitos. O quadro a seguir representa de forma hipotética algumas tarefas de um projeto de desenvolvimento de software. Esse quadro mostra as atividades, sua duração e as respectivas interdependências.

Tarefa	Duração (dias)	Dependências
T1	5	-
T2	8	T1
T3	2	T2
T4	4	-
T5	2	T3, T4
T6	4	-
T7	3	T5

Com base na análise do quadro, o tempo mínimo necessário para terminar o projeto é de

A. 13 dias.
B. 15 dias.
C. 16 dias.
D. 20 dias.
E. 28 dias.

1. Introdução teórica

Técnica de Revisão e Avaliação de Programa (PERT). Método do Caminho Crítico (CPM).

Segundo o guia Project Management Body of Knowledge (PMBOK), o "gerenciamento de tempo do projeto inclui os processos necessários para gerenciar o término pontual do projeto" (PMI, 2008). Essencialmente, a equipe envolvida no gerenciamento do projeto deve ser capaz de definir e sequenciar os recursos e a duração das atividades necessárias para a finalização do projeto nos prazos e com os custos estimados previamente.

Diversas técnicas foram desenvolvidas para auxiliar o gerenciamento do projeto, como aquelas que usam gráficos de Gantt.

[2]Questão 34 – Enade 2014.

Durante o final da década de 1950 e o início da década de 1960, especialmente nos Estados Unidos, o surgimento de grandes obras de engenharia e de projetos de proporção inédita, como o programa espacial e o programa de desenvolvimento de mísseis *Polaris*, entre outros, passaram a requerer o emprego de novas técnicas que permitissem o gerenciamento de projetos que não apenas fossem complexos, mas que também apresentassem elevadas incertezas em relação à execução de várias de suas atividades (KERZNER, 2013).

Uma das maiores dificuldades na utilização de técnicas como o gráfico de Gantt é a dependência entre atividades (KERZNER, 2013). Dessa forma, foram desenvolvidas novas técnicas baseadas em redes de atividades ou eventos, que permitem incorporar as interdependências das atividades, as incertezas nos tempos e as análises de diferentes cenários.

Duas técnicas destacam-se na categoria de redes: Técnica de Revisão e Avaliação de Programa (PERT) e Método do Caminho Crítico (CPM).

A PERT foi desenvolvida entre os anos de 1958 e 1959 pelo escritório de projetos da marinha americana, com o auxílio da consultoria Booz, Allen e Hamilton, e foi inicialmente utilizada para o Sistema de Armas Polaris (KERZNER, 2013).

Em uma típica rede PERT, bolhas representam eventos e flechas representam atividades, com suas respectivas durações. As flechas indicam a interdependência dos eventos de um projeto. Ao construirmos a rede, podemos identificar que existe um caminho mais longo do que os demais, chamado de caminho crítico. Esse caminho determina a duração do projeto.

PERT e CPM são técnicas parecidas, mas não são idênticas. Ambas utilizam diagramas de redes, mas não da mesma forma. No caso da técnica CPM, consideram-se tempos determinísticos e no caso da técnica PERT consideram-se tempos probabilísticos.

Em um gráfico CPM, a ênfase é dada às atividades e aos eventos, enquanto nos gráficos PERT ocorre o oposto. Técnicas do tipo CPM são normalmente utilizadas em projetos bem definidos, o que ocorre, por exemplo, em muitos projetos de engenharia civil, na área de construções. Já em projetos de pesquisa e desenvolvimento, nos quais há muitas incertezas e dificuldades na estimativa de prazos, normalmente é utilizada a técnica PERT (KERZNER, 2013).

2. Resolução da questão

Na figura 1, há um gráfico de rede do projeto, com 6 marcos e 7 atividades.

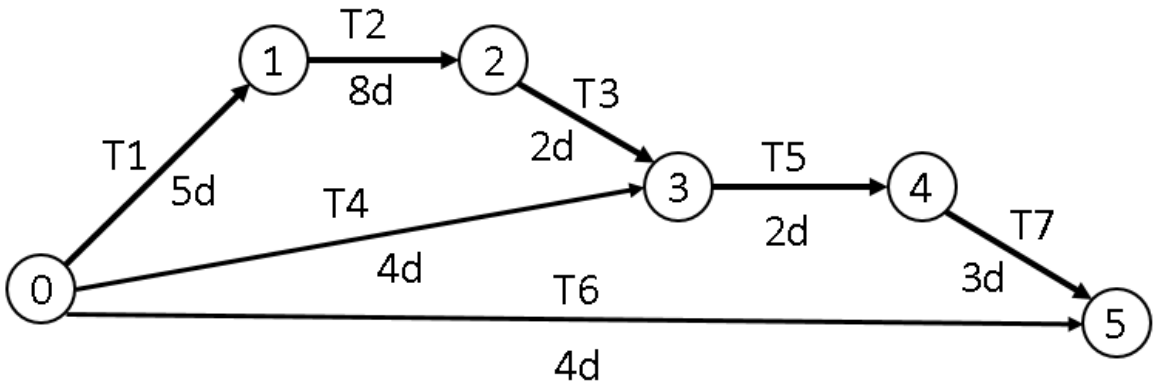


Figura 1. Rede PERT para problema.

O caminho crítico é composto pelas atividades T1, T2, T3, T5 e T7, com duração total de 5+8+2+2+3=20 dias.

Alternativa correta: D.

3. Indicações bibliográficas

- KERZNER, H. R. *Project management: a systems approach to planning, scheduling, and controlling*. 10. ed. New Jersey: John Wiley & Sons, 2013.
- PMI - Project Management Institute. *The guide to Project Management Body of Knowledge* – PMBOK. 4. ed. Project Management Institute, 2008.

Questão 3

Questão 3.[3]

O Rational Unified Process (RUP) é subdividido em fases, que indicam a ênfase que é dada ao projeto em um dado instante, e em fluxos de trabalho, que definem os grupos de atividades a serem realizadas ao longo das fases.

Considerando que no desenvolvimento de software utilizam-se técnicas de análise e projeto orientado a objetos e que as atividades sejam distribuídas de acordo com o RUP, o momento em que devem ser elaborados os diagramas de classes e de interação entre os objetos do sistema é

A. na fase de concepção, no fluxo de modelagem de negócios.
B. na fase de elaboração, no fluxo de análise e design.
C. na fase de concepção, no fluxo de análise e design.
D. na fase de elaboração, no fluxo de requisitos.
E. na fase de concepção, no fluxo de requisitos.

1. Introdução teórica

Rational Unified Process (RUP)

O Rational Unified Process (RUP) é o processo de desenvolvimento de software ou de estrutura de projeto que encoraja o desenvolvimento iterativo e incremental (FOWLER, 2005). Isso significa que um projeto complexo é dividido em pequenos "miniprojetos" (SCOTT, 2003), cujas atividades são executadas e entregues parcialmente ao final de cada iteração.

Uma nova iteração leva a uma mudança incremental e adiciona mais características ao produto. Essa abordagem é bastante diferente da abordagem tradicional em "cascata", na qual as diversas fases de um projeto são realizadas de forma linear e progressiva.

Um exemplo de abordagem em cascata é o modelo clássico "Análise/Design/Codificação/Teste" (PRESSMAN, 2011). Um dos grandes problemas dessa abordagem é que, frequentemente, novas e importantes informações sobre o projeto são descobertas em fases posteriores à fase de análise, como, por exemplo, na fase de codificação ou na fase de teste. Em uma abordagem do tipo cascata, não é possível refazer etapas anteriores e qualquer nova informação leva à abertura de um defeito.

[3]Questão 24 – Enade 2014.

Para evitar esse tipo de problema, o RUP divide o desenvolvimento em diversas iterações e permite que elas se comportem como miniprojetos (SCOTT, 2003).

Uma fase é o tempo decorrido entre dois marcos principais, os quais correspondem aos pontos em que os gerentes de projetos e demais envolvidos no gerenciamento tomam decisões importantes sobre o andamento do projeto. Cada fase tem um foco específico, mas todas as fases são compostas de uma série de disciplinas.

As quatro fases do RUP, de acordo com Fowler (2005) e Scott (2003), são as que seguem.

- **Concepção**: estabelecimento da viabilidade do sistema proposto.
- **Elaboração**: identificação dos casos principais de uso, com ênfase na identificação dos requisitos.
- **Construção**: desenvolvimento de funcionalidades suficientes para o lançamento do projeto.
- **Transição**: finalização das atividades do projeto, incluindo a distribuição e o treinamento do usuário.

Em cada fase do projeto, as tarefas são divididas como indicado abaixo.

- **Modelagem de negócios**: entendimento da estrutura da organização para a qual o sistema será entregue (KRUCHTEN, 2004).
- **Requisitos**: construção dos casos de uso que capturam as necessidades funcionais do sistema (SCOTT, 2003).
- **Análise e design**: transformação dos requisitos em um sistema candidato e implantação de uma arquitetura robusta para o sistema, em concordância com o ambiente de execução (SHUJA e KREBS, 2007).
- **Implementação**: construção do modelo a ser implantado (SCOTT,2003).
- **Teste**: construção do modelo de teste, com os diversos casos de teste identificados (SCOTT, 2003).
- **Implantação**: implementação de aspectos ligados à instalação do software.
- **Ambiente**: identificação e adaptação das ferramentas, do ambiente e dos processos (CRAIG, 2005).
- **Gerenciamento de projeto**: planejamento do projeto propriamente dito.
- **Gerenciamento de mudança e configuração:** configuração do software, solicitação de mudanças e gerenciamento do impacto dessas mudanças.

2. Análise das alternativas

A, C e E - Alternativas incorretas.
JUSTIFICATIVA. A fase de concepção corresponde à avaliação inicial do projeto. Por ser uma etapa preliminar, não se trata de um bom momento para a elaboração dos diagramas de classes.

B - Alternativa correta.
JUSTIFICATIVA. Normalmente, na fase de elaboração e na fase de fluxo de análise e design, são feitos os diagramas de classe e de interação entre objetos, uma vez que, nesses momentos, é investido grande esforço na organização da arquitetura do sistema. Esses não são os únicos momentos em que isso é feito: o momento que exige mais esforços é o da criação de uma arquitetura.

D - Alternativa incorreta.
JUSTIFICATIVA. No fluxo de requisitos, estamos mais interessados em criar os casos de uso do sistema do que em trabalhar na arquitetura do sistema.

3. Indicações bibliográficas

- CRAIG, L. *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. Upper Saddle River: Prentice Hall, 2005.
- FOWLER, M. *UML Essencial: um breve guia para a linguagem-padrão de modelagem de objetos*. Porto Alegre: Bookman, 2005.
- KRUCHTEN, P. *The rational unified process: an introduction*. Boston: Addison-Wesley Professional, 2004.
- PRESSMAN, R. S. *Engenharia de software: uma abordagem profissional*. Porto Alegre: AMGH, 2011.
- SCOTT, K. *O processo unificado explicado.* Porto Alegre: Bookman, 2003.
- SHUJA, A. K.; KREBS, J. *IBM rational unified process reference and certification guide.* Upper Saddle River: Pearson Education, 2007.

Questões 4, 5 e 6

Questão 4.[4]

Leia o texto a seguir.

A verificação e a validação de uma interface de usuário ocorrem em três pontos distintos: análise, projeto e teste. Considerando um cenário de uma aplicação web, tal verificação pode ser realizada através de testes de interface, testes de usabilidade e testes de compatibilidade.

PRESSMAN, R. *Engenharia de software: uma abordagem profissional*. 7. ed. McGraw Hill, 2011 (com adaptações).

Nesse contexto, avalie as afirmativas.

I. O teste de interface experimenta mecanismos de interação e valida aspectos estéticos da interface do usuário, apontando erros específicos de interface e erros na maneira como a interface implementa as semânticas de navegação, funcionalidade ou exibição de conteúdo.

II. O teste de usabilidade avalia o grau com o qual os usuários podem interagir efetivamente com a aplicação e o grau em que a aplicação dirige as ações do usuário.

III. O primeiro passo no teste de compatibilidade é definir uma série de configurações típicas encontradas do lado cliente e suas respectivas variantes, identificando características como plataforma, sistema operacional e navegador.

É correto o que se afirma em

A. I, apenas.

B. III, apenas.

C. I e II, apenas.

D. II e III, apenas.

E. I, II e III.

Questão 5.[5]

Leia o texto a seguir.

Às vezes, garantia de qualidade significa simplesmente a definição de procedimentos, processos e padrões que visam reforçar que a qualidade de software seja atingida. Em outros casos, a garantia de qualidade também inclui todo o gerenciamento de configuração, atividades de verificação e validação aplicados após o produto ter sido entregue por uma equipe de desenvolvimento.

SOMMERVILLE, I. *Engenharia de software*. 9. ed. São Paulo: Pearson, 2011. p.455.

Qualquer alteração inserida no processo de desenvolvimento de software aumenta a amplitude de erros e tende a descaracterizar o projeto inicialmente planejado, afetando a qualidade do produto ou serviço. O controle e a garantia da qualidade de software visam satisfazer as necessidades dos *stakeholders*; assegurar que os requisitos dos usuários sejam

[4]Questão 28 – Enade 2014.
[5]Questão 33 – Enade 2014.

atendidos pelas especificações dos produtos ou serviços; definir os processos para que o desenvolvimento do sistema atenda esses requisitos; gerenciar versões, mudanças, releases e a construção do sistema. A fim de assegurar a garantia da qualidade do software é indispensável definir um processo para controlar a documentação elaborada durante o processo de desenvolvimento.

Considerando essa situação, avalie as afirmativas.

I. As técnicas de verificação e validação de software estabelecem a confiança de que o sistema que esteja sendo desenvolvido é adequado ao seu propósito.

II. A gerência de configuração provê pontos de controle sobre os artefatos produzidos e modificados por diferentes recursos.

III. O gerenciamento de mudanças deve documentar a detecção de bugs e problemas introduzidos após a construção de uma nova versão do software.

IV. Faz parte das atribuições da equipe de garantia de qualidade examinar se os testes do sistema proporcionam cobertura dos requisitos e manter registros adequados do processo de teste.

V. A revisão de software tem por objetivo armazenar os documentos num repositório central.

É correto apenas o que se afirma em

A. I, II e III.

B. I, II e IV.

C. I, III e V.

D. II, IV e V.

E. III, IV e V.

Questão 6.[6]

A gerência de configuração de software, também chamada de gerência de mudança e configuração é o processo que mantém atualizadas as informações dos elementos de configuração.

A esse respeito, avalie as afirmativas.

I. A auditoria de software é um processo que consiste em verificar se a linha de base foi atendida ou se os requisitos do sistema foram atendidos.

II. A linha de base é o marco de referência, a partir do qual serão feitos os controles de mudança, e pode ser a versão 1.0 de um software.

[6]Questão 30 – Enade 2014.

III. Um item de configuração é um elemento unitário, que compõe aquele software e deve ser gerenciado.

É correto o que se afirma em

A. I, apenas.

B. II, apenas.

C. I e III, apenas.

D. II e III, apenas.

E. I, II e III.

1. Introdução teórica

1.1. Verificação, validação e testes

A qualidade de software é um tema muito importante para as empresas desenvolvedoras de programas e inclui as fases de verificação, validação e testes.

A verificação busca a garantia de que o software implementará uma função específica de forma correta (PRESSMAN e MAXIM, 2016).

A validação visa à certificação de que o software criado atenderá aos requisitos do cliente (PRESSMAN e MAXIM, 2016).

Nos escopos da verificação e da validação, os testes correspondem ao último elemento a partir do qual a qualidade poderá ser estimada e os erros poderão ser descobertos (PRESSMAN e MAXIM, 2016).

1.2. Teste de interface

A interface de uma aplicação web pode ser bastante complexa e envolver diversas formas de interação. Os testes de interface intencionam validar tanto os aspectos estéticos, importantes para a imagem de uma empresa, quanto a forma como as funcionalidades são implementadas pela interface.

Há uma série de objetivos importantes nos testes de interface (PRESSMAN e MAXIM, 2016), como os que seguem.

- Testar as características da interface para garantir que elas sejam apresentadas ao usuário sem erros, especialmente erros ligados aos conteúdos estéticos e visuais.
- Testar os mecanismos individuais da interface de forma análoga aos testes-unidade.

- Orientar os testes de cada um dos mecanismos da interface para uma categoria de usuário.
- Verificar o quão "fácil" é a navegação do usuário pela interface nos testes de usabilidade.
- Testar, em diversos ambientes, incluindo diferentes navegadores, sistemas operacionais e configurações, a compatibilidade entre as funcionalidades.

1.3. Teste de usabilidade

Segundo Bartié (2002), os testes de usabilidade têm como alvo "a facilidade de navegação entre as telas da aplicação, a clareza de textos e mensagens (...) e o acesso simplificado de mecanismos de apoio ao usuário".

Em outras palavras, o objetivo do teste de usabilidade é verificar o quão fácil ou intuitivo é um software.

1.4. Teste de compatibilidade

Devido à grande quantidade de plataformas a partir das quais um sistema web pode ser acessado e devido aos tipos de navegadores que rodam em diferentes plataformas, é importante testar o sistema e sua compatibilidade em relação às diversas plataformas de execução.

Mesmo com a existência de diversos padrões que deveriam garantir o mesmo tipo de experiência em diferentes plataformas, frequentemente bugs e desvios do padrão podem levar a um problema de execução em determinado ambiente. O problema torna-se ainda mais complexo quando levamos em consideração a variedade de telas de diferentes tamanhos e qualidades que podem ser utilizadas para se acessar um sistema.

O teste de compatibilidade busca verificar quais configurações funcionam de forma correta e visa a identificar como o sistema se comporta em plataformas diferentes das oficialmente suportadas.

1.5. Gerenciamento da configuração

Segundo Sommerville (2011),

> *o gerenciamento da configuração está relacionado às políticas, aos processos e às ferramentas para gerenciamento de mudanças dos sistemas de software.*

Para compreendermos a questão do gerenciamento da configuração, basta pensarmos nas complexas dependências que um programa pode ter: versões de bibliotecas, versão do sistema operacional e eventual dependência da disponibilidade de outros programas auxiliares.

Ao longo do seu desenvolvimento, um software passa por várias versões e revisões, com correções de defeitos e melhorias incorporadas ao longo do tempo. Essas mudanças podem afetar aspectos da configuração do programa, bem como alterar ou incorporar novas dependências. O gerenciamento da configuração deve lidar com todas essas questões.

Pode-se dizer que quatro atividades são fundamentais para o gerenciamento da configuração (SOMMERVILLE, 2011), conforme descrito a seguir.

- **Gerenciamento de mudanças.** O gerenciamento de mudanças refere-se à manutenção do acompanhamento das solicitações dos clientes e desenvolvedores por mudanças no software e à decisão sobre se e quando das mudanças devem ser implementadas. Ao longo do tempo de vida do software, começando ainda na sua concepção, o cliente pode solicitar funcionalidades, melhorias e correções. O gerenciamento de mudanças envolve os processos de controle e monitoramento.
- **Gerenciamento de versões**. As diversas alterações no código fonte de um programa devem ser controladas, especialmente em projetos que envolvem grande número de desenvolvedores.
- **Construção do sistema**. A construção do sistema normalmente envolve a sua compilação, dependendo da linguagem utilizada, bem como a sua ligação com bibliotecas.
- **Gerenciamento de releases**. As versões do programa enviadas ao cliente costumam ser chamadas de releases. É importante acompanhar as versões entregues ao cliente e suas dependências.

1.6. Revisões

Presmann e Maxim (2016) definem o processo de revisão de software como um filtro para seu processo de desenvolvimento. Os autores também definem que a revisão deve ser

capaz de apontar melhorias necessárias em um produto e de identificar as partes de um produto em que implementos não são necessários ou desejáveis.

1.7. Auditoria e revisões

Existem algumas diferenças entre os conceitos de auditoria e de revisão. Enquanto as revisões técnicas são feitas de engenheiros de software para engenheiros de software com o objetivo de descobrir erros, a auditoria é feita pela equipe de Software Quality Assurance (SQA) com o objetivo de verificar se as diretrizes de qualidade estão sendo seguidas pela equipe de engenharia de software (PRESMANN e MAXIM, 2016).

1.8. Linha de base

Segundo Sommerville (2011), uma

> *linha de base é uma coleção de versões de componentes que compõem um sistema.*

O padrão IEEE 610.12-1990 (IEEE, 1990) define a linha de base como

> *uma especificação ou produto que for formalmente revisado e acordado, que depois serve de base para mais desenvolvimentos, e que pode ser modificado apenas seguindo-se um procedimento formal de controle de mudanças.*

1.9. Item de configuração

Sommerville (2011) define um item de configuração como

> *qualquer coisa associada a um projeto de software que tenha sido colocado sobre o controle de configuração.*

O objetivo da equipe responsável é gerenciar cuidadosamente a evolução da configuração, a fim de controlar as mudanças que afetem esse item.

2. Análise das afirmativas

Questão 4.

I - Afirmativa correta.
JUSTIFICATIVA. O teste de interface busca identificar erros tanto no sentido estético, como uma página apresentada de forma errada, quanto no sentido de navegação, a fim de identificar erros nas funcionalidades implementadas no sistema.

II - Afirmativa correta.
JUSTIFICATIVA. Wiergers e Beatty (2013) definem usabilidade como "a facilidade que o usuário tem de aprender, memorizar e utilizar o sistema". No caso de um sistema web, é importante que a navegação seja intuitiva e que o usuário consiga facilmente encontrar as informações relevantes no sistema.

III - Afirmativa correta.
JUSTIFICATIVA. Devido às diversas possibilidades de configurações, é importante se concentrar nas mais comuns e identificar suas principais variações. Dessa forma, a empresa que desenvolve o sistema pode garantir um funcionamento adequado para a maior quantidade possível de usuários (ou apenas para os usuários relevantes).

Alternativa correta: E.

Questão 5.

I - Afirmativa correta.
JUSTIFICATIVA. Tão importante quanto atender aos requisitos do cliente é haver a presença de um software que apresente baixa quantidade de falhas. As técnicas de verificação e validação trabalham nesses dois aspectos.

II - Afirmativa correta.
JUSTIFICATIVA. A gerência da configuração envolve diversas atividades, como a construção do sistema e o gerenciamento de mudanças.

III - Afirmativa incorreta.
JUSTIFICATIVA. O gerenciamento de mudanças não é feito apenas após a detecção de bugs: trata-se de um processo contínuo a ser realizado durante toda a vida do software.

IV - Afirmativa correta.
JUSTIFICATIVA. A equipe de garantia da qualidade não trabalha apenas fazendo testes. A parte da equipe que cuida dessa área deve manter registros dos resultados e verificar se todos os requisitos estão sendo cobertos pelos testes.

V - Afirmativa incorreta.
JUSTIFICATIVA. A revisão não se limita a armazenar documentos, mas também visa a identificar problemas e melhorias em um artefato.

Alternativa correta: B.

Questão 6.

I - Afirmativa incorreta.
JUSTIFICATIVA. Segundo Presmann e Maxim (2016), a auditoria da configuração de software complementa a revisão técnica e é uma avaliação do objeto de configuração. Adicionalmente, as revisões técnicas estão centradas "na exatidão técnica do objeto de configuração modificado" (PRESMANN e MAXIM, 2016). A auditoria também deve verificar se os processos de desenvolvimento de software foram seguidos e se foram utilizados os padrões de engenharia de software. Dessa forma, seu escopo é muito maior do que apenas verificar se a linha de base e os requisitos estão sendo atendidos.

II - Afirmativa correta.
JUSTIFICATIVA. O uso da linha de base permite que mudanças sejam incorporadas de forma organizada a um projeto de software (PRESMANN e MAXIM, 2016).

III - Afirmativa correta.
JUSTIFICATIVA. Todo item de configuração deve estar sob o controle de configuração e deve ter um nome único (SOMMERVILLE, 2011).

Alternativa correta: D.

3. Indicações bibliográficas

- BARTIÉ, A. *Garantia da qualidade de software.* Rio de Janeiro: Elsevier, 2002.
- IEEE. Padrão 610.12-1990. IEEE *Standard glossary of software engineering terminology*. New York, 1990.
- *PRESSMAN, R. S.; MAXIM, B. R. Engenharia* de software: uma abordagem profissional. 8. ed. Porto Alegre: Bookman, 2016.
- SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo: 2011.
- WIEGERS, K.; BEATTY, J. *Software Requirements*. 3. ed. Redmond: Microsoft Press, 2013.

Questão 7

Questão 7.[7]

Os termos inovação e empreendedorismo dizem respeito à prática da gestão e à mudança criativa. A inovação preocupa-se, sobretudo, com o desenvolvimento de produtos, já o empreendedorismo dedica-se mais à questão da criação de pequenas empresas.

Nesse contexto, avalie as afirmativas a seguir.

I. Inovação e sustentabilidade são processos antagônicos, já que a inovação contribui de forma importante para a degradação do meio ambiente devido à sua associação com o aumento do desenvolvimento econômico e do consumo.

II. A exploração de um problema social transformado em uma oportunidade de negócio visando à geração de lucro ao empreendedor é um dos principais objetivos do empreendedorismo social.

III. Nem todo resultado de um processo criativo está associado a uma inovação.

É correto o que se afirma em

A. I, apenas.

B. II, apenas.

C. I e III, apenas

D. II e III, apenas

E. I, II e III.

1. Introdução teórica

Inovação e empreendedorismo

A inovação é a implementação de um produto melhorado ou novo que visa ao resultado econômico.

A capacidade de inovar está entre os fatores mais importantes no desempenho de um negócio: a empresa inova para crescer e conquistar espaço no mercado competitivo.

As inovações podem ser de caráter tecnológico, caráter gerencial ou caráter social. A inovação tecnológica ocorre em produtos e processos produtivos, a inovação gerencial acontece nos processos ou na estrutura gerencial e a inovação social é aquela que altera padrões de relacionamento entre pessoas e empresas.

[7]Questão 23 – Enade 2014.

Empreendedorismo é a disposição para que sejam identificados problemas e oportunidades e investidos recursos na criação de um negócio, de um projeto ou de um movimento capaz de alavancar mudanças e gerar impacto positivo.

Nesse universo, destaca-se o empreendedorismo social, em que o empreendedor cria negócios nos quais a responsabilidade social é o norte. São negócios lucrativos que resolvem problemas sociais por meio da venda de produtos ou serviços.

2. Análise das afirmativas

I – Afirmativa incorreta

JUSTIFICATIVA. Uma inovação tem a intenção de gerar resultado econômico para uma empresa. Não está implícito que a inovação contribua diretamente para a degradação do meio ambiente. Pelo contrário, muitas das inovações procuram alterar produtos e processos produtivos com a intenção de reduzir impactos ao meio ambiente e de melhorar a imagem da empresa.

II – Afirmativa correta

JUSTIFICATIVA. O empreendedorismo social é aquele em que o empreendedor cria negócios nos quais a responsabilidade social é o norte. São negócios lucrativos que resolvem problemas sociais por meio da venda de produtos ou serviços.

III – Afirmativa correta

JUSTIFICATIVA. O processo criativo pode ou não pode estar associado a uma inovação. A criação de um novo produto, por exemplo, é uma invenção. A inovação é a transformação da invenção.

Alternativa correta: D.

3. Indicações bibliográficas

- BES, F. T. de; KOTLER, P. *A bíblia da inovação.* São Paulo: Leya, 2011.
- DORNELAS, J. *Empreendedorismo – transformando ideias em negócios.* Rio de Janeiro: Empreende/LTC, 2014.

- HIRSCH, R. D.; PETERS, M. P.; SHEPHERD, D. E. *Empreendedorismo.* Porto Alegre: AMGH, 2014.
- SAKAR, S. *Empreendedorismo e inovação*. 3. ed. São Paulo: Escolar, 2014.
- SCHREIBER, D. *Inovação e desenvolvimento organizacional.* Novo Hamburgo: FEEVALE, 2012.
- TIDD, J.; BESSANT, J. *Gestão da inovação.* Porto Alegre: Bookman, 2015.

Questão 8

Questão 8.[8]

As classes costumam possuir relacionamentos entre si, chamados de associações, que permitem que elas compartilhem informações entre si e colaborem para a execução dos processos executados pelo sistema.

Com base nesse contexto, construa um diagrama de classes para representar as associações que seguem.

- Uma revista científica possui título, ISSN e periodicidade.
- Essa revista publica diversas edições com os seguintes atributos: número da edição, volume da edição e data da edição. Importante destacar que cada instância da classe edição relaciona-se única e exclusivamente a uma instância da classe revisão científica, não podendo relacionar-se com nenhuma outra.
- Um artigo possui o título e nome do autor. Um artigo é um conteúdo exclusivo de uma edição. E uma edição obrigatoriamente tem que possuir no mínimo 10 e no máximo 15 artigos.

1. Introdução teórica

Orientação a objetos: classes

Na orientação a objetos, utiliza-se o conceito de classes como uma forma de agrupar elementos que apresentam similaridades. Por exemplo, podemos imaginar a classe de todas as escolas de Ensino Médio do Brasil: ela agrupa enorme quantidade de diferentes escolas, de diferentes cidades e de diferentes tamanhos, com diversos alunos. Todas essas escolas têm uma característica em comum: são escolas do Ensino Médio. Contudo, isso não significa que todas as escolas são iguais: provavelmente uma escola bem é diferente da outra.

Essa ideia de agrupamento de elementos que pertencem a uma mesma categoria foi a que inspirou o desenvolvimento do conceito de classe na orientação a objetos. As classes correspondem às categorias de elementos, enquanto os objetos são instâncias de uma classe, de uma forma similar à ideia de que uma escola de Ensino Médio específica poderia ser considerada uma instância da classe "escolas do Ensino Médio".

A UML, acrônimo para a expressão Unified Modeling Language, possui um diagrama específico para trabalhar com classes, chamado de Diagrama de Classes. Os diagramas de

[8]Questão Discursiva 3 – Enade 2014.

classes "são utilizados para capturar as relações estáticas do software" (PILONE e PITMAN, 2006). Ainda segundo PILONE e PITMAN (2006), "uma classe representa um grupo de coisas que têm estado e comportamentos comuns".

A maioria dos programas orientados apresenta mais de uma classe, com diversos tipos de relações entre si. Uma dessas relações é chamada de associação. Podemos dizer que a associação é uma relação do tipo "...tem um...". Por exemplo, suponha a interface gráfica de um sistema operacional que pode ter uma classe para as janelas e outra classe para o ponteiro do mouse. Em dado instante, uma instância da classe janela pode conter um objeto do tipo ponteiro do mouse. Observe que se o usuário mover o ponteiro para outra janela, outro objeto vai "receber" o objeto. Um tipo mais forte de associação é chamado de agregação, que implica uma propriedade e, possivelmente, uma relação entre as linhas de vidas dos objetos envolvidos. Ainda mais forte do que a agregação é a composição, que indica uma relação do tipo "todo-parte". Por exemplo, no caso de uma interface gráfica, poderíamos dizer que uma janela possui uma barra de títulos e isso é representado por uma relação de composição (PILONE e PITMAN, 2006).

É possível especificar, no diagrama de classes, quantas instâncias de uma classe em particular estão envolvidas em uma associação, agregação ou composição, o que é chamado de multiplicidade (PILONE e PITMAN, 2006). O valor é indicado ao lado da classe de que é o alvo e, caso seja omitido, assume-se o valor 1. Exemplos de associações, agregações e composições são mostrados na figura 1.

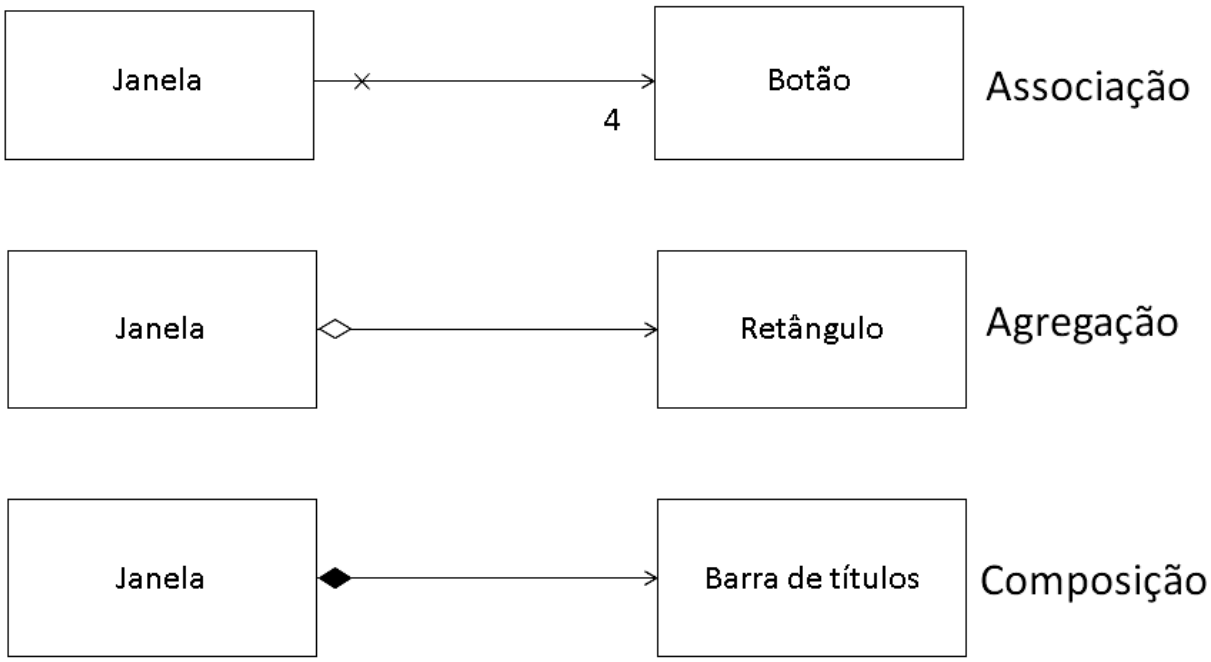


Figura 1. Exemplo de associação, agregação e composição.
Fonte. Adaptado de PILONE e PITMAN (2006).

2. Resposta padrão do INEP

A relação entre as classes Revista_Científica e Edição é muito forte, uma vez que, para existir a edição de uma revista, a revista também deve existir. Além disso, os artigos também fazem parte de uma edição, de forma que ambas as relações vão ser do tipo composição. Além disso, podemos representar os atributos de uma classe no diagrama de classes, dentro do mesmo retângulo, com a visibilidade marcada no lado esquerdo do nome do atributo (por exemplo, o símbolo "-" significa um atributo privado, ou seja, que está disponível apenas para membros da mesma classe).

A classe Revista_Científica possui como atributos: titulo_revista, issn_revista e periodicidade. A classe Edicao possui como atributos numero_edicao, volume_edicao e data_edicao. Finalmente, a classe Artigo possui como atributos titulo_artigo e nome_autor. Observe que uma edição deve possuir de 10 a 15 artigos, o que significa essa deve ser a multiplicidade da composição entre a classe Edicao e Artigo. Finalmente, a multiplicidade da composição entre Revista_Cientifica e Edicao deve ser 0.. ou 1..*, sendo que o último caso implicaria que uma revista deveria ter ao menos uma edição. O padrão de resposta do INEP é mostrado na figura 2.*

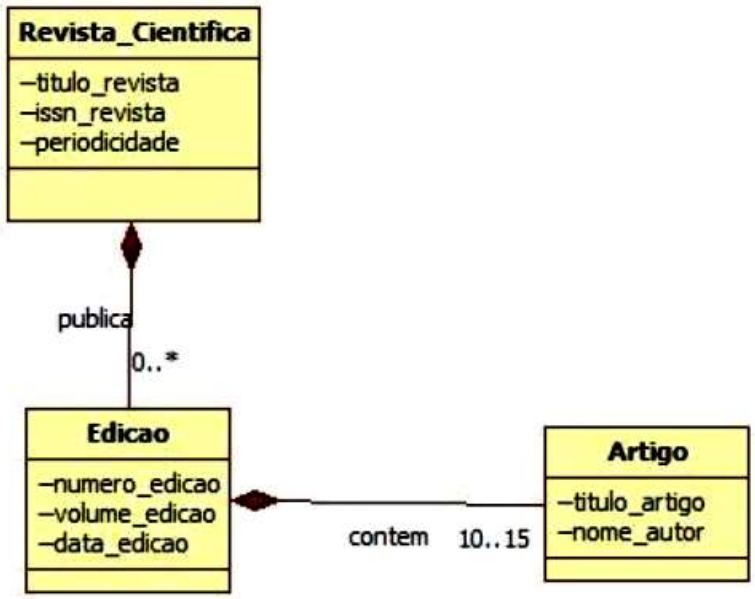


Figura 2. Diagrama de classes para o exercício.
Disponível em <http://download.inep.gov.br/educacao_superior/enade/padrao_resposta/2014/padrao_resposta_tecnologia_analise_desenv_sistemas.pdf>. Acesso em 18 set. 2017.

3. Indicação bibliográfica

- PILONE, D.; PITMAN, N. *UML 2: Rápido e Prático.* Rio de Janeiro: Alta Books, 2006.

Questão 9

Questão 9.[9]

Uma estrutura de dados do tipo pilha pode ser usada em um algoritmo que permite imprimir uma palavra de forma invertida. Exemplo: FELICIDADE deve ser impresso como EDADICILEF.

Considere as variáveis declaradas abaixo.

```
pilha[1..50]: caractere;
i, topo: inteiro;
palavra: string;
```

Em pseudocódigo, faça o que se pede nos itens a seguir.

A. Desenvolva a rotina push que inclui um elemento na pilha.
B. Desenvolva a rotina pop que retira um elemento da pilha.
C. Desenvolva a rotina que leia a palavra e, usando a pilha, a imprima de forma invertida.

1. Introdução teórica

Estruturas de dados: pilhas

A estrutura de dados denominada pilha tem comportamento muito similar ao comportamento de uma pilha de objetos na vida real. Por exemplo, suponha uma pilha de livros sobre uma mesa: idealmente, podemos adicionar livros ao topo da pilha ou retirar um livro do topo da pilha. Os demais livros que se encontram abaixo do livro que está no topo não estão (diretamente) acessíveis. Essa disciplina nos processos de inserção e de remoção garante que a informação sobre a ordem de inserção seja mantida. Dessa forma, o primeiro objeto a ser inserido é o último a ser removido e o último objeto a ser inserido é o primeiro a ser removido, o que costuma ser chamado, em inglês, de *LIFO (Last In, First Out)*.

[9]Questão Discursiva 4 – Enade 2014.

2. Padrão de resposta INEP

Parte A.

A primeira parte da questão corresponde à inserção dos elementos no topo da pilha. Do enunciado, é possível perceber a sugestão do uso de um vetor estático com 50 elementos, ou seja, a pilha vai armazenar no máximo 50 elementos. Uma variável deve conter o índice do elemento que é o topo da pilha, ou seja, o último elemento a ser inserido. Essa variável deve ser incrementada cada vez que inserimos um novo elemento na pilha. É importante controlarmos o tamanho máximo, uma vez que o vetor tem apenas 50 posições. Caso o valor da variável "topo" torne-se maior do que 50, o programa deve imprimir uma mensagem de erro. A variável c contém o caractere a ser depositado no topo da pilha. Isso é ilustrado no pseudocódigo abaixo.

```
função push(c:caractere)
início
  se topo >= 50
    escreva "Erro: Pilha Cheia"
  senão
    topo <- topo + 1
    pilha[topo] <- c
  fimse
fimfunção
```

Parte B.

A segunda função essencial para a implementação de uma pilha é a função pop, que retira o último elemento acrescentado ao topo da pilha. Essa função deve conter uma proteção para o caso de a pilha encontrar-se vazia, pois, nesse caso, não é possível remover nenhum elemento. Finalmente, é importante que a variável "topo" seja decrementada de uma unidade para cada remoção, uma vez que ela deve apontar para o elemento que se encontra imediatamente abaixo do elemento retirado, o qual vai ser o novo topo da pilha. O pseudocódigo a seguir ilustra essa função.

```
função pop(): caractere
início
  se topo <= 0
    escreva "Erro: Pilha Vazia"
  senão
    devolva(pilha[topo])
    topo <- topo -1
  fimse
fimfunção
```

Parte C.

Devido ao próprio funcionamento de uma pilha, em que o último elemento a ser acrescentado é o primeiro a ser retirado, basta acrescentarmos os caracteres na pilha um a um utilizando a função push e, depois, retirá-los utilizando a função pop. Como a pilha é uma estrutura de dados do tipo LIFO, depois de depositarmos todos os caracteres de uma palavra em uma pilha, eles vão ser retirados na ordem reversa a que foram colocados, como ilustrado na figura 1.

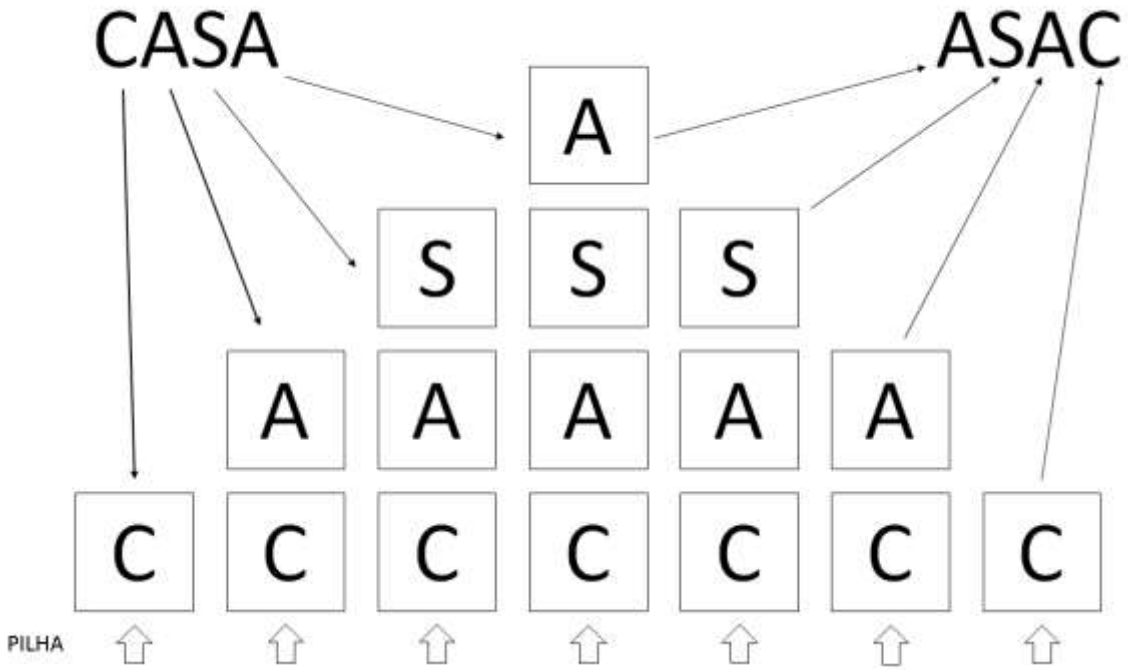


Figura 1. Uma pilha com a palavra casa.

O pseudocódigo do algoritmo é ilustrado logo a seguir.

```
função inverte()
início
      topo <- 0
      leia palavra
      para i <- 1 até tamanho(palavra) passo 1 faça
            push(palavra[i])
      fimpara
      para i <- 1 até tamanho(palavra) passo 1 faça
            imprima (pop())
      fimpara
fimfunção
```

Observe que o programa inicialmente lê toda a palavra, posteriormente insere essa palavra na pilha caractere a caractere (utilizando o comando push) e finalmente retira os caracteres na ordem inversa utilizando o comando pop.

Disponível em <http://download.inep.gov.br/educacao_superior/enade/padrao_resposta/2014/padrao_resposta_tecnologia_analise_desenv_sistemas.pdf>. Acesso em 18 set. 2017.

3. Indicações bibliográficas

- CELES, W.; CERQUEIRA, R.; RANGEL, J. R. *Introdução à estrutura de dados.* Rio de Janeiro: Campus, 2004.
- FEOFILLOF, P. *Algoritmos em linguagem C*. Elsevier Brasil, 2009.

Questão 10

Questão 10.[10]

Matrizes multidimensionais são vetores capazes de armazenarem mais de uma posição de cada elemento que será indicado por dois ou mais índices. Um exemplo de matrizes multidimensionais são as matrizes matemáticas, que representam valores tabulados em linhas e colunas.

```
01 algoritmo “matriz”
02 var
03 i, j : inteiro;
04 m1 : vetor [1..3, 1..3] de inteiro;
05 m2 : vetor [1..3, 1..3] de inteiro;
06 inicio
07 para i de 1 ate 3 faça
08   para j de 1 ate 3 faça
09         m1[i,j] := i + 1;
10         m2[i,j] := j + 1;
11   fimpara;
12 fimpara;
13 para i de 1 ate 3 faça
14   para j de 1 ate 3 faça
15         se (m1[i,j] = m2[i,j]) então
16               m1[i,j] := 0;
17         senão
18               m2[i,j] := 1;
19         fimse;
20   fimpara;
21 fimpara;
22 fimalgoritmo
```

Considerando o algoritmo acima e com base no teste de mesa, faça o que se pede nos itens a seguir.

A. Apresente os dados dos vetores m1 e m2 ao término da execução da linha 12.

B. Apresente os dados dos vetores m1 e m2 ao término da execução da linha 21.

[10]Questão Discursiva 5 – Enade 2014.

1. Introdução teórica

Vetores e matrizes em linguagens de programação

Diversas linguagens de programação apresentam estruturas de dados lineares, frequentemente chamadas de vetores. Na linguagem C, por exemplo, os vetores correspondem a um conjunto de posições de memória vizinhas, todas de mesmo tamanho e de dado tipo.

Contudo, em virtude de alguns domínios do conhecimento, especialmente na Matemática, pode-se querer trabalhar com estruturas de dados com mais dimensões, como as matrizes. Por exemplo, na linguagem C, podemos declarar uma matriz de números inteiros de 9 elementos (3 por 3) da seguinte forma:

```
int M[3][3];
```

Devemos observar que, na linguagem C, os vetores e matrizes começam do índice zero, de forma que o primeiro elemento de uma matriz seria o elemento M[0][0]. A figura 1 ilustra como os elementos da matriz estão dispostos logicamente. Do ponto de vista físico, as matrizes são armazenadas em uma região contínua de memória, similar aos vetores.

Figura 1. Visualização de uma matriz de inteiros 3 por 3, em C.

2. Padrão de resposta INEP

Parte A.

A matriz m1 armazena o valor da linha (dada pelo incide i) + 1, tendo a seguinte forma:

2 2 2

3 3 3

4 4 4

A matriz m2 armazena o valor da coluna (dada pelo incide j) + 1, tendo a seguinte forma:

2 3 4

2 3 4

2 3 4

Parte B.

Observe que a linha 15 modifica todos os elementos da matriz m1 que têm o mesmo valor que o elemento de mesma posição da matriz m2. Ao observarmos a matriz m1 obtida no item anterior, percebemos que eles correspondem aos elementos da diagonal. Esses elementos são igualados a zero, obtendo-se:

0 2 2

3 0 3

4 4 0

Por outro lado, a matriz m2 é modificada apenas no caso contrário, ou seja, quando os elementos são diferentes, o que corresponde a todos os elementos exceto os da diagonal. Nesses casos, escreve-se 1, obtendo-se:

2 1 1

1 3 1

1 1 4

Disponível em <http://download.inep.gov.br/educacao_superior/enade/padrao_resposta/2014/padrao_resposta_tecnologia_analise_desenv_sistemas.pdf>. Acesso em 18 set. 2017.

3. Indicações bibliográficas

- CELES, W.; CERQUEIRA, R.; RANGEL, J. R. *Introdução à estrutura de dados.* Rio de Janeiro: Campus, 2004.
- FEOFILLOF, P. *Algoritmos em linguagem C*. Elsevier Brasil, 2009.

ÍNDICE REMISSIVO