PROCESSO SELETIVO – 07/2026

Área de Conhecimento: Engenharia de Software

PROVA ESCRITA – PADRÃO DE RESPOSTA

QUESTÃO 1: Modelagem de Sistemas

Segundo descrito pelo Capítulo 4 de Valente (2020):

(a) Propósito dos Modelos e Simplificação: Os modelos de software atuam como artefatos intermediários de abstração situados entre a informalidade dos requisitos e a complexidade executável do código-fonte, tendo como propósito principal auxiliar no entendimento, na análise e na comunicação estrutural de um sistema. Como todo modelo constitui uma simplificação deliberada ou aproximação da realidade - conforme a máxima estatística de que todos os modelos são aproximações imperfeitas -, a sua utilidade prática não reside em representar a totalidade exata do sistema, mas sim em servir como uma abstração direcionada e eficiente para o estudo de propriedades específicas do software em desenvolvimento.

(b) Diagramas Estáticos vs. Dinâmicos: A distinção conceitual fundamental entre os diagramas estruturais e os diagramas comportamentais na UML reside na perspectiva sob a qual o sistema é modelado: a arquitetura fixa em contraste com o fluxo em execução. Os diagramas estruturais retratam a perspectiva estática do software, focando nos elementos que não mudam com o passar do tempo nem durante o processamento de dados, atuando como o esqueleto do sistema. O Diagrama de Classes exemplifica essa categoria ao definir o vocabulário do modelo com suas classes, atributos, métodos e relacionamentos, sendo amplamente aplicado na modelagem conceitual e no projeto orientado a objetos. Outro exemplo representativo é o Diagrama de Componentes, utilizado na arquitetura de software para ilustrar a organização e as dependências entre os módulos físicos, bibliotecas e executáveis do sistema. Por outro lado, os diagramas comportamentais capturam a perspectiva dinâmica do software, descrevendo como os objetos colaboram, reagem a eventos e alteram seus estados ao longo do tempo. O Diagrama de Sequência ilustra essa categoria ao detalhar a troca de mensagens entre atores e objetos em ordem cronológica estrita, sendo essencial para especificar a execução interna de um caso de uso. Da mesma forma, o Diagrama de Máquina de Estados foca nos ciclos de vida de objetos individuais com lógica de controle complexa, mapeando as transições entre diferentes estados disparadas por eventos específicos. Ambas as categorias são estritamente complementares no ciclo de desenvolvimento, pois enquanto os diagramas estáticos estabelecem as entidades e fronteiras arquiteturais disponíveis, os diagramas dinâmicos mostram como essas entidades colaboram e reagem para concretizar as funcionalidades e os requisitos do software.

Pág. 01 de 04 - Documento assinado digitalmente. Para conferência, acesse o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo UDESC 00044036/2026 e o código 9AA57R0A.

QUESTÃO 2: Processos de Software

Proposta de Solução baseada nos conceitos de Processos de Software apresentados por Valente (2020), Capítulo 2, do livro Engenharia de Software Moderna

(a) O modelo Waterfall, ou em cascata, caracteriza-se por um ciclo de vida sequencial e linear onde as etapas de requisitos, projeto, implementação e testes ocorrem de forma estanque e sem sobreposição, oferecendo como principal vantagem a alta previsibilidade de custos e prazos quando os requisitos são fixos, mas acumulando o grave risco de rigidez a mudanças e descoberta tardia de falhas apenas ao final do projeto. Em contrapartida, os métodos ágeis fundamentam-se em ciclos iterativos e incrementais que entregam pequenas versões funcionais do software com frequência, tendo como vantagens a rápida adaptação a mudanças de requisitos e o retorno de valor precoce ao cliente, enquanto seus principais riscos residem na menor previsibilidade inicial de prazos globais e na dependência de alta colaboração e maturidade da equipe.

(b) A filosofia ágil, formalizada no Manifesto Ágil em 2001, fundamenta-se em quatro valores centrais que priorizam indivíduos e interações em detrimento de processos e ferramentas, software em funcionamento em vez de documentação exaustiva, colaboração contínua com o cliente superando a rigidez de contratos, e a capacidade de responder prontamente a mudanças em vez de seguir cegamente um plano estático. Para viabilizar esses princípios de forma prática, o framework Scrum estabelece um ciclo iterativo e incremental estruturado em ciclos de trabalho curtos chamados de sprints, operando por meio de papéis, artefatos e eventos bem definidos. O Product Owner gerencia e prioriza o Product Backlog garantindo a maximização do valor do produto, o Scrum Master atua como facilitador removendo impedimentos e garantindo a adesão ao framework, e o time de desenvolvedores executa a construção técnica do software. Essa dinâmica é pontuada por artefatos essenciais, como o Product Backlog, o Sprint Backlog e o Incremento utilizável ao final de cada ciclo, e articulada por eventos fundamentais, que incluem o Sprint Planning para o planejamento do ciclo, a Daily Scrum para sincronização diária da equipe, a Sprint Review para inspeção do produto entregue junto aos stakeholders, e a Sprint Retrospective dedicada à melhoria contínua dos processos do time.

Pág. 02 de 04 - Documento assinado digitalmente. Para conferência, acesse o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo UDESC 00044036/2026 e o código 9AA57R0A.

QUESTÃO 3: Arquitetura de Software

Com base em Valente (2020), Capítulo 7 seção 4, sobre Arquiteturas de Microsserviços, seguem respostas aos questionamentos:

(a) A primeira vantagem é permitir a evolução mais rápida e independente do sistema, pois cada time passa a ter seu próprio regime de liberação de novas releases, resolvendo o gargalo entre o desenvolvimento ágil e a entrada em produção burocrática típica dos monolitos. A segunda vantagem é a escalabilidade em granularidade mais fina: enquanto a escalabilidade horizontal de um monolito exige replicar instâncias idênticas do sistema inteiro, na arquitetura de microsserviços é possível replicar apenas os serviços diretamente responsáveis pelos problemas de performance - como ilustra o exemplo em que seis instâncias do serviço M1 são disponibilizadas em um novo servidor. A terceira vantagem é a heterogeneidade tecnológica: como os microsserviços são autônomos e independentes, podem ser implementados em linguagens, frameworks e bancos de dados distintos, a exemplo do serviço de cadastro de clientes em Java com banco relacional e do serviço de recomendação em Python com banco NoSQL. A quarta vantagem é a possibilidade de falhas parciais: ao contrário do monolito, em que a queda do banco de dados derruba todos os serviços, a indisponibilidade de um microsserviço (como o de recomendações) não impede que os clientes continuem pesquisando produtos e realizando compras - a área de recomendações apenas fica desabilitada, vazia ou ausente. O autor acrescenta ainda que essa arquitetura tornou-se possível graças às plataformas de computação em nuvem, que permitem alugar máquinas virtuais pagas por hora, facilitando a escalabilidade horizontal dos serviços.

(b) A arquitetura de microsserviços é mais complexa do que a monolítica, pois, por construção, os microsserviços são processos independentes e dão origem a sistemas distribuídos. Assim, sua utilização impõe o enfrentamento de todos os desafios inerentes a sistemas distribuídos: a complexidade, já que a comunicação entre módulos em máquinas diferentes deixa de ser por chamadas de método e passa a exigir protocolos de comunicação, como HTTP/REST, demandando dos desenvolvedores o domínio de tecnologias de comunicação em redes; a latência, pois a comunicação entre microsserviços envolve um atraso maior, sobretudo quando os serviços estão em máquinas distantes, gerando um custo de comunicação não desprezível; e as transações distribuídas, uma vez que a autonomia de dados de cada microsserviço dificulta garantir que operações sobre dois ou mais bancos de dados sejam atômicas - ou executam com sucesso em todos ou falham -, podendo exigir protocolos como o two-phase commit. Portanto, na perspectiva do autor, a arquitetura não deve ser adotada quando esses custos inerentes aos sistemas distribuídos — complexidade de comunicação, latência e dificuldade de garantir atomicidade transacional — tornarem-se obstáculos desfavoráveis ao sistema em questão. Adicionalmente, o autor destaca a recomendação de que nunca se deve iniciar um novo sistema diretamente com uma arquitetura de microsserviços (adotando a máxima de construir primeiro um monólito modular). Como em sistemas novos o domínio do negócio e os contornos dos módulos ainda são instáveis e desconhecidos, tentar dividi-los em microsserviços prematuramente gera acoplamento incorreto na rede, tornando a refatoração extremamente onerosa. Portanto, a arquitetura também deve ser evitada em sistemas em estágio inicial onde a equipe é pequena e o modelo de negócio ainda precisa ser validado.

Pág. 03 de 04 - Documento assinado digitalmente. Para conferência, acesse o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo UDESC 00044036/2026 e o código 9AA57R0A.

QUESTÃO 4: Engenharia de Requisitos

Com base em Valente (2020), no Capítulo 3 sobre Requisitos do livro Engenharia de Software Moderna:

Requisitos Funcionais: Definem **o que** o sistema deve fazer, ou seja, suas funcionalidades, regras de negócios e os serviços que serão oferecidos aos usuários para resolver seus problemas. Eles determinam as ações que o software precisa executar e, na maioria das vezes, são especificados em linguagem natural. No contexto de um **sistema bancário**, exemplos de requisitos funcionais incluem:

- Permitir a consulta de saldos e extratos de contas correntes.
- Realizar transferências bancárias entre contas (como TEDs e Pix).
- Efetuar o pagamento de boletos e faturas.

Requisitos Não-Funcionais: Definem **sob quais restrições** o sistema deve operar, estando associados à qualidade do serviço prestado, desempenho, segurança e usabilidade. Eles funcionam como limites impostos à arquitetura e à implementação do software. Diferentemente dos funcionais, os requisitos não-funcionais devem ser especificados de forma quantitativa utilizando métricas claras (como tempo de resposta, porcentagem de disponibilidade ou uso de recursos) para evitar ambiguidades. No contexto de um **sistema bancário**, exemplos de requisitos não-funcionais incluem:

- **Disponibilidade:** O sistema de home-banking deve operar com uma disponibilidade mínima de 99,9% do tempo.
- **Desempenho:** 99% de todas as transações de consulta e pagamento realizadas em qualquer janela de 5 minutos devem ter um tempo de resposta máximo de 1 segundo.
- **Segurança:** Todas as senhas e dados sensíveis de transações devem ser armazenados utilizando algoritmos de criptografia robustos e padrões de conformidade vigentes.

Guilherme Piegas Koslovski

Gilmário Barbosa dos Santos

Carla Diacui Medeiros Berkenbrock

Presidente da Banca Examinadora

Pág. 04 de 04 - Documento assinado digitalmente. Para conferência, acesse o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo UDESC 00044036/2026 e o código 9AA57R0A.

Assinaturas do documento

Código para verificação: **9AA57R0A**

Este documento foi assinado digitalmente pelos seguintes signatários nas datas indicadas:

CARLA DIACUI MEDEIROS BERKENBROCK (CPF: 025.XXX.559-XX) em 09/11/2026 às 14:05:15
Emitido por: "SGP-e", emitido em 30/03/2018 - 12:39:40 e válido até 30/03/2118 - 12:39:40.
(Assinatura do sistema)

GUILHERME PIEGAS KOSLOVSKI (CPF: 002.XXX.110-XX) em 09/11/2026 às 14:11:54
Emitido por: "SGP-e", emitido em 30/03/2018 - 12:35:42 e válido até 30/03/2118 - 12:35:42.
(Assinatura do sistema)

Para verificar a autenticidade desta cópia, acesse o link https://portal.sgpe.sea.sc.gov.br/portal-externo/conferencia-documento/VURFU0NfMTIwMjJfMDAwNDQwMzZfNDQwNDZfMjAyNl85QUE1N1IwQQ== ou o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo **UDESC 00044036/2026** e o código **9AA57R0A** ou aponte a câmera para o QR Code presente nesta página para realizar a conferência.