PROCESSO SELETIVO – 04/2026

Área de Conhecimento: Engenharia de Software

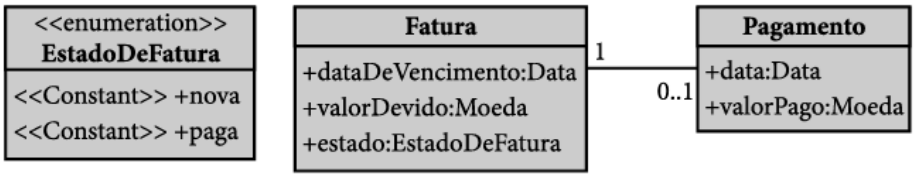
PROVA ESCRITA – PADRÃO DE RESPOSTA

QUESTÃO 1: Modelagem de Sistemas

Segundo Wazlawick R. (2014), não é adequado modelar a transição monotônica crescente com herança, porque nesse caso uma instância de FaturaNova deveria ser transformada em uma instância de FaturaPaga ou ela teria de ser destruída e criada novamente do zero e ter todas as suas referências reconstruídas.

Segundo o autor mencionado, é adequado modelar essa situação dividindo o conceito original em dois: um que representa a fatura e outro que representa seu pagamento com as propriedades que forem adicionadas quando o pagamento ocorrer. Esses dois conceitos estão ligados por uma associação de 1 para 0..1, conforme a Figura 1. Com este modelo é impossível que uma fatura tenha data de pagamento definida e valor pago indefinido, e isso é garantido sem adicionar qualquer restrição em seus atributos ou associações. O estado de uma fatura é um atributo derivado, o qual pode ser calculado da seguinte forma: se não há pagamento ligado à fatura, então a fatura é nova; caso contrário, ela é paga. Esta condição poderia ser expressa pelo modelo através de OCL (*Object Constraint Language*), verificando se um objeto está ligado a algum outro ou não.

Figura 1 – Forma adequada de modelar uma transição monotônica crescente.



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.*

1. Análise Comparativa:

- Modelo Preditivo (Waterfall): Baseia-se na premissa de que os requisitos são estáveis e previsíveis. Embora traga o benefício teórico de uma arquitetura rigidamente documentada no início (*Big Design Up Front*), apresenta um risco catastrófico para este cenário. Conforme dados históricos como o *CHAOS Report* (discutido no Cap. 2 do livro *Engenharia de Software Moderna*), processos sequenciais tendem a estourar prazos e orçamentos em ambientes voláteis. Como a API do governo muda semanalmente, o time ficaria preso tentando atualizar especificações em papel, resultando em um sistema obsoleto antes mesmo de ser programado. O software só ficaria pronto no final, impedindo validações antecipadas.
- Modelo Adaptativo (Ágil): Tem como característica principal a adoção de ciclos curtos e iterativos. O grande benefício é a resposta a mudanças mais do que seguir um plano (Manifesto Ágil). Ele assume o desenvolvimento em ambientes com informações imperfeitas e parciais. O risco seria perder o rigor técnico em prol da velocidade,

o que é mitigado ao adotar as práticas de engenharia corretas.

2. Justificativa de Escolha:
A abordagem recomendada é Adaptativa. Para mitigar a volatilidade semanal da API externa e blindar a segurança dentro dos 90 dias, propõe-se uma estratégia que extrai o melhor de todos os três métodos ágeis detalhados na literatura de Valente: o ritmo de gestão do Scrum, a otimização de fluxo contínuo do Kanban e o rigor técnico do Extreme Programming (XP).

3. Aplicação Prática:
Para garantir a qualidade de engenharia e a entrega no prazo, a rotina do time aplicará práticas, tais como:

- Sprints e Histórias de Usuário (Scrum): O prazo de 90 dias será dividido em ciclos fixos curtos (*Sprints* de 2 semanas). O escopo vasto será quebrado em *Histórias de Usuário* (especificações resumidas focadas no valor de negócio). A cada final de iteração, um incremento real de software funcional (S++) será gerado e validado. Isso garante que o "cerne" da integração com o SUS esteja pronto e testado muito antes do 90º dia.
- Quadro Visual e Limites de WIP (Kanban): Para lidar com as mudanças semanais na API do governo sem sobrecarregar a equipe, será adotado um Quadro Kanban com Limites de WIP (*Work in Progress*). Se a coluna de "Testes" ou "Homologação do SUS" atingir o limite máximo de tarefas em andamento, os desenvolvedores ficam proibidos de puxar novas funções. Eles devem ajudar a escoar o gargalo atual. Isso evita o desperdício com alternância de contexto e garante foco na conclusão (*"parar de começar e começar a terminar"*).
- Integração Contínua (XP): Com múltiplos desenvolvedores codificando sob extrema pressão de tempo, o risco de conflitos destrutivos no código é alto. A prática de Integração Contínua do XP obriga o time a subir suas modificações para o repositório principal diariamente. Um serviço automatizado fará o *build* contínuo, prevenindo o travamento do projeto na reta final.
- Testes Automatizados (XP): Para blindar a segurança dos prontuários (LGPD), o desenvolvimento será orientado por testes. Nenhuma história será considerada concluída sem passar por uma suíte de testes unitários e de integração automatizados. Se uma atualização da API do governo quebrar a segurança do sistema, os testes acusarão o erro em minutos na esteira de CI, permitindo correção imediata e protegendo a reputação da marca

QUESTÃO 3: Modelos Estruturais

Com base em Valente (2020), seção 7.2, no que se refere à arquitetura em camadas, trata-se de um padrão em que as classes são organizadas em módulos maiores, denominados camadas, dispostas de forma hierárquica, de modo que uma camada só deve utilizar serviços da camada imediatamente inferior. Uma de suas principais vantagens é a partição da complexidade do sistema em componentes menores, o que favorece a organização do desenvolvimento. Também, essa arquitetura disciplina as dependências entre as partes do sistema, contribuindo para o entendimento, a manutenção e a evolução do software. Além disso, essa organização facilita a substituição de camadas e também o reúso, uma vez que uma mesma camada pode atender a mais de uma camada superior.

Quanto às desvantagens ou limitações da arquitetura em camadas, o autor não elenca nenhuma desvantagem de forma explícita, mas, ao tratar de sistemas em duas camadas, ele explicita que, quando as camadas de interface e aplicação são unificadas no cliente, ocorre a desvantagem de concentrar todo o processamento nessa máquina, exigindo, portanto, maior poder computacional dos clientes. Esse é o ponto que deve ser indicado como limitação explicitamente mencionada.

Em relação à arquitetura MVC, a Visão é responsável pela apresentação da interface gráfica, as Controladoras tratam os eventos gerados pelos dispositivos de entrada e o Modelo armazena os dados da aplicação e a lógica relacionada ao domínio. Portanto, a principal contribuição do MVC consiste na separação entre o código da interface com o usuário e a lógica do domínio, o que torna essas partes mais fáceis de modificar. Entre as vantagens, está a especialização do trabalho de desenvolvimento, permitindo separar responsabilidades entre desenvolvedores voltados à interface e desenvolvedores voltados ao modelo, a possibilidade de utilização de um mesmo modelo por diferentes visões e a melhoria da testabilidade, já que os elementos do modelo, por não serem visuais, tendem a ser mais fáceis de testar.

Sobre às desvantagens ou limitações do MVC, o autor não apresenta uma desvantagem direta do MVC, entretanto, pode ser citado, como limitação prática, o fato de que nem sempre existe uma distinção clara entre Visão e Controladoras, sendo inclusive observado que, em muitos sistemas, essa separação não é rigorosa.

Por fim, quanto à diferença entre arquitetura em camadas/três camadas e arquitetura MVC, a arquitetura em três camadas organiza o sistema, de forma geral, em Interface com o Usuário, Lógica de Negócio e Banco de Dados, sendo normalmente uma arquitetura distribuída, na qual a interface executa no cliente, a lógica de negócio em um servidor de aplicação e o banco de dados em outra camada. Já o MVC, em sua formulação original, foi proposto para a implementação de interfaces gráficas, organizando-as em Visão, Controladoras e Modelo. Portanto, não se tratam de arquiteturas equivalentes, mas podem ser utilizadas de forma complementar, onde o MVC podia ser empregado especificamente na implementação da camada de interface de um sistema em três camadas. Assim, um sistema pode adotar três camadas como arquitetura geral e, simultaneamente, utilizar MVC na camada de interface. Por fim, a confusão entre os termos se intensificou com a popularização dos frameworks Web, que passaram a empregar a terminologia de MVC para organizar aplicações que, em certa medida, se assemelham a sistemas em três camadas.

QUESTÃO 4: Engenharia de Requisitos

Com base em Valente (2020), seção 3.5, e Sommerville (2013), seção 2.3.1, MVP e prototipação têm semelhanças, mas não são a mesma coisa. Os dois são usados para evitar desperdício de tempo e dinheiro no desenvolvimento de software. Em vez de construir logo um sistema completo, a ideia é criar uma versão inicial para aprender mais cedo sobre o produto.

A principal semelhança entre eles é que ambos ajudam a testar ideias antes de investir demais. Nos dois casos, não se desenvolve o sistema inteiro logo no começo. Além disso, tanto o MVP quanto o protótipo servem para obter feedback e fazer melhorias com base no que foi observado.

Por outro lado, a principal diferença está no objetivo de cada um. O MVP é criado para testar a viabilidade de um negócio ou produto no mercado. Ou seja, ele busca responder se os usuários realmente querem aquele sistema e se vale a pena continuar investindo nele. Já a prototipação é usada para entender melhor os requisitos, a interface e possíveis soluções técnicas do sistema.

Outra diferença importante é que o MVP é um produto, mesmo sendo simples e com poucas funções. Ele é colocado em uso por clientes reais, para medir resultados reais, como uso, retorno e interesse. Já o protótipo não precisa ser um produto final, nem ser usado por clientes reais. Muitas vezes, ele serve apenas para demonstrar ideias, validar requisitos ou testar a interface com usuários e stakeholders.

Também há diferença na forma de avaliação. No MVP, são usadas métricas de negócio, como aquisição de usuários, retenção, receita e recomendação. Na prototipação, a avaliação costuma focar em descobrir erros, omissões, melhorias de interface e ajustes nos requisitos.

Além disso, o protótipo pode até não ser executável, como no caso de maquetes em papel ou protótipos do tipo “Mágico de Oz”. Já o MVP precisa funcionar minimamente bem, porque será usado em uma situação real. Se ele tiver qualidade muito ruim, pode gerar uma avaliação errada do produto.

Por fim, pode-se dizer que o MVP está mais ligado à validação de mercado, enquanto a prototipação está mais ligada à validação de requisitos e do projeto do sistema.

Concluindo, MVP e prototipação são parecidos porque ambos permitem aprender antes de construir o sistema completo. Porém, o MVP testa se o produto faz sentido para o mercado, enquanto a prototipação testa se a solução está bem entendida e bem projetada.

Carla Diacui Medeiros Berkenbrock

Rebeca Schroeder Freitas

Fabiano Baldo

Presidente da Banca Examinadora

Assinaturas do documento

Código para verificação: **8T6DL1K5**

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

FABIANO BALDO (CPF: 028.XXX.209-XX) em 22/06/2026 às 15:23:28
Emitido por: "SGP-e", emitido em 30/03/2018 - 12:47:06 e válido até 30/03/2118 - 12:47:06.
(Assinatura do sistema)

Para verificar a autenticidade desta cópia, acesse o link https://portal.sgpe.sea.sc.gov.br/portal-externo/conferencia-documento/VURFU0NfMTIwMjJfMDAwMjMyNjNfMjMyNjhfMjAyNl84VDZETDFLNQ== ou o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo **UDESC 00023263/2026** e o código **8T6DL1K5** ou aponte a câmera para o QR Code presente nesta página para realizar a conferência.