Adendo - Material para prova escrita

Anielle Severo Lisbôa de Andrade

Disponível em: https://github.com/AniLisboa/resumos-concurso-anielle-andrade

3. Arquitetura de Software

Seção 2. FUNDAMENTOS & QUALIDADE (A Base Teórica)

- **Guias da Arquitetura: Atributos de Qualidade = Requisitos Não Funcionais (RNFs)** → (Link: Eng. de Requisitos).
- **Exemplos Chave (PSEM):**
  - **P**erformance (tempo de resposta).
  - **S**egurança (proteção).
  - **E**scalabilidade (crescimento).
  - **M**anutenibilidade (facilidade de mudança) → (Link: Manutenção/Evolução).
- **Modelo 4+1 Vistas (Kruchten):** "Descrever para diferentes stakeholders".
  - **Lógica:** Funcionalidade → Para Devs.
  - **Processo:** Comportamento/Concorrência → Para Integradores.
  - **Implantação:** Hardware/Rede → Para Infra/DevOps.
  - **Implementação:** Módulos/Código → Para Ger. de Desenvolvimento.
  - **+1 Cenários (Casos de Uso):** Validação da arquitetura → (Link: Eng. de Requisitos).

Seção 3. PADRÕES (As Ferramentas)

- **Diferença Crucial:**
  - **ESTILOS Arquiteturais:** MACRO, template, estrutura geral.
    - Ex: **Camadas**, Fluxo de Dados (Pipe-Filter).
  - **PADRÕES:** MICRO, solução p/ problema recorrente e específico.
- **Tipos de Padrões:**
  - **De Arquitetura:** Solução estrutural.
    - Ex: **Three-Tier (3 Camadas)**: Apresentação ↔ Lógica de Negócio ↔ Dados.
  - **De Projeto (GoF):** Solução de código (classes/objetos).
    - Ex: **Factory** (criação), **Singleton** (instância única).
- **Benefícios:** Vocabulário comum, reuso de soluções comprovadas.

Seção 4. TENDÊNCIAS (A Evolução)

- **4.1. Contexto ÁGIL:**
  - NÃO: **Big Design Up Front (BDUF)**.
  - SIM: **Design Evolutivo / Incremental**. Arquitetura é **contínua**.
  - Modelo: **Twin Peaks** (Reqs e Arq. co-evoluem).
  - Metáfora: Arquitetura como **"guia vivo"**.
- **4.2. MICROSERVIÇOS:**
  - Contraste: vs. **Monolito**.
  - **VANTAGENS (PROS):**
    - **E**scalabilidade (independente).
    - **I**ndependência Tecnológica.
    - **R**esiliência (falha isolada).
    - **M**anutenibilidade (código menor).
  - **DESVANTAGEM (CONS): COMPLEXIDADE!** (Gestão, orquestração, **testes distribuídos**, monitoramento.)
  - **Conceito-chave: Trade-offs**.

Seção 5. CONCLUSÃO (A Mensagem Final)

- **Síntese:** Arquitetura transcende o código, foca em decisões de alto nível para garantir qualidades.
- **Adaptação:** Evolui com tendências (Ágil, Microserviços).
- **Objetivo Final:** Sistema não só atende requisitos, mas é **robusto, flexível e sustentável** a longo prazo → **Sucesso de negócio**.

10. Padrões Arquiteturais e de Projetos

2. PADRÕES ARQUITETURAIS (O Esqueleto / Macro)

- **Conceito:** Alto nível, decisão crítica, difícil de mudar. Define as qualidades globais (escalabilidade, manutenibilidade).
- **Exemplos:**
    - **Cliente-Servidor:** Fundamental. Cliente (solicita) ↔ Servidor (provê). Centraliza dados.
    - **Em Camadas (Layered):** Organização lógica (Apresentação, Negócio, Dados). Separação de preocupações → Alta **Manutenibilidade**.
    - **Microserviços:** Moderno. Serviços pequenos, autônomos, desacoplados. (Contraste: vs. Monolito).

3. PADRÕES DE PROJETO (Os Tijolos / Micro / GoF)

- **Conceito:** Soluções de design OO. Fonte: **"Gang of Four" (GoF)**.
- **Classificação GoF (Ponto Avançado):**
    - **Finalidade:** O que faz? → **Criação, Estrutura, Comportamento**.
    - **Escopo:** Aplica-se a quê? → Classes ou Objetos.
- **Categorias & Exemplos Chave:**
    - **CRIAÇÃO (Como criar objetos?):**
        - **Singleton:** Garante **instância única**. (Ex: Conexão DB, Configs).
    - **ESTRUTURA (Como compor estruturas?):**
        - **Adapter:** "Traduz" interfaces incompatíveis (ponte).
        - **Facade:** Simplifica um subsistema complexo (fachada).
    - **COMPORTAMENTO (Como objetos se comunicam?):**
        - **Observer:** Notificação 1-para-muitos. (Ex: Inscrição em canal, UI).

4. COMPLEMENTARIDADE (A Grande Síntese - Seção Estratégica)

- **Relação:** Não são excludentes, atuam em níveis diferentes.
- **Analogia Central: Arquitetura = Esqueleto | Padrões de Projeto = Tijolos**.
- **Exemplo Prático:** Dentro de um **Microserviço** (Arq.), usamos **Facade** ou **Strategy** (Projeto).
- **Benefício Real:** Reuso de **CONHECIMENTO**, não só de código.
    - → Vocabulário comum.
    - → Comunicação eficiente da equipe.
    - → Evita "reinvenção da roda".

5. CONCLUSÃO (A Mensagem Final)

- **Síntese:** Padrões são um catálogo de soluções testadas, a espinha dorsal de um bom software.
- **Visão Elevada:** Não são regras, são uma **mentalidade de engenharia**.
- **Argumento Final:** Padrões são a prova do **amadurecimento da Eng. de Software**:
    - Deixou de ser "arte de programação".
    - Tornou-se "**processo de engenharia** robusto".

7. Linhas de Produto de Software

1. INTRODUÇÃO (A Motivação)

Processos-Chave:

- **Eng. de Domínio:** CRIA os ativos reutilizáveis.
- **Eng. de Aplicação:** USA os ativos para criar produtos.

Visão Estratégica: LPS é um **INVESTIMENTO** com retorno a longo prazo (↓custo, ↓time-to-market).

4. SÍNTESE (O Papel do Reuso e da Arquitetura)

- **Reuso:** LPS é a resposta para os problemas do reuso ad hoc. É o **reuso sistemático e planejado**.
- **Arquitetura:** É o **ativo MAIS CRÍTICO**, a **"cola"** da LPS.
    - **Dualidade Essencial:** Precisa ser **ESTÁVEL** (nas partes comuns) e **FLEXÍVEL** (nos pontos de variação).
    - **Risco:** Arquitetura ruim → nega todos os benefícios da LPS.

5. CONCLUSÃO (A Mensagem Final)

- **Recap:** LPS transcende o reuso ad hoc, com base em arquitetura e variabilidade.
- **Fatores de Sucesso (Não-técnicos):** Planejamento, **comprometimento gerencial**, **mudança cultural**.
- **Visão Final:** LPS não é só uma técnica, é uma **FILOSOFIA de produção**.
    - Oferece **vantagem competitiva sustentável**.
    - Transforma a capacidade de inovar de uma organização.

Referências

[1] BACHMANN, Felix; CLEMENTS, Paul C. Variability in Software Product Lines. Software Engineering Institute, Pittsburg. set. 2005. 61p. Relatório Técnico.
[2] EZRAN, Michael; MORISIO, Maurizio; TULLY, Colin. Practical Software Reuse. Berlin: SPRINGER, 2013.
[3] MOLINARI, L. Gerência de Configuração: técnicas e práticas no desenvolvimento do software. Florianopolis: Visual Books, 2007.
[4] PRESSMAN, Roger S.. Engenharia de Software. 6a ed., São Paulo, McGRAW-Hill, 2006.

6. Melhoria de Processo de Software (MPS)

1. INTRODUÇÃO (O "Porquê")

- **Definição:** Esforço **contínuo** para aprimorar processos (desenv/manutenção).
- **Relação Fundamental: Qualidade do Processo → Qualidade do Produto**.
- **Objetivo:** ↑ Qualidade, ↓ Custos, ↓ Tempo. → Competitividade Organizacional.
- **Insight Chave (Trade-offs):** Impossível otimizar tudo ao mesmo tempo (Ex: **Rapidez vs. Visibilidade**).
- **Natureza:** Atividade **cíclica**, de longo prazo.

2. MODELOS DE MATURIDADE (A Teoria Clássica)

- **Modelo Principal: CMMI** (Capability Maturity Model Integration).
    - **Conceito:** Roteiro evolutivo de melhores práticas.
    - **5 NÍVEIS DE MATURIDADE (Essencial saber):**
        - **1 - Inicial:** Caótico, reativo, baseado em "heroísmo".
        - **2 - Gerenciado:** Projetos são controlados a nível de projeto.
        - **3 - Definido:** Processo padronizado em toda a organização.
        - **4 - Gerenciado Quantitativamente:** Gerenciamento estatístico, com métricas.
        - **5 - Em Otimização:** Foco em melhoria contínua e inovação.

Ciclo de Melhoria (O "Como"):

- **Etapas:** 1. **Medição** → 2. **Análise** → 3. **Mudança**.

3. MÉTRICAS (A Prática Baseada em Dados)

- **Base:** Medição é o alicerce da melhoria. Sem dados, é "achismo".
- **Estratégia:** Deve ser guiada por **objetivos e perguntas** (ex: "estamos entregando com qualidade?").
- **2 Tipos de Métricas (Complementares):**
    - **DE PROCESSO:** Medem *COMO* o trabalho é feito (eficácia do fluxo, taxa de correção de defeitos).
    - **DE PRODUTO:** Medem *O QUE* foi entregue (complexidade do código, usabilidade, densidade de defeitos).
- **Síntese:** Juntas, fornecem uma **visão holística**.

4. TENDÊNCIAS ATUAIS (A Modernização)

- **4.1. Contexto ÁGIL:**
    - **Abordagem:** Não é formal/burocrática, mas **orgânica e contínua**.
    - **Mecanismo-Chave: Reunião de Retrospectiva** (Scrum) → ciclo de inspeção e adaptação do processo pela própria equipe.
- **4.2. Automação (O Catalisador / DevOps):**
    - **Papel:** Fornece **feedback rápido e sistemático**.
    - **Ferramentas-Chave:**
        - **Automação de Testes**.
        - **Integração Contínua (CI):** Integrar código automaticamente.
        - **Entrega Contínua (CD):** Manter o software sempre pronto para deploy.
    - **Resultado:** Transforma a MPS de atividade lenta → para componente **ágil e eficiente** do dia a dia.

5. CONCLUSÃO (A Grande Síntese)

- **Visão Integrada:** O sucesso da MPS não está em um único método.
    - **CMMI:** Fornece o roteiro estruturado.
    - **Ágil:** Promove a melhoria contínua e orgânica.
    - **Métricas:** São a base para todas as decisões.
    - **Automação:** É o catalisador que torna tudo eficiente.
- **Mensagem Final:** A jornada da MPS é a **integração de diferentes ferramentas e filosofias** para capacitar as equipes a entregar software de alta qualidade.

2. Reuso de Software

Seção 2: Fundamentos e Estratégias do Reuso
2.1 Reuso Ad-hoc vs, Sistemático

A prática de reuso de software se divide em duas abordagens principais, com resultados muito distintos: o reuso ad hoc e o sistemático.

O **Reuso Ad-Hoc**, também chamado de **oportunístico**, é a forma mais básica e informal. Acontece quando um desenvolvedor, de maneira não planejada, copia e adapta um trecho de código de um projeto antigo para resolver um problema atual. Embora possa parecer uma solução rápida, essa prática é insustentável a longo prazo, pois leva à duplicação de código, cria pesadelos de manutenção (uma correção precisa ser feita em vários lugares) e depende do conhecimento individual dos membros da equipe, não sendo um ativo da organização.

Em contraste, o **Reuso Sistemático**, ou **planejado**, é uma decisão estratégica da empresa. Ele trata o reuso como uma disciplina de engenharia, onde a organização investe na criação de ativos de software (componentes, arquiteturas, etc.) de alta qualidade, projetados desde o início para serem reutilizáveis. O objetivo é transformar o desenvolvimento em um processo mais previsível e eficiente, semelhante a uma linha de produção industrial.

Para que o reuso sistemático funcione, é necessária uma estrutura organizacional, descrita pelo **Paradigma Produtor-Consumidor**. Este modelo propõe a divisão de papéis dentro da equipe de desenvolvimento:

1. **O Produtor de Ativos:** Uma equipe especializada e dedicada a criar os componentes reutilizáveis. O foco deles é exclusivamente na qualidade, generalidade e documentação desses ativos.
2. **O Consumidor de Ativos:** A equipe de desenvolvimento de aplicações, que utiliza os ativos prontos, fornecidos pelos produtores, para montar os produtos finais de forma mais rápida e com maior qualidade.
3. Um conceito central é o **Mercado Interno**, onde os próprios desenvolvedores são os "clientes" dos ativos. Para que esse mercado funcione, não basta criar os ativos; é crucial mantê-los, documentá-los com manuais e guias, e promovê-los para incentivar a adoção dentro da empresa.

O paradigma também destaca que apenas criar os ativos não é suficiente. É preciso criar um "ecossistema de reuso" com atividades de suporte essenciais, como a **manutenção contínua** dos ativos, a criação de **documentação clara** (manuais e guias) e até mesmo um "marketing interno" para promover a adoção dos componentes reutilizáveis.

Em suma, a disciplina de reuso de software evolui de uma prática individual e caótica (ad-hoc) para uma estratégia organizacional robusta (sistemática). O Paradigma Produtor-Consumidor fornece o modelo de como estruturar as equipes e os processos para alcançar esse nível de maturidade, sendo a base fundamental para estratégias ainda mais avançadas, como as Linhas de Produto de Software.

1. O Alicerce (O que vamos construir?):

- **1. Engenharia de Requisitos:** É a planta baixa do prédio. Define o que o software deve fazer e quais as restrições. É o ponto de partida de tudo.

2. A Estrutura (Como vamos construir?):

- **3. Arquitetura de Software:** É o projeto estrutural. Define os pilares, as vigas, os andares. É a decisão de mais alto nível sobre como o sistema será organizado para atender aos requisitos (especialmente os não funcionais, como segurança e desempenho).
- **10. Padrões Arquiteturais e de Projetos:** São as técnicas e soluções de engenharia testadas e aprovadas que o arquiteto usa para desenhar a estrutura (ex: usar concreto protendido, estrutura metálica, etc.). Os padrões são o "como" em um nível mais detalhado.
- **2. Reuso de Software:** É a decisão de usar "peças pré-fabricadas" em vez de construir tudo do zero. Uma boa arquitetura facilita o reuso, e uma estratégia de reuso influencia as decisões de arquitetura.

3. A Garantia de Qualidade (A construção está correta e segura?):

- **4. Verificação, Validação e Teste de Software:** É a inspeção da obra. Garante que cada parte (tijolo, viga) está correta (verificação) e que o prédio final atende às necessidades do morador (validação). Isso acontece durante e após a construção.
- **5. Manutenção, Refatoração e Evolução de Software:** É a vida do prédio após a entrega. Inclui consertar vazamentos (manutenção corretiva), adaptar para novas normas (adaptativa) e melhorar a estrutura interna para evitar problemas futuros (refatoração, que é uma manutenção preventiva).

4. A Gestão e a Estratégia (Como organizar tudo isso?):

- **11. Gestão de Projetos de Software:** É o mestre de obras e o administrador. Ele gerencia o cronograma, o orçamento, os recursos e os riscos para garantir que o prédio seja entregue. Ele orquestra todas as outras atividades.
- **7. Linhas de Produto de Software (LPS):** É uma estratégia de negócio para construtoras que fazem condomínios. Em vez de projetar uma casa de cada vez, você cria um "projeto mestre" (Engenharia de Domínio) e depois gera várias casas parecidas, mas com pequenas variações (Engenharia de Aplicação). **É a forma mais avançada de Reuso de Software.**

- **6. Melhoria de Processo de Software:** É a construtora buscando um selo de qualidade (como ISO 9001) para seus métodos de construção. O objetivo é tornar o processo de construir prédios mais previsível, eficiente e com maior qualidade, aprendendo com os projetos passados.

5. A Fronteira do Conhecimento (Como podemos ser melhores e mais inteligentes?):

- **8. IA Aplicada na Engenharia de Software:** São as ferramentas avançadas que o mestre de obras e os engenheiros usam. É como usar drones para inspecionar a fachada, ou um software que prevê falhas estruturais com base em dados. A IA pode ser aplicada em *todos* os outros tópicos.
- **9. Engenharia de Software Experimental:** É o "laboratório de pesquisa" da construtora. É onde eles testam cientificamente se um novo tipo de cimento (uma nova ferramenta de teste, por exemplo) é realmente melhor que o antigo. Fornece as *evidências* de que as técnicas que usamos nos outros 10 tópicos realmente funcionam.

Quais são os Mais Importantes?

Se você tivesse que focar em um "núcleo duro" que sustenta todos os outros, seriam estes quatro:

1. **Engenharia de Requisitos (Nº 1):** Porque se você errar aqui, você constrói a coisa errada perfeitamente. Nenhum outro tópico conserta um requisito mal definido.
2. **Arquitetura de Software (Nº 3) / Padrões (Nº 10):** Porque as decisões de arquitetura são as mais difíceis e caras de mudar depois. Uma arquitetura ruim compromete a qualidade, a manutenção e a evolução do software para sempre.
3. **Verificação, Validação e Teste (Nº 4):** Porque um software que não funciona ou não é confiável não tem valor, não importa quão bem planejado ou arquitetado ele seja.
4. **Gestão de Projetos de Software (Nº 11):** Porque é o processo que une todas as partes. Sem uma boa gestão, a melhor equipe técnica pode falhar em entregar o projeto no prazo, no custo ou com o escopo correto.

Os outros tópicos são, em geral, especializações, estratégias ou aprofundamentos destes quatro pilares.

`Gestão de Projetos (11)` → gerencia o processo que começa com `Requisitos (1)` → que guia a `Arquitetura (3)` e os `Padrões (10)` → cuja qualidade é garantida por `V&V` e `Teste (4)` → e que continua a viver através da `Manutenção e Evolução (5)`.

Os outros tópicos (`Reuso`, `LPS`, `Melhoria`, `IA`, `Experimental`) são estratégias e ferramentas que potencializam e otimizam esse fluxo principal.