**Estimativa de Software**

**Análise por Pontos de Função (APF)**

A Análise por Pontos de Função (APF) é uma técnica utilizada para medir a funcionalidade de um software com base nos requisitos fornecidos pelo usuário. Essa métrica é independente da tecnologia utilizada e ajuda a estimar o esforço necessário para o desenvolvimento do sistema.

* **1º Passo: Decomposição Funcional**

Componente Funcional Básico (CFB): Uma unidade elementar dos Requisitos Funcionais do Usuário (RFU), representando o que precisa ser feito.

Exemplo:

O requisito "Manter Clientes" pode ser decomposto nos seguintes CFBs:

* "Incluir um novo cliente".
* "Reportar Compras do Cliente".
* "Alterar Detalhes do Cliente".

**Requisitos Funcionais (RF) e Não Funcionais (RNF):**

**RF:** Descrevem o que o sistema deve fazer (funções e serviços).

**RNF:** Descrevem como o sistema deve ser (desempenho, segurança, usabilidade).

**Fatores de ajuste:**

Ajustam a complexidade do desenvolvimento com base em índices multiplicadores, como integração com outros sistemas ou requisitos de portabilidade.

Nota: Não confunda **RF** com Requisitos de Usuário! Os RF podem incluir requisitos técnicos para implementar regras de negócio ou resolver questões de engenharia e arquitetura.

* **2º Passo: Classificação Funcional**

Os CFBs (equivalentes aos itens do backlog de produto) devem ser classificados em categorias que refletem seu objetivo. A APF utiliza as seguintes categorias:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Componente funcional básico** | | | | |
| Interação função da transação | | | Armazenamento função de dados | |
| Entrada externa | Saída externa | Consulta externa | Arquivo lógico interno | Arquivo de interface externa |

* **Entrada Externa (EE):**

Corresponde a componentes que capturam, transformam e armazenam dados vindos de fora do sistema.

Exemplo: Tela de entrada de dados, programa de carga em massa a partir de um arquivo, TL para carga de Datawarehouse.

* **Saída Externa (SE):**

Gera a exportação de dados para fora da aplicação, transformando os dados extraídos.

Exemplo: Geração de arquivo de carga para BI a partir de banco relacional de ERP.

* **Consulta Externa (CE):**

Realiza buscas e exibe informações sem transformar os dados.

Exemplo: Tela de consulta de cadastro, página web para pesquisa de dados, script SQL para consulta de banco de dados.

* **Arquivo Lógico Interno (ALI)**

Base de dados exclusiva do sistema em desenvolvimento. Não é acessada diretamente por outros sistemas.

Exemplo: Tabelas de um sistema de gestão interna.

* **Arquivo de Interface Externa (AIE):**

Base de dados usada na troca de informações entre o sistema em desenvolvimento e outros sistemas externos.

Exemplo: Arquivo XML usado para integração entre sistemas legados.

* **3º Passo: Identificação de Dados Referenciados (DER)**

A quantidade de dados elementares usados na execução das transações é chamada de DER (Dados Elementares Referenciados).

Exemplo:

Campos como "Nome do Cliente", "Endereço" ou "Código do Pedido" são DERs.

* **4º Passo: Classificação da Complexidade Funcional**

Cada CFB é avaliado de acordo com a quantidade de DERs e de arquivos referenciados.

A complexidade funcional é classificada como:

* Simples
* Média
* Complexa

**Tabela de Complexidade Funcional:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo de CFB** | **Simples** | **Médio** | **Complexo** |
| Entrada Externa | 1 a 4 DERs | 5 a 15 DERs | + 15 DERs |
| Saída Externa | 1 a 5 DERs | 6 a 19 DERs | + 19 DERs |
| Consulta | 1 a 4 DERs | 5 a 15 DERs | + 15 DERs |

* **5º Passo: Cálculo do Esforço de Projeto**

Com os pontos de função não ajustados calculados, aplica-se fatores de ajuste para considerar elementos que influenciam a produtividade, como:

* Ferramentas utilizadas (IDE, frameworks, etc.).
* Requisitos não funcionais (desempenho, segurança, portabilidade).
* Complexidade técnica (integrações, volume de dados).

Esses fatores resultam nos pontos de função ajustados, que permitem estimar o esforço necessário para o projeto.

**Estratégias de Teste**

A indústria de software segue um roteiro estabelecido para liberar novos produtos ou versões atualizadas de produtos existentes. Esse processo é essencial para garantir a qualidade e funcionalidade antes da disponibilização ao público em geral.

**Ciclo de Liberação de Software**

Empresas como Google e IBM utilizam as seguintes etapas no ciclo de desenvolvimento e liberação: Alpha → Beta → Release Candidate → RTM (Release to Manufacturing) → GA (General Availability)

* **Alpha:**

Primeira fase de testes, geralmente realizada internamente. O software ainda está em desenvolvimento, contendo funcionalidades iniciais que podem não estar completas ou estáveis.

|  |  |
| --- | --- |
| **Objetivo** | **Quem testa** |
| Identificar bugs iniciais, problemas graves e validar o funcionamento básico. | Desenvolvedores, equipes de QA e, ocasionalmente, um grupo seleto de usuários internos. |

* **Beta:**

Segunda fase de testes, aberta para um grupo maior de usuários externos, muitas vezes chamada de "teste de aceitação do usuário".

|  |  |
| --- | --- |
| **Objetivo** | **Quem testa** |
| Avaliar a usabilidade, funcionalidade em condições reais e identificar falhas não detectadas na fase Alpha. | Um grupo de usuários finais ou testadores externos que fornecem feedback à equipe de desenvolvimento. |

* **Release Candidate (RC):**

Versão do software que está próxima de ser finalizada, contendo todas as funcionalidades planejadas e corrigindo os principais problemas reportados nas fases anteriores.

|  |  |
| --- | --- |
| **Objetivo** | **Quem testa** |
| Validar se o software está pronto para lançamento, executando testes de regressão e verificações de estabilidade. | Internamente ou por um grupo restrito de usuários avançados. |

* **RTM (Release to Manufacturing):**

Versão pronta para distribuição, entregue para fabricantes ou distribuidores. Muitas vezes chamada de "versão Gold".

|  |  |
| --- | --- |
| **Objetivo** | **Quem acessa** |
| Preparar o software para produção em larga escala e distribuição. | Empresas parceiras, fabricantes de hardware ou provedores de serviços que incluem o software em seus produtos. |

* **GA (General Availability):**

Fase final em que o software é oficialmente lançado ao público em geral.

Considerado estável e pronto para uso em larga escala.

**Objetivo**:

Tornar o software amplamente acessível para clientes, usuários finais e organizações.

**O Teste e a Governança e a Qualidade de Software**

O teste de software é fundamental para assegurar a qualidade, detectando erros, defeitos e falhas. Um erro é o engano cometido por um desenvolvedor durante a criação do sistema, que gera um defeito no componente, tornando-o incapaz de funcionar corretamente. Esse defeito, quando exposto em execução, resulta em uma falha, ou seja, um desvio no comportamento esperado do software.

Ao identificar e corrigir erros antes que se transformem em falhas no uso real, os testes garantem que o sistema opere conforme esperado, promovendo maior confiabilidade e satisfação do usuário.

**Níveis de Teste: Modelo V**

FLUXO DE PROJETO: CICLO DE VIDA

Levantamento de Requisitos → Modelagem Funcional → Arquitetura Técnica de Sistema → Construção de Código

↓

Código Pronto

↓

FLUXO DE TESTE: CICLO DE TESTE

Teste Unitário de Código → Teste de Integração entre Componentes → Teste de Sistema Completo → Teste de Aceitação pelo Usuário

Nas etapas produtivas, avalia-se o cumprimento dos processos definidos para a fábrica de software e o gerenciamento do projeto. Verifica-se se o termo de abertura foi criado, se o código-fonte reutiliza componentes arquiteturais e se os métodos das classes foram detalhados em algoritmos. Valida-se ainda se os diagramas representam semanticamente o escopo funcional descrito no backlog do produto.

* **Levantamento de Requisitos**:

Verificação da lista de requisitos do backlog, realizada pelo analista de processos e de negócios, junto ao cliente.

* **Modelagem Funcional**:

Compatibilidade entre especificações (UML e MER) e requisitos, e consistência entre componentes (classes e casos de uso). Responsáveis: analista de sistemas, processos e negócios.

* **Arquitetura Técnica**:

Definição da organização modular, biblioteca de reuso, comunicação entre componentes e tecnologias para o ambiente de desenvolvimento. Responsáveis: arquiteto de solução e analista de sistemas.

* **Construção de Código**:

Aderência do código às especificações de arquitetura e sistema. O programador realiza essa etapa com suporte pontual do analista de sistemas.

Os testes verificam o status e registro de aplicação (data, hora e responsável).

* **Teste Unitário**:

Realizado pelo programador, avalia lógica, cálculos e formatação de entrada e saída.

* **Teste de Integração**:

Especialistas avaliam a comunicação entre componentes por meio de planos e roteiros baseados em parâmetros e protocolos.

* **Teste de Sistema**:

Simulação do funcionamento completo do sistema, com base em planos e casos de uso, verificando rotinas de negócio. Responsáveis: especialistas em testes alinhados com analistas de sistemas.

* **Teste de Aceitação**:

Usuários finais experimentam o software para aprovação e liberação.

**Tipos e Técnicas de Teste**

Os tipos de teste definem o propósito e a ênfase da avaliação:

* **Teste Funcional:**

Foca no funcionamento do software, analisando os serviços prestados, como geração, armazenamento, processamento, distribuição e proteção de informações.

* **Teste Não Funcional:**

Avalia atributos de qualidade como estética, facilidade de uso e desempenho, sem relação direta com a funcionalidade.

* **Teste Estrutural:**

Examina aspectos técnicos, como engenharia e arquitetura do software.

* **Teste de Mudança:**

Avalia componentes alterados e os impactos em partes não modificadas. Divide-se em:

* **Teste de Confirmação (Re-Teste):**

Verifica a eficácia, eficiência e efetividade dos componentes modificados.

* **Teste de Regressão:**

Garante que componentes não alterados continuam funcionando corretamente após mudanças.

Astécnicas direcionam como realizar os testes:

* **Caixa Preta:**

Avalia o software sem considerar sua construção interna, testando entradas e saídas esperadas.

* **Caixa Branca:**

Examina o código e a lógica interna, avaliando trechos específicos do programa

**Exemplo Prático: Software de Planilha**

* **Testes Funcionais:**

Tabular dados, somar, calcular percentuais.

* **Testes Não Funcionais:**

Beleza dos gráficos, facilidade para digitar fórmulas.

* **Testes Estruturais:**

Integração OLE com PowerPoint/MS-Word e banco de dados MS-SQL.

* **Técnica da Caixa Preta:**

Preencher células, aplicar fórmulas, verificar resultados.

* **Técnica da Caixa Branca:**

Inspecionar código-fonte, acompanhar parâmetros e variáveis.

**Teste Unitário**

Os testes unitários isolam a funcionalidade de um componente, analisando seu comportamento sem interferências externas. Em programação orientada a objetos, isso inclui testar métodos de classes específicas, mascarando chamadas externas e simulando retornos.

Para realizar testes funcionais de caixa branca, são utilizados métodos como:

* **Inicialização de Variáveis:**

Falhas intermitentes podem ocorrer pela ausência de inicialização. Antes de iniciar os testes, é preciso mapear variáveis, verificar seu uso inicial e corrigir problemas de não inicialização.

* **Complexidade Ciclomática:**

Método baseado em grafos, determina o número mínimo de testes necessários para cobrir caminhos independentes em pontos de decisão.

* **Teste de Enlace:**

Avalia loops em três condições: sem entrada, com entrada única e com reexecução. Testes começam do loop mais interno para o mais externo.

* **Teste de Limites:**

Avalia valores próximos aos limites inferiores e superiores, testando cenários críticos que impactam decisões lógicas.

* **Partição de Condição/Equivalência:**

Examina grupos de dados que levam às mesmas decisões, incluindo valores não numéricos, ampliando os casos definidos por testes de limites.

**Automação e IA na Criação de Testes**

Automação de testes unitários com scripts e IA generativa ajuda a otimizar o esforço de testes. Porém, a IA não substitui o planejamento criterioso. É essencial orientar adequadamente as ferramentas de IA, definindo o que, quando e como testar, garantindo uma cobertura eficiente e eficaz dos casos necessários.

**Programação Orientada por Testes**

Prática de desenvolvimento sugerida por metodologias ágeis como XP e Scrum. Ele incentiva os desenvolvedores a escreverem testes automatizados antes mesmo da implementação. Essa abordagem contrasta com o método tradicional de primeiro implementar e depois testar.

Suas vantagens são: garantir maior qualidade ao software, facilitar a evolução e manutenção do código, produzir um código mais organizado e a automatização e agilização do processo de testes.

**Ciclo do TDD:**

Crie um teste para avaliar o funcionamento da   
aplicação → Execute o teste e confirme a falha inicial → Desenvolva um código para passar no teste → Refatore o   
código → Selecione outro item de backlog e repita o processo.

**JUnit e Automação de Testes:**

O JUnit é uma ferramenta que permite a automação de testes unitários por script. Ele pode ser usado tanto em programação orientada a testes quanto em processos tradicionais de desenvolvimento.

**Benefícios do JUnit:**

* Padroniza o formato dos testes;
* Documenta casos de teste;
* Permite repetir testes múltiplas vezes;
* Facilita o reuso de testes entre programadores da equipe;
* Agiliza a execução e avaliação de testes;
* Analisa a cobertura lógica dos testes no programa.