**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.

**AI Testing**

As aplicações de IA possuem características que as distinguem dos sistemas tradicionais de informação, pois exigem a validação de conexões de ideias e simulações de reações humanas em um processo contínuo e dinâmico. Isso torna mais complexas as estimativas de esforço e a avaliação da qualidade.

Na IA, a validação se baseia em dois conceitos principais: a "verdade básica", que representa os dados considerados factuais, e as "respostas esperadas", que devem estar alinhadas a esses dados. Além disso, o processo de validação envolve métricas como:

* **Riscos de respostas inadequadas**:

Incluindo discurso de ódio, conteúdo violento, sexual ou que incite crimes e automutilação.

* **Qualidade das respostas**:

Analisando coerência, fluência, relevância e fundamentação para o contexto.

* **Interpretação de sentimentos:**

Verificando respostas adequadas a interações neutras, negativas ou positivas.

* **Jailbreaks:**

Quando a IA quebra restrições impostas às suas respostas.

**Métricas de Qualidade em IA**

Os modelos de IA preveem respostas com base em dados acumulados e podem ser avaliados por sua acuracidade (percentual de respostas corretas) e precisão (nível de exatidão das respostas positivas). Essas métricas são estruturadas com base nos seguintes cenários:

* **Verdadeiro Positivo (VP):**

Casos positivos corretamente classificados.

* **Verdadeiro Negativo (VN):**

Casos negativos corretamente ignorados.

* **Falso Positivo (FP):**

Casos negativos incorretamente classificados como positivos.

* **Falso Negativo (FN):**

Casos positivos classificados como negativos.

**Checklist para Validação (Ribeiro et al., 2020)**

Para garantir testes abrangentes, utiliza-se uma matriz de checklist que cobre aspectos como vocabulário, robustez, justiça, reconhecimento de entidades, lógica e papéis semânticos. Cada item do checklist ajuda a criar cenários de teste fundamentais, compondo um conjunto que simula diferentes situações necessárias para validação.

**Automação de Testes em IA**

Ferramentas modernas, como as usadas no ChatGPT-4, facilitam o registro de métricas e o aprimoramento contínuo dos sistemas. O processo de teste envolve:

1. Criar um conjunto de dados de teste, manual ou sintético.
2. Usar uma LLM para anotar as respostas da IA.
3. Agregar as anotações em métricas de desempenho no Estúdio de IA para análise e melhoria.

**Teste de Sistema com Execução Manual**

A modelagem funcional, frequentemente representada por diagramas de casos de uso em projetos que utilizam UML, serve como base para o desenvolvimento dos casos de teste. Essa abordagem possibilita a criação de testes focados em avaliar os aspectos funcionais do software por meio da técnica de **caixa preta**, que se concentra nos resultados esperados sem considerar a lógica interna do sistema.

**Etapas para Teste de Sistema:**

* **Casos de Teste Baseados em Casos de Uso:**

Identificar os cenários funcionais definidos nos casos de uso e desenvolver casos de teste que avaliem a conformidade do sistema com os requisitos funcionais.

* **Testes Não Funcionais:**

Avaliar questões como performance, verificando tempos de resposta e estabilidade sob diferentes condições e analisar usabilidade, garantindo que o sistema seja intuitivo e eficiente para os usuários finais.

|  |  |
| --- | --- |
| **Caso De Teste** | |
| **Identificação** | Um código ou rótulo único que identifica o caso de teste, geralmente formado por combinações de letras e números. |
| **Descrição de Objetivo** | Um texto claro e direto que explica o propósito do teste, ou seja, o que ele busca validar. Deve estar relacionado a um cenário específico de uso do sistema descrito nos casos de uso. |
| **Objeto Avaliado** | O programa, módulo ou funcionalidade que está sendo testado. Pode ser identificado pelo nome ou código associado no sistema. |
| **Preparação** | As ações ou condições necessárias antes de executar o teste. Inclui configurações iniciais, permissões, dados que precisam ser carregados no sistema e qualquer requisito prévio. |
| **Massa de Dados de Entrada** | Lista de valores ou variáveis que serão informados ao sistema durante a execução do teste. |
| **Massa de Dados de Saída Esperada** | Os resultados esperados após a execução do teste, descritos em termos de variáveis, mensagens exibidas ou mudanças no estado do sistema. |
| **Procedimento de Teste** | **Resultado Esperado de Cada Passo** |
| Uma lista de passos detalhados que orienta a execução do teste. É como um "manual de instruções" que descreve o que deve ser feito para simular o cenário de teste. | Uma descrição clara e objetiva do que se espera que o sistema faça ao executar os passos do teste. O resultado esperado pode incluir mensagens na interface, alterações em dados ou estados do sistema, ou outras evidências. |

**Teste Sem Critério Dirigido**

Os testes sem critério dirigido baseiam-se em procedimentos amplos e exploratórios, sem um foco estrito em valores ou condições pré-definidas. Nesse tipo de abordagem, as entradas podem ser selecionadas de maneira mais flexível, buscando identificar possíveis falhas no comportamento funcional do sistema. Esses testes são úteis em estágios iniciais de desenvolvimento ou quando o objetivo é avaliar a robustez do software em diferentes cenários, mesmo sem critérios estritamente definidos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exemplo: Caso De Teste – Sem critério dirigido** | | | | |
| **Identificação** | | | Teste UC-0101 | |
| **Descrição de Objetivo** | | | Avaliar o funcionamento do cadastro de um cliente (cenário principal do caso de uso, onde um novo cliente é cadastrado com sucesso) | |
| **Objeto Avaliado** | | | CADCLI.EXE (Alimentar esse dado quando aplicar o teste sobre o prog.final) | |
| **Preparação** | | | Listar os registros existentes na TABCLIENTE (tabela de registros de clientes do banco de dados- o nome oficial da tabela ainda pode ser desconhecido) | |
| **Massa de Dados de Entrada** | | | Um CPF válido e que não exista na Tabela de Clientes do banco de dados Um nome de pessoa física qualquer | |
| **Massa de Dados de Saída Esperada** | | | Mensagem “Cliente cadastrado” é exibida Cliente informado registrado na Tabela de Clientes do banco de dados (verificar o cadastro via SQL) | |
| **Procedimento de Teste** | | | **Resultado Esperado de Cada Passo** | |
| **1** | | O usuário digita todos os dados de entradas a Fornecer | **1** | Campos de CPF e Nome do Cliente são exibidos na tela com os dados informados |
| **2** | O usuário seleciona “Salvar Cliente” | | **2** | Tela exibe a mensagem ”Cliente Cadastrado” |
| **3** | Executar SQL para consultar o registro incluído na tabela | | **3** | Registro encontrado na tabela do Banco de Dados |

**Teste com Critério Dirigido**

Os testes com critério dirigido seguem uma abordagem estruturada, baseada em condições específicas e claramente definidas para a execução. Essa técnica é orientada por dados precisos e detalhados, como valores de entrada e estados esperados do sistema. O foco principal é validar o comportamento do sistema sob condições controladas e garantir que ele atende aos requisitos estabelecidos de maneira confiável. Esses testes são particularmente úteis para cenários críticos ou prioritários no ciclo de desenvolvimento.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exemplo: Caso De Teste – Com critério dirigido** | | | | |
| **Identificação** | | | Teste UC-0101 | |
| **Descrição de Objetivo** | | | Avaliar o funcionamento do cadastro de um cliente (cenário principal do caso de uso, onde um novo cliente é cadastrado com sucesso) | |
| **Objeto Avaliado** | | | CADCLI.EXE (Alimentar esse dado quando aplicar o teste sobre o prog.final) | |
| **Preparação** | | | Garantir que na TABCLIENTE (tabela de registros de clientes do banco de dados), não exista registro com o atributo CPF = 122.222.333-44 | |
| **Massa de Dados de Entrada** | | | CPF = 122.222.333-44, Nome = João da Silva Querequeque | |
| **Massa de Dados de Saída Esperada** | | | Mensagem “Cliente cadastrado” é exibida -Cliente informado registrado na Tabela de Clientes do banco de dados (verificar o cadastro via SQL) | |
| **Procedimento de Teste** | | | **Resultado Esperado de Cada Passo** | |
| **1** | | O usuário digita o CPF do cliente = 122.222.333-44 | **1** | Campo de CPF é exibido na tela com os dados informados |
| **2** | O usuário digita o nome do cliente = João da Silva Querequeque | | **2** | Campo de Nome é exibido na tela com os dados informados |
| **3** | O usuário seleciona salvar tabela | | **3** | Tela exibe a mensagem ”Cliente Cadastrado” |
| **4** | Executar SQL para consultar o registro incluído na tabela | | **4** | Registro encontrado na tabela do Banco de Dados |

**Como Planejar um Caso de Teste?**

Os testes com base em casos de uso são elaborados por analistas de sistemas ou de negócios, profissionais que possuem um profundo entendimento da lógica de negócio do software. Esses testes buscam simular os cenários previstos nos documentos de caso de uso, os quais explicam as funcionalidades do sistema do ponto de vista do usuário final e dos analistas envolvidos.

A abordagem consiste em criar casos de teste específicos para cada cenário de uso identificado no caso de uso. Cada cenário, por sua vez, representa um fluxo possível dentro de uma funcionalidade do sistema e requer a execução de uma sequência de passos que simulem as interações do usuário com o sistema.

**Principais Características dos Testes com Base em Casos de Uso:**

* Aplicável aos testes de sistema e aceitação.
* Derivado diretamente dos cenários de uso descritos nos documentos de caso de uso.
* Garante que cada cenário funcional tenha um caso de teste correspondente.

**Como Planejar um Caso de Teste com Base em Casos de Uso**

1. **Identifique os Cenários de Uso:** Analise os documentos de caso de uso e mapeie todos os fluxos funcionais.
2. **Crie Casos de Teste:** Para cada cenário de uso, elabore um caso de teste que simule os passos descritos no fluxo.
3. **Prepare os Testes Funcionais de Sistema:** Use os casos de teste como base para validar funcionalidades específicas e garantir que o software atende aos requisitos funcionais estabelecidos.

**Teste de Sistema Automatizado e Funcional**

A automação de testes permite a reexecução de casos de teste de forma repetida, com foco em aspectos funcionais (uso e operação das funcionalidades) ou não funcionais (performance, portabilidade, estresse, carga, tolerância a falhas e recuperação).

**Aplicando Estratégias de Teste: Smoke Test**

O Smoke Test ou teste de fumaça é uma avaliação inicial e simples para verificar se a aplicação responde adequadamente. Ele é executado logo no início do ciclo de testes para confirmar que o sistema está estável o suficiente para testes mais detalhados.

**Testes Automatizados por Record & Playback**

A técnica de Record & Playback utiliza ferramentas de automação que gravam as interações do usuário com a interface gráfica da aplicação. Essas ações são convertidas em scripts de teste, que podem ser reexecutados conforme necessário, reproduzindo exatamente o comportamento registrado.

**Elementos de um Script de Teste por Record & Playback:**

* **Dados de Teste:**

Dados de entrada e os resultados esperados.

* **Procedimento de Teste:**

Passos detalhados que descrevem a lógica de execução.

* **Ações Gravadas:**

Operações realizadas sobre a interface da aplicação.

**Vantagens:**

* Simplicidade e praticidade para criar testes rapidamente.

**Desvantagens:**

* Não gerencia dados armazenados na aplicação.Por exemplo, ao repetir um teste de inclusão, um cliente já existente pode gerar falhas indesejadas.
* Necessidade de preparação detalhada do ambiente e dos dados antes de reaplicações.

**Testes Automatizados com Abordagem Data-Driven**

A técnica Data-Driven Testing separa os dados de teste dos scripts de automação, armazenando-os em arquivos externos. Os scripts mantêm apenas os procedimentos e ações de teste genéricos, enquanto os dados específicos de cada caso de teste são acessados separadamente.

**Vantagens:**

* Manutenção Simplificada: Alterações nos dados ou novos casos de teste podem ser implementados sem modificar os scripts.
* Colaboração Facilitada: O projetista de teste elabora os arquivos de dados de teste e o implementador trabalha nos scripts de automação.

Essa abordagem oferece flexibilidade e eficiência, permitindo a criação de conjuntos de testes escaláveis e reutilizáveis, essenciais para projetos complexos e dinâmicos.