PERCURSO DE APRENDIZAGEM 4

UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO
EAD

TESTE E VALIDAÇÃO DE SISTEMAS

FERRAMENTAS DE AUTOMAÇÃO DE TESTES

Este trabalho está licenciado com uma Licença Creative Commons
Atribuição-NãoComercial-SemDerivações 4.0 Internacional.

Sumário

1. Tipos de ferramentas de automação de testes

2. Critério para automação e os cuidados para implantação

3. Análise de viabilidade da automação dos testes

4. Ferramentas de teste: processos de verificação e validação de software; gestão de testes, defeitos, indicadores de qualidade

Sumário clicável

Voltar ao sumário

Olá

Agora que você já sabe o que é teste e quais são os tipos, e também já conhece as técnicas de especificações e estratégias de teste, é o momento de aplicarmos este conhecimento por meio da automação de teste.

Neste Percurso de Aprendizagem, descobriremos os conceitos mais relevantes para automação de teste. Iremos estudar também sobre ferramentas que auxiliam na realização e no monitoramento dos mais variados tipos de teste, dentre eles os testes não funcionais, como teste de carga e testes de estresse, e veremos também como aplicar essas ferramentas para testes funcionais. Além disso, mencionaremos como coletar indicadores para balizar a evolução do desenvolvimento relacionado à qualidade de software.

Voltar ao sumário

1. Tipos de ferramentas de automação de testes

Assim como sabemos que alguns casos são altamente recomendáveis automatizar, também devemos estar conscientes que **nem tudo é passível da automação**.

IMPORTANTE

Portanto, situações em que as funcionalidades são pouco usadas, não convém implantar a automação de teste, pois será um desperdício de tempo e de esforço para pouco retorno. Uma outra situação em que não é indicado aplicar automação de testes é quando há funcionalidades que precisam ser inspecionadas visualmente. Além disso, quando falamos de **protótipos**, não é indicado que se construam **scripts de automação** para realizar testes neles, porque protótipos não são versões de software para realização de testes e o tipo de teste aplicado a eles é utilizado para construção da ideia do sistema. Quando vamos inserir novas funcionalidades em algum sistema, não é adequado utilizar automação, pois é preciso que sejam realizadas as **averiguações das regras de negócio**, bem como o **comportamento esperado** para aquela funcionalidade.

Por isso, para sabermos identificar a melhor estratégia e as situações que são cabíveis à automação de teste, é necessário conhecermos quais são as técnicas de automação bem como quais as ferramentas que podem dar suporte a esses procedimentos.

1.1 - Técnicas de automação

Quando falamos em automação de testes, é preciso entendermos as técnicas que estão embutidas nas ferramentas que proporcionam realizar as automações. Deste modo, podemos listar cinco técnicas: **Keyword-driven testing** (KDT); **Data-driven testing** (DDT); **Domain Specific Language** (DSL), **Recorde e playback** e **Negar validadores**.

Vamos conhecer cada uma delas?

1.1.1 - KDT (Keyword-driven testing)

É também conhecido como **teste orientado por palavra-chave**. A proposta sugerida por essa perspectiva de teste orientado por palavra-chave confere levantar cenários de testes que são baseados utilizando palavras-chave que representam de fato as ações que acontecem no sistema. Neste caso, os analistas de testes projetam cenários que seguem uma **estrutura em camadas**. A primeira é caracterizada pela **lógica abstrata** e depois vem a **camada estrutural,** que é a central de automação, portanto as automações utilizam várias funções e métodos.

Voltar ao sumário

Na estrutura KDT, a parte que realiza a execução do caso de teste é a estrutura principal; já as outras partes interagem com APIs e as ferramentas mais que possam existir no arcabouço da automação dos testes. Considerando a forma como a técnica KDT foi construída, permite-se aos analistas de teste construir variadas formas de cenários de testes automatizados, utilizando um conjunto fixo de palavras-chave. E nesta técnica não é exigida habilidade alguma de codificação.

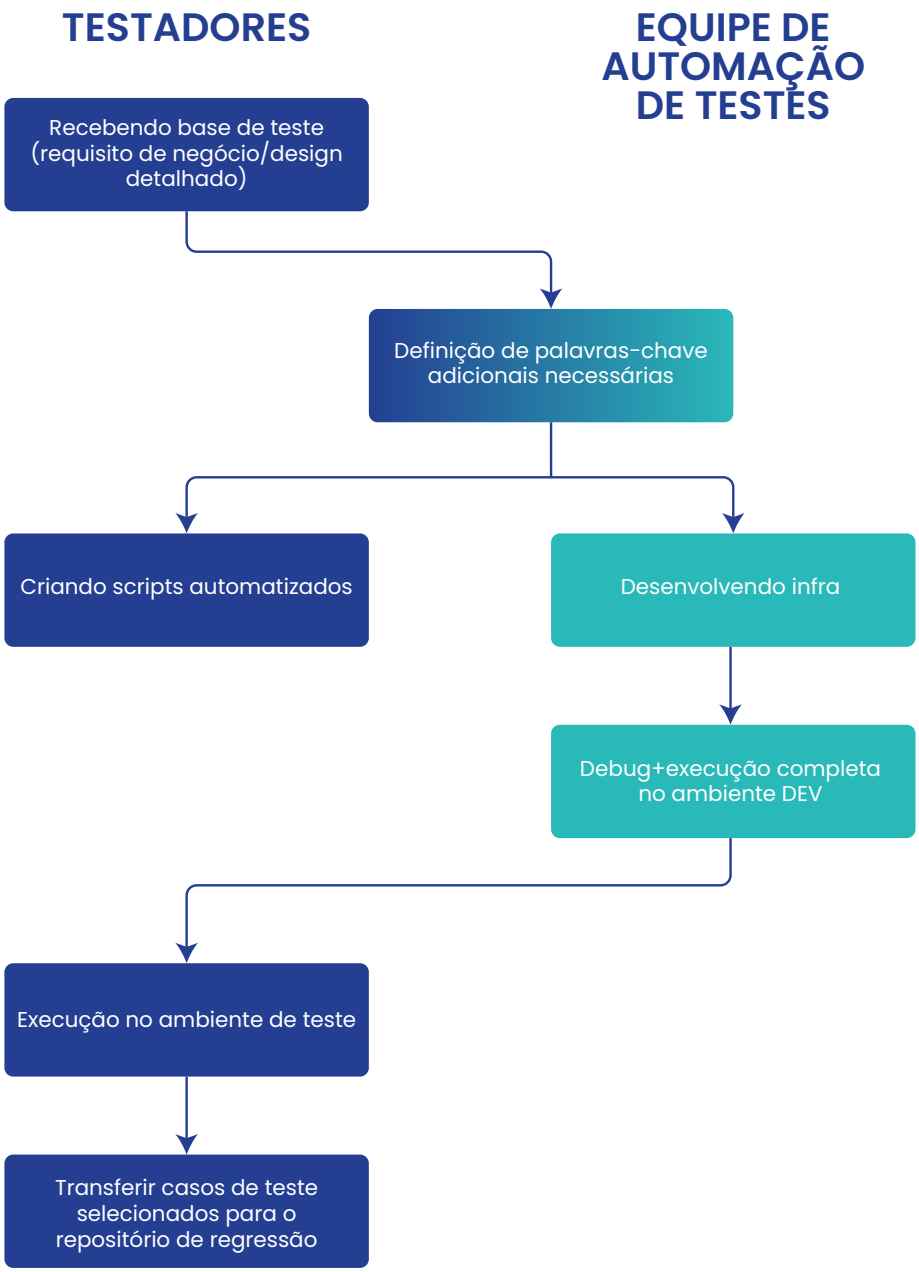
Os frameworks KDT são baseados em **três fatores principais**:

- Vocabulário (palavras-chave e frases) baseado nas necessidades e terminologia, às vezes chamado de Business Readable DSL (Domain Specific Language).
- Um analisador (compilador), que é responsável pela agregação de palavras-chave, verificação de sintaxe e orientação de dados. Normalmente, a entrada é baseada em um formato conhecido, como CSV, XLS, XML, SQL dinâmico e muito mais.
- Estrutura central de automação – um conjunto de funções de serviço genéricas e funções baseadas em elementos, que é responsável pela execução e relatório do teste. Em alguns casos, interage com ferramentas de terceiros para a execução real (KARDASHOV, 2015).

Na Figura 1, podemos ver a **representação do processo da técnica KDT**. Perceba que na técnica KDT indica-se que se tenham dois perfis de profissionais atuando na equipe de testes, que estão divididos entre duas responsabilidades: **testadores** (analista de teste) e o **time de automação de teste**.

Voltar ao sumário

Figura 1 – Representação da técnica KDT



Fonte: Elaborado por EAD Unifor (2023)

Para entender melhor

Ainda sobre a Figura 1, note que os testadores recebem as **bases de testes** que são os documentos de negócio e também o **design detalhado do projeto**. Depois disso, eles definem as **palavras-chave** a serem utilizadas junto com a equipe de automação de teste. Em momento posterior, o testador cria o **script automatizado**, enquanto a equipe de teste prepara o ambiente para serem executados os testes. A equipe de automação de teste faz o **debug do script** criado e executa todos os scripts em ambiente de desenvolvimento. Após isso, os testadores são responsáveis por realizar os testes em ambiente de teste e guardar os testes no repositório dos casos de testes para futuros testes de regressão.

Voltar ao sumário

Para que possamos fixar melhor como podemos modelar um teste baseado nesta técnica, considere o Quadro 1, que mostra a definição das palavras-chave para três diferentes contextos, referindo-se a alguns objetos e, na terceira coluna do quadro, vemos os valores que são testados.

Quadro 1 – Exemplo de aplicação do KDT

Palavra-chave	Objeto	Valor
Navegador aberto	Chrome	www.google.com
Definir texto	Caixa de texto de pesquisa	"O que é KDT?"
Aperte o botão	Botão Pesquisar	

Fonte: Elaborado por EAD Unifor (2023)

A atividade de definir palavra-chave é responsabilidade dos **testadores funcionais** sob a supervisão e controle dos especialistas em automação de testes. Já quem desenvolve as funções de utilidades são os **especialistas em automação**. Para desenvolver sequências, caracterizadas pelas funções definidas pelo usuário, os testadores funcionais e especialistas de automação trabalham em conjunto. Por fim, para desenvolver e executar casos de testes, é responsabilidade dos testadores funcionais.

O empenho que é exigido para se construir uma infraestrutura KDT de forma eficiente é muito grande, e em grande parte dos casos exige um time dedicado que pode não ser simples para as instituições de pesquisa e desenvolvimento que encontramos no cotidiano. A dificuldade para aprender o vocabulário proprietário (palavras-chave e frases), naturalmente passará a possuir o mesmo grau de esforço de aprender uma linguagem nova de script (Phyton/Javascript/VBS). Então, como podemos perceber, **memorizar palavras-chave proprietárias** pode ser mais complexo, às vezes, e com mais inflexibilidade do que uma linguagem de script que possa ser utilizada. Para se estabelecer a melhor estrutura KDT a ser utilizada, é preciso analisar adequadamente as ferramentas que já estão propostas no mercado, pois criar uma estrutura KDT do zero, certamente, não conferirá uma ideia inteligente.

Para se estabelecer a melhor estrutura KDT a ser utilizada, é preciso analisar adequadamente as ferramentas que já estão propostas no mercado, iniciando pela análise daquelas que possuem soluções de código aberto, como o **Cucumber** atrelado com **Appium** ou **Selenium**. Existem ainda possibilidades de ferramentas no mercado que já incluem a implementação de KDT, como, por exemplo, UFT ou TestComplete KDT.

> Ao estabelecer uma estrutura proprietária para automação de teste sem interface ou API, o uso de uma linguagem de script (Python/Javascript/VBS) deve ser considerado como a base KDT para testar a lógica de negócios e uma linguagem de programação de nível superior (C++/C#/Java) como a estrutura de automação central para execução de teste. Ao separar as duas tecnologias, é mais fácil superar a maioria dos problemas no mundo da automação de testes.

> Se o desenvolvimento ágil e a integração contínua são um fator, o KDT provavelmente não é a escolha ideal (a menos que a estrutura esteja lá para começar). O uso de ferramentas como Microsoft TFS, JUnit ou Robot Framework com a combinação de Jenkins ou Bamboo como servidores de construção/CI (Integração Contínua) geralmente será uma solução melhor (KARDASHOV, 2015).

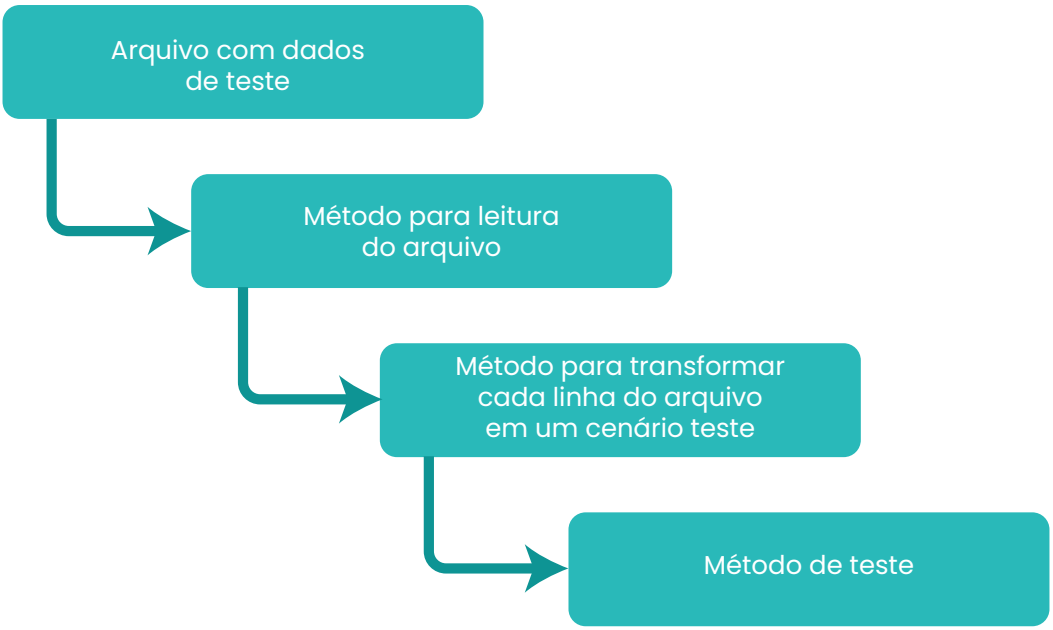
Para qualquer que seja a técnica escolhida de automatizar, o planejamento é parte crucial de um projeto desta natureza. É preciso saber qual é o alvo de automatizar, o que será automatizado e quem participará.

1.1.2 - DDT (Data-Driven Testing)

O teste orientado a dados ou DDT também é chamado de **teste parametrizado**. O DDT é um processo de teste de software que é usado para testar o software de computador aplicando algumas condições como entradas verificadas. O DDT se concentra em um conjunto de dados que estão presentes em bancos de dados para uma estrutura automatizada onde o teste é feito. Resolve problemas de processo grandes e demorados para a realização de testes separados.

Existem **várias formas de testes orientados a dados** nos quais os dados estão disponíveis em algum formato, como arquivos de valores separados por vírgula (CSV), planilhas do Excel, tabelas de banco de dados, matrizes de script e variáveis de tabela, assim como mostra a Figura 2.

Figura 2 – Processo do DDT



Fonte: Adaptado de Oliveira (2020) por EAD Unifor

Como você pode ver, o processo da técnica DDT se baseia em **quatro passos**:

- **Arquivo com dados de teste** – é aqui que se identifica qual será o local de onde a massa de teste será de consultada. Portanto, se seu repositório dos dados for um arquivo CSV, ele já deve estar com as informações dos dados inseridos. Se for uma API, deve-se estabelecer o arcabouço necessário para retornar os dados. Assim como os exemplos citados, os outros casos também devem permitir que os dados sejam acessíveis de alguma forma.

Voltar ao sumário

- **Método para leitura do arquivo** – este é um método criado para buscar todas as informações no arquivo de dados do teste. Para se construir esse tipo de método, é necessário que haja um driver ou uma biblioteca concatenada com alguma lógica de repetição. A intenção de criar esse método é que utilização dos conceitos de reutilização, fazendo com que toda vez que seja necessário utilizar essa funcionalidade, não haja a necessidade de ser desenvolvido novamente.
- **Método para transformar cada linha do arquivo em um cenário de teste** – como o próprio nome do passo diz, este é o momento durante o qual se cria um método para variar os cenários de testes. Cada linha do arquivo é entendida como um cenário e diversificado em outros.
- **Método de teste** - este é o método ao qual serão submetidos os dados coletados do arquivo dentro da perspectiva de entrada ou saída de dados, que depende de qual lógica foi utilizada no desenvolvimento.

IMPORTANTE

Um ponto importante sobre a técnica DDT é que dependendo da forma de desenvolvimento ao utilizar os conceitos de reúso, o método para leitura de arquivo e o método para transformar cada linha do arquivo em um cenário podem ser desenvolvidos em um único método.

Algumas das **vantagens de se utilizar DDT**, é que mantém dados individuais de scripts de teste e scripts de teste semelhantes que podem ser usados para vários conjuntos de dados de teste de entrada para que possamos obter o resultado automaticamente; ele **fornece ainda um cenário claro para os scripts de teste**. O DDT mantém todos os registros de forma muito eficiente e é muito fácil de acessar, além de diminuir a capacidade de ocorrência de erros. O DDT é muito fácil de processar para nós, criadores humanos.

1.1.3 - DSL (Domain Specific Language)

O **Domain Specific Language** (DSL), que em tradução livre significa **Linguagens Específicas de Domínio**, confere uma linguagem de programação que possui um alto nível de abstração, de forma que otimiza as soluções de problema de uma classe específica. Uma DSL é direcionada pelas definições e preceitos do campo ou domínio.

As linguagens como **Java**, **C** ou **Ruby** são linguagens de uso geral, portanto, possuem uma estrutura complexa. Já as linguagens específicas de domínio são mais fáceis de serem compreendidas. Para se construir uma DSL, é preciso envolver os especialistas da área para a qual está sendo criada a DSL em colaboração com os desenvolvedores. E normalmente essas DSLs são utilizadas por pessoas que não são programadoras e que possuem fluência no domínio em que a linguagem foi especificada.

Voltar ao sumário

Uma vez que você tenha uma linguagem e um mecanismo de transformação, seu trabalho, no aspecto específico do desenvolvimento de software coberto pela DSL, se torna muito mais eficiente. Desta forma, as abstrações se tornam mais amigáveis e específicas de domínio, **sem sobrecarregar o tempo de execução**, pois a estrutura interna que gera os códigos, como um compilador, remove as abstrações e gera o código com eficácia. Utilizando esta técnica, o código se torna menos confuso por conta das especificidades de programação e as preocupações com o domínio podem ser expressas de forma mais fiel possível na DSL.

A utilização de DSLs proporciona a comunicação eficiente entre as pessoas técnicas e as especialistas do domínio, pois as abstrações e as notações são todas alinhadas. Além disso, é essencial mencionar que o domínio pode ser uma área de negócio, como, por exemplo:

- Linguagem de domínio específico para seguros de vida, desenvolvida internamente em grandes empresas de seguros.
- Linguagem de domínio específico para simulação de combate.
- Linguagem de domínio específico para o cálculo do salário.
- Linguagem de domínio específico para o faturamento (JETBRAINS, 2000)

O uso de DSLs é um mecanismo de execução que torna a lógica do aplicativo expressa no código DSL independentemente da plataforma de destino. O uso de DSLs pode aumentar a qualidade do produto criado: menos erros, melhor conformidade arquitetural, maior capacidade de manutenção.

As DSL existem em **dois aspectos**: **as externas**, que criam uma linguagem própria, e **as internas**, que na verdade utilizam um subconjunto de instruções de uma linguagem já existente e utilizada no sistema.

Entre alguns clássicos exemplos de DSLs internas, temos o uso da Criteria do Hibernate, o uso do Ruby nos arquivos de build do Rake. Entre as DSL externas, temos as macros do Excel e o xml do Ant.

1.1.4 - Recorde e playback

A técnica Record & Playback consiste em, utilizando uma ferramenta de automação de teste, gravar as ações executadas por um usuário sobre a interface gráfica de uma aplicação e converter estas ações em scripts de teste que podem ser executados quantas vezes for desejado. Cada vez que o script é executado, as ações gravadas são repetidas, exatamente como na execução original.

Para cada caso de teste, é **gravado um script de teste completo** que inclui os dados de teste (dados de entrada e resultados esperados), o procedimento de teste (passo a passo que representa a lógica de execução) e as ações de teste sobre a aplicação.

Voltar ao sumário

Umas das **vantagens de se utilizar essa técnica,** é porque é muito simples e prática, sendo uma perspectiva ótima para testes que são executados poucas vezes. Entretanto, quando se pensa em suas desvantagens, a lista aumenta, tendo em vista o alto custo e a dificuldade de manutenção; a baixa taxa de reutilização; o curto tempo de vida e a alta sensibilidade a mudanças no software a ser testado e no ambiente de teste, e se houver uma alteração na interface gráfica da aplicação, poderá exigir a regravação de todos os scripts de teste.

1.1.5 – Negar validadores

Negar validadores nada mais é do que **preparar a sua ferramenta de teste** para realizar validações de cenários de exceções. Portanto, você deve programar a ferramenta de forma que se espera um valor negativo do comportamento. Se o comportamento negativo for verdadeiro, isso quer dizer que o teste foi atendido com sucesso.

IMPORTANTE

É importante ressaltar que cada uma dessas técnicas pode vir incorporada a diferentes ferramentas. O esperado é que se realize a viabilidade das ferramentas, o que será tratado no Circuito 3 deste Percurso de Aprendizagem, levando em consideração os tipos de testes e as ferramentas que são utilizadas para executar os diferentes tipos de projetos automatizados.

Voltar ao sumário

2. Critério para automação e os cuidados para implantação.

Tendo em vista que conhecemos as técnicas utilizadas para realizar automação de teste, bem como as ferramentas que podem ser consideradas como apoio da execução dos testes, é chegado o momento de entendermos alguns critérios que são utilizados para automação.

Além de entendermos os critérios que norteiam a automação de testes, é importante sabermos quais as preocupações que devemos ter ao realizarmos a implantação deste tipo de atividade em uma instituição.

Os critérios de automação estão ligados a verificar o quão aplicável e viável é inserir testes automatizados no contexto do desenvolvimento de software ao passo dos testes manuais. Assim sendo, a seguir está um conjunto de características que compõem as preocupações no momento de optar ou não por automação de teste (ISTQB, 2016):

- **Frequência de uso** - um ponto importante para averiguar se determinado teste é suscetível à automação é a frequência com a qual o teste é executado durante os lançamentos do sistema. Desta forma, se nas suas execuções de testes manuais um determinado cenário sempre é testado, então, entende-se que ele é um forte candidato a ser automatizado, pois há uma frequência de utilização do cenário de teste. Com isso, podemos afirmar que quanto mais lançamentos do sistema aconteçam, maiores serão as vantagens de possuir testes automatizados. Uma situação a ser considerada é que uma vez que os testes funcionais sejam abordados na automação de teste, eles conseguem ser utilizados para realizar os testes de regressão do sistema. Isso reflete em um alto retorno sobre o investimento (ROI) e ameniza os riscos de erros na base de código que já foi desenvolvida. Agora, se um determinado script de teste é utilizado apenas uma vez por ano e o sistema que está sendo testado for alterado dentro daquele ano específico, pode não ser uma boa ideia automatizar testes, visto que é um comportamento específico. Portanto, para este último caso, é mais indicado que se realizem testes manuais.
- **Complexidade para automatizar** - quando nos deparamos com sistemas complexos que precisam ser testados, é interessante pensar em automatizar algumas tarefas para que sejam gerados benefícios ao reduzir os esforços do testador em executar manualmente procedimentos repetitivos, maçante, longos e com pré-disposição em executar erros. Porém, é preciso ter cautela em identificar o que será automatizado, pois alguns scripts podem ser complexos e não lucrativos para automatizar. Alguns fatores que podem cooperar para esse segundo ponto é possuir um sistema que não seja compatível com as elucidações de testes automatizados à disposição no mercado; a pluralidade dos temas que são expostos como parte de uma execução do teste; a interatividade com interfaces externas e sistemas proprietários; situações relacionadas ao teste de usabilidade; o tempo que será necessário para construir os scripts de automação etc.
- **Compatibilidade e suporte de ferramentas** - a infinidade de tecnologias, plataforma, linguagens de desenvolvimento utilizadas para criar aplicações é um desafio à

parte para o testador. Depois ele precisa entender qual a ferramenta de teste que pode estar disponível ou não para suportar a plataforma trabalhada e até onde é benéfico utilizar a mesma. Existem algumas possibilidades para se obter as ferramentas de teste, sendo elas por meio de fornecedores comerciais, sistemas de código aberto e até mesmo sistemas que podem ser desenvolvidos pela equipe interna da instituição. Portanto, depende de cada instituição analisar as suas necessidades e entender quais os recursos que precisam para dar suporte às ferramentas de teste. Normalmente, os vendedores comerciais disponibilizam seus produtos de software oferecendo assistência de maneira paga, e quando se trata de empresas maiores existe um ecossistema de especialistas que pode suportar a implementação da ferramenta de teste. Já as ferramentas de código aberto oferecem uma outra perspectiva de mantimento, sendo impulsionada pelos fóruns online, nos quais usuários podem compartilhar informações e postar perguntas. Por outro lado, quando se trabalha com ferramenta de teste desenvolvida internamente, é confiada aos profissionais que compõem a equipe da instituição para prestar suporte à equipe de qualidade. Portanto, analisar a compatibilidade das ferramentas de teste versus o sistema que será submetido a esta ação é um passo crucial na automação, não sendo indicado superestimar os problemas que podem acontecer, caso não se identifiquem no projeto os níveis de compatibilidade de forma adequada.

Figura 3 – A compatibilidade das ferramentas de testes com o sistema ao qual serão submetidas é imprescindível.

Fonte: Freepik (2023)

- **Maturidade do processo de teste** - quando se pensa em automatizar testes, é preciso analisar se os processos de testes de software estão bem estabelecidos na instituição, pois uma vez que os processos estão bem estabelecidos, de forma que estejam bem estruturados, e que a equipe já trabalha disciplinadamente e de forma repetitiva, torna o trabalho de automação de teste um processo mais consistente, uma vez que quando é inserida automação no desenvolvimento, é necessário entender que deverá obrigatoriamente existir o gerenciamento do código automatizado e também de todos os componentes relacionados.
- **Adequação da automação para a fase do ciclo de vida do produto de software** - um ponto importante a ser considerado para automação de teste é em qual

Voltar ao sumário

momento o sistema se encontra no que diz respeito ao ciclo de vida do produto de software. Sabemos que há sistemas que demoram meses ou até anos para serem construídos e durante esse tempo há muitas mudanças e expansões podendo adicionar refinamentos para atender alguma necessidade do usuário final ou resolver defeitos. Quando estamos no início do desenvolvimento de um sistema, as soluções podem ser rápidas para suportar teste automatizado, como, por exemplo, os layouts e controles de telas podem ser otimizados ou aprimorados. Além disso, estabelecer automação para um ambiente dinâmico exige atenção contínua e, por muitas vezes, pode não ser eficiente ou eficaz, pois é difícil mudar qualquer item em um sistema quando ele já está em utilização em ambiente produtivo. Quando falamos de sistema desenvolvido no modelo cascata, o melhor momento de se implementar automação de teste é quando há uma estabilidade e se identifica o núcleo de funcionalidade. À medida que o tempo vai passando, os sistemas podem chegar a se tornar obsoletos, sendo redesenhados para outras tecnologias ou retirados do mercado. Deste modo, a automação de teste não é indicada para quando um sistema já se encontra em fase final do ciclo de vida do produto, pois haverá pouco valor agregado.

- **Sustentabilidade do ambiente automatizado** - os ambientes de teste para automação devem ser preparados com cautela, visto que precisam ser ajustáveis e adaptáveis às mudanças que podem ocorrer no sistema que está sendo testado. Percebemos essa característica por atributos como a capacidade de identificar e consertar de forma eficiente e eficaz os problemas que podem surgir na automação, simplicidade com que os componentes de automação podem absorver manutenções e em realizar inclusão de novos recursos com plenitude, de forma que o suporte para ambiente de automação seja acionado de forma acessível.
- **Controle do sistema (pré-condições, configuração e estabilidade)** - um analista de automação de teste deve ser capaz de perceber as características de controle e visibilidade do sistema para que possa ajudar a construir testes automatizados de forma objetiva. Se não houver controle por parte do analista de teste, as automações ficarão subordinadas às interações de interface do usuário, resultando em solução de automação de teste menos tratada.
- **Planejamento técnico em apoio à análise do ROI** - automatizar teste traz várias vantagens para a equipe de teste e consequentemente para a equipe como um todo do projeto. Porém, um investimento considerável de esforço e custo é sempre atrelado ao implementar um projeto de teste automatizado que seja eficiente e eficaz. É indicado que se faça um estudo antes da implementação dos testes automatizados para averiguar a rentabilidade do esforço e tempo que serão gastos neste processo. Para isso, é importante criar métricas que consigam calcular o retorno de investimento sobre as ações definidas para alcançar automação de teste, nas quais se deve pontuar qual é o potencial pretendido e qual é o resultado da implementação da automação de teste. São exemplos de alguns pontos que precisam ser analisados: dispor de ferramentas em ambientes de teste para realizar automação; corrigir dados dos testes e os casos de teste; escopo do esforço de automação de teste; papéis e responsabilidades; educação da equipe de teste para a mudança de paradigma; esforço paralelo; colaboração entre o time; relatório de automação de teste.

Voltar ao sumário

Para que haja uma transição saudável em mudanças de *mindset* com relação a deixar os testes manuais para implementar os testes automatizados, é necessário também tratar algumas áreas, sendo elas:

- **Disponibilidade de ferramentas no ambiente de teste para automação de teste** - todo o aparato de ferramentas que é devidamente selecionado necessita ser instalado e verificado para que esteja em pleno funcionamento no ambiente de teste. Algumas ações, como realizar download de alguma versão específica do software utilizado, configuração da instalação apropriada, são necessárias para suportar os testes do sistema em questão e atestar que a automação irá funcionar de forma correta no ambiente de teste versus no ambiente de desenvolvimento da automação.
- **Correção de dados de teste e casos de teste** - um dos pontos cruciais para realizar testes automatizados é averiguar a qualidade dos dados que são utilizados. Corrigir os dados de testes manuais e os casos de teste é um ponto primordial para estabelecer que utilização da automação beneficiará positivamente o projeto atingindo os resultados esperados. As execuções dos testes automatizados necessitam das informações de dados explicitamente onde são utilizados nas entradas, navegação, sincronização e validação.
- **Escopo do esforço de automação de teste** - uma forma inteligente de iniciar a automação de teste é limitando o escopo que será feito em caráter inicial. Isso ajuda a moldar um sucesso inicial de automação bem como colher *feedback* sobre alguma questão técnica existente. Um projeto piloto ajuda a identificar as funcionalidades mais importantes no sistema para que represente de forma consistente os ganhos ao se optar pela automação. Trabalhar em cima de projetos pilotos ajuda a coletar as lições aprendidas, bem como determinar melhorias nas estimativas futuras em relação a tempo, prazo e custo, além de verificar as áreas que se fazem necessárias na utilização de recursos técnicos de formas específicas. Além disso, adotando uma estratégia de implementar um piloto, torna-se rápido averiguar se a automação terá sucesso. Outro ponto importante a ser considerado nessas automações é que precisa ter cuidado ao escolher os casos de testes que serão utilizados no projeto. A ideia é escolher casos de testes que exigem um esforço menor para automatizar, mas que vão retornar um valor agregado considerável. Outra forma de iniciar projetos pilotos é aplicando automação em testes regressivos ou testes de fumaça, pois esses testes costumam ser executados frequentemente, então é esperado que se identifique o valor agregado de forma mais rápida. Com esses projetos pilotos de automação, surge como ponto importante o abandono do esforço manual e também os graves problemas que possam ser identificados. Neste sentido, um novo horizonte é trabalhado estabelecendo novas extensões para onde se pode colocar esforço e dinheiro. No projeto piloto devem ser priorizados os testes mais críticos para a empresa, portanto aqueles testes que são muito complicados para ser automatizados devem ser evitados no plano piloto. O foco do projeto piloto é mostrar que automação de teste funciona de uma forma que agregue valor ao negócio da empresa.
- **Educação da equipe de teste para mudança de paradigma** - existem diversos tipos de testadores, que mudam de acordo com suas habilidades. Alguns são mais especialistas no domínio do sistema em relação ao usuário final ou são mais envolvidos como analista de negócio; outros possuem *skills* mais técnicos capazes

de perceber melhor as características de arquitetura do sistema. Portanto, para que os testes possam ser eficientes, é preciso haver uma mistura de profissional com esses diferentes atributos. Quando a equipe de teste possui mais conhecimento na automação, a equipe se torna mais especializada. Desta forma, é preciso orientar a equipe, assim quando as primeiras ações de automação são executadas, de forma que possa evangelizar toda a equipe sobre a perspectiva de automação, reduzindo possíveis problemas que sejam gerados pela ansiedade da equipe em saber qual será o seu papel.

- **Papéis e responsabilidades** - é necessário que todos da equipe sejam envolvidos para realizar a automação de teste. Cada um terá o seu papel, podendo até mesmo ser um designer de casos de teste, ou pode ser a pessoa que faça as implementações ou até mesmo alguém que possa realizar a manutenção do ambiente de teste que tem uma característica mais técnica. Isso significa dizer que é necessário inserir analistas de testes que tenham mais proximidades com programação e conhecimentos mais técnicos. Os esforços que são colocados em cima dos projetos para automação devem trabalhar de forma mais simples e direta possível, pois este ambiente precisa ser usual para as pessoas técnicas e não técnicas. As pessoas que são especialistas no domínio precisam estabelecer relatórios, revisando-os para averiguar se as funcionalidades do aplicativo estão de acordo com o esperado; já os integrantes da equipe que são mais técnicos têm o dever de garantir que o ambiente automatizado está em execução de forma correta. É importante mencionar que esses especialistas técnicos podem também ser um desenvolvedor com um *skill* voltado para testes.
- **Cooperação entre Desenvolvedores e o Engenheiro de Automação de Testes** - como em qualquer projeto, a automação só funcionará de forma satisfatória quando houver cooperação entre todos os envolvidos, principalmente dos desenvolvedores e analistas de qualidade, que precisam atuar em conjunto para garantir o sucesso do produto. Estes dois profissionais terão que atuar de forma muito próxima, sendo os desenvolvedores apoiando em informações mais técnicas e o analista de teste como promotor da qualidade, seja do código do desenvolvedor ou possuindo visão crítica sobre a testabilidade dos projetos.
- **Esforço paralelo** - por vezes a automação funciona de forma paralela aos testes manuais para que se possa fazer a transição de forma tranquila. À medida que vai evoluindo, os scripts automatizados substituem os scripts manuais, mas antes é necessário fazer uma validação manual do script automatizado para garantir que todas as situações dos cenários de testes executados manualmente estão ocorrendo de forma correta.
- **Relatórios de automação de teste** - é indispensável que se gerem relatórios de execuções de testes automatizados. E para isso existem vários tipos com diferentes perspectivas que podem incluir os critérios que são utilizados para aprovar ou reprovar um script, estatística das execuções de testes de forma geral e também o desempenho da automação de teste, bem como os problemas que possam surgir.

1.1 – Cuidados para implantação de automação

Sabemos que optar pela implantação de automação de testes em uma organização pode ocorrer de não funcionar adequadamente, pois assim como a própria construção da codificação de um sistema, este também é um projeto.

Voltar ao sumário

ATENÇÃO

Um dos cuidados que devemos ter é realizar um **projeto piloto** antes mesmo de implantar de forma oficial a automação de teste. Optar pela realização de um piloto facilita o entendimento do que foi planejado em relação à automação de teste perante os benefícios desejados. Desta forma, é através da implementação de um projeto piloto que conseguimos identificar se a solução da automação é adequada para a realidade da empresa, bem como podemos entender se as soluções propostas se adequam aos processos, métodos e ferramentas existentes.

Outra perspectiva de cuidado quando se realiza um piloto da automação, antes de ser efetivamente realizadas as mudanças, é que se faz necessário o entendimento das necessidades dos testadores perante as interfaces que devem ser projetadas.

As equipes devem criar procedimentos de padronização para a manipulação, o gerenciamento, a forma de armazenar e como manter as soluções de automação, bem como os ativos de teste. Nestas preocupações podemos citar, por exemplo, estabelecer as nomenclaturas para os arquivos de teste, dentre outras possibilidades que podem ser organizadas utilizando um padrão a ser definido.

Os gerentes que estão realizando a gestão de projetos de automação devem pensar em métricas e processos de medição para realizar o monitoramento da automação de teste, abordando temas como usabilidade, manutenção e escalabilidade.

Voltar ao sumário

3. Análise de viabilidade da automação dos testes

Agora que você já tem em mente todos os critérios e áreas que são necessários para se analisar antes de realizar o processo de aplicação de automação de teste, é chegado o momento de falarmos sobre como esses critérios podem ser utilizados para averiguar a **viabilidade da automação**.

Primeiramente, devemos conhecer o processo genérico de uma solução de teste automatizada. Este processo pode ter alguma especificidade quando falamos de diferentes organizações, no entanto este deve ser obrigatoriamente bem constituído e coerente com o objetivo de fornecer os resultados esperados com a meta de minimizar a energia gasta para executar testes, caso fossem executados de forma manual.

Portanto, uma estrutura que podemos mencionar de forma genérica para a construção de **solução de automação** é a que segue:

1. **Criação dos cenários de teste** - todos os casos de testes devem ser repensados, caso já existam ou precisam ser estruturados e registrados em alguma ferramenta de gestão de teste, de forma que seja fácil correlacionar com os requisitos do sistema. É com base nesses casos de teste contendo todos os passos necessários e os resultados esperados expostos que o analista de testes automatizados irá criar os scripts de automação.
2. **Análise do âmbito a automatizar** - é de importância incalculável que se analise o que será automatizado, porque é neste momento que se definem os testes que serão automatizados para o sistema em questão criado e, levando em consideração os critérios criados no passo anterior, é realizada a seleção dos testes que serão atendidos.
3. **Criação dos scripts automáticos** - é neste momento que se realiza a construção do script com base no que foi especificado manualmente. Nos casos de teste, são adicionadas algumas ações que devem ser validadas de acordo com os resultados esperados. Estes scripts devem seguir intrinsecamente o que está escrito nas ações do teste e deverão ser reportados a alguma situação inesperada, de forma que esses erros possam ser tratados. Por exemplo, alguma mensagem de erro no sistema deve ser sinalizada quando encontrado.
4. **Validação dos scripts** - execuções prévias dos testes automatizados precisam ser realizadas. Pode-se inclusive organizá-las em forma de grupos lógicos sequenciais atendendo a ordem pela qual irão ser executados quando já estiverem em funcionamento, com o intuito de simular as situações reais de execução. O objetivo é predizer os problemas que possam surgir durante o momento que a solução automatizada estará em operação. É a partir daqui que os scripts estarão prontos para serem implantados no ambiente de destino deles.

Voltar ao sumário

5. **Execução de scripts automáticos** - após as validações, os scripts devem ser transportados para o ambiente ao qual eles foram destinados. E aqui há a atuação do analista de teste para serem executados os casos de testes automatizados no ambiente para o qual eles foram destinados juntos com a massa de teste e qualquer outra condição e/ou configuração de testes que sejam importantes para garantir o bom funcionamento do script automatizado.
6. **Manutenção do script de teste** - este passo compreende todas as atualizações que possam ser solicitadas em consequência da disponibilização das versões novas do sistema, mudanças de ambiente ou mesmo adaptações do sistema.

Sabendo disso, podemos classificar as análises de **viabilidade de automação** como macro e micro. A **viabilidade macro** está ligada com o quão madura uma empresa se encontra para aderir aos processos de teste com um valor baixo e alta eficiência. Ela está é essencialmente conectada à gerência sênior da empresa. Já a **viabilidade micro** se refere aos cenários de teste e quão veloz se consegue recuperar o investimento em dado cenário. Ela está intrinsecamente relacionada aos analistas de testes.

Para **planejar uma automação de testes,** é necessário saber o quê e quando alguma coisa deve ser automatizada de forma a estabelecer os cronogramas de entregas, preparar os usuários dos testes, monitorar a execução dos scripts, padronizar os processos de desenvolvimento, fomentar o reúso de componente (bibliotecas de funções e trechos de script) e desenvolver aplicações testáveis.

ATENÇÃO

Um projeto de automação de teste deve seguir os mesmos **padrões de desenvolvimento de uma aplicação normal**, precisando estar coerente todo o processo, desde os levantamentos dos requisitos do teste, passando pela codificação (criação dos scripts), testes (dos scripts) até a aceitação por parte dos usuários (das pessoas que utilizarão o script automático para testar).

Além disso, realizar o treinamento das pessoas que irão utilizar os **scripts automatizados** é essencial, pois se não forem orientados a utilizar de forma adequada os scripts automáticos, os usuários não poderão utilizar de forma efetiva para executar as tarefas.

Voltar ao sumário

Figura 4 – Antes de automatizar, pergunte-se o quê e, porquê e quando deve ser automatizado

Fonte: Freepik (2023)

IMPORTANTE

Um ponto importante aqui é que, além de executar os scripts, é de extrema importância **interpretar os resultados gerados** a partir dessas execuções de uma forma precisa, visualizando, por exemplo, quantidade de defeitos encontrados, quantidades de casos de testes executados, tempo de execução dos testes, etc. Uma observação interessante sobre esse tema é que, até uns quatro ou cinco anos atrás, os testadores tinham uma certa resistência em aderir automação de teste por medo de serem desligados de suas funções, porém hoje esse paradigma foi quebrado e a automação é vista como algo que pode contribuir positivamente para o trabalho dos analistas de testes, ficando em evidência que mesmo que existam testes automatizados, é muito difícil abolir os testes manuais de forma definitiva, pois algumas verificações precisam ser feitas necessariamente utilizando a segunda perspectiva de teste.

Um outro ponto importante é que para a automação **não basta criar apenas scripts automatizados sensacionais**, é necessário analisar se usuários estão verdadeiramente aproveitando-se da melhor forma possível em relação aos benefícios da automatização, pois desenvolver scripts automatizados tem custo elevado, e para um script ser rentável é necessário que ele realmente seja executado, em vez da sua versão manual. Então, nesse caso, é interessante definir metas em relação à utilização dos scripts automatizados.

Voltar ao sumário

Ainda como parte do planejamento, é necessário estabelecer o **reúso de componentes**, estabelecendo processos que sejam compartilhados dentro da aplicação, que são particionados em diferentes componentes de uma forma que seja possível **reutilizar em múltiplos scripts**. Isto simplifica a **codificação**, porque se deve apenas realizar a chamada do componente em vez de codificar a mesma função por várias vezes, diminuindo os custos de desenvolvimento, além de facilitar a manutenção, porque simplesmente é atualizado o componente para consertar todos os scripts.

De uma forma geral, as ferramentas de automação precisam acessar de forma direta, simples e consistente os objetos do teste da aplicação, pois os custos de método de acesso avançado são muito superiores aos métodos diretos e a manutenção também fica mais cara.

Existem ainda algumas formas de realizar o estudo de viabilidade, de forma a trazer números sobre os investimentos que são realizados. As duas formas mais conhecidas para analisar investimentos são a **Cost Benefit Analysis** (CBA) e **o Return On Investment** (ROI). A CBA, às vezes chamada de análise de custo-benefício, é uma abordagem sistemática para estimar os pontos fortes e fracos das alternativas. É usada para determinar as opções que fornecem a melhor abordagem para obter benefícios, preservando economias em, por exemplo, transações, atividades e requisitos funcionais de negócios. Uma CBA pode ser usada para comparar cursos de ação concluídos ou potenciais e para estimar ou avaliar o valor em relação ao custo de uma decisão, projeto ou política. É comumente usado para avaliar decisões de negócios ou políticas (particularmente políticas públicas), transações comerciais e investimentos em projetos. **A CBA tem duas aplicações principais**:

1. Determinar se um investimento (ou decisão) é sólido, verificando se – e quanto – seus benefícios superam seus custos.
2. Fornecer uma base para comparar investimentos (ou decisões), comparando o custo total esperado de cada opção com seus benefícios totais esperados.

Ao aplicar CBA no contexto de um projeto de automação de teste, ajuda a apontar os scripts que darão o maior retorno ao serem automatizados. Alguns exemplos de métricas que podem ser incluídas nos cálculos da CBA, mas não se limitam, são:

- Número de casos de teste a serem substituídos;
- Tempo de desenvolvimento dos scripts manuais e automatizados;
- Ciclos de teste futuro;
- Número estimado de execuções por ciclo de teste;
- Tempo de execução manual e automatizado.

Já o ROI, conhecido também como **retorno sobre o investimento ou taxa de retorno**, diz respeito a quanto você ganha de dinheiro em relação ao que é investido em determinado produto e/ou projeto.

Agora que vimos como se calcula a viabilidade de projetos de automação de teste, é interessante mencionarmos sobre como escolher a melhor ferramenta para que os testes sejam desenvolvidos.

Voltar ao sumário

Primeiro, devemos ter a consciência de que **nenhuma ferramenta poderá resolver todos os problemas relacionados** à **automação de teste**. Por mais que você tenha a melhor ferramenta recomendada no mercado, não adianta se não há pessoas treinadas para utilizá-la. As ferramentas não possuem autonomia de criação de scripts, no caso os scripts só conseguem entender as que foram codificadas para serem feitos. E é claro, **nem tudo deve ser automatizado,** como comentamos em momento anterior, então os ciclos de testes deverão possuir uma mistura dos dois tipos de testes, tanto manuais como automatizados.

Depois de realizar a viabilidade macro, é chegado o momento de **identificar quais ferramentas estão aptas no mercado que possam atender** à**s necessidades identificadas no projeto de automação**. É indispensável levar em conta as tecnologias suportadas versus as necessidades da empresa, a facilidade de uso, a integração com outras ferramentas, o custo de licença ao longo dos anos, a adaptação a diferentes cenários, sendo eles simples ou complexos etc.

1.1 – Conhecendo as ferramentas que podem ser viável para o seu projeto de automação

A tecnologia está sempre em evolução e constantes mudanças de paradigmas, rotinas, detalhes, já fazem parte do dia a dia dos profissionais que atuam nesta área. Com as ferramentas para automação de teste não é diferente, **elas sofrem atualizações constantemente,** e algo que tratamos hoje como ponta de linha, daqui a poucos anos ou meses já pode ser tratado como obsoleto. Por isso, é importante o profissional ficar atento às novas tecnologias.

IMPORTANTE

Apesar disso, é importante frisar que algumas estruturas estabelecidas não sofrem alterações, como, por exemplo, existem ferramentas que dão suporte exclusivamente para **testes funcionais** e outros que **dão suporte para os testes não funcionais**. Desta forma, os testes de unidade, por exemplo, possuem ferramentas específicas para que sejam executados esses procedimentos, assim como o teste de carga possui uma ferramenta apropriada para se estabelecer os melhores resultados e assim sucessivamente.

1.1.1 – Ferramentas para testes unitários

Os testes unitários se preocupam em averiguar as partes menores que compõem um sistema, como, por exemplo, os métodos, funções, dentre outros. A seguir, podemos conferir algumas ferramentas que podem ser utilizadas para apoiar a sua automação:

- **JUnit** - essa ferramenta é, de longe, uma das mais utilizadas atualmente. O JUnit é open-source e trata-se de um framework criado por Erich Gamma e Kent Beck e possui estrutura para criação de testes para a linguagem Java.
- **PHPUnit** - foi criado por Sebastian Bergmann e é um framework de teste de unidade para a linguagem PHP. Ele faz parte da família XUnit (nome coletivo de várias estruturas de teste de unidade que derivam sua estrutura e funcionalidade do SUnit de Smalltalk).

Voltar ao sumário

- **NUnit** - é uma ferramenta de código aberto utilizado para testes de unidade que suporta o .NET Framework e Mono. O propósito dele é muito parecido com JUnit para o Java, além disso o NUnit é da família XUnit.
- **TestNG** - esta ferramenta é direcionada para a linguagem Java. O JUnit e NUnit inspiraram a sua criação e seu criador foi Cédric Beust. O TestNG cobre uma variedade muito grande em relação às categorias de testes, disponibilizando funções muito eficientes e que têm uma usabilidade fácil para ser utilizadas em testes de unidade, funcional, ponta a ponta, integração etc.
- **Jest** – esta ferramenta foi construída em cima do Jasmine, e é feita a manutenção pela Meta. Ela foi construída por Christoph Nakazawa EEE direcionada para JavaScript. O objetivo é ser simples e disponibilizar suporte para vários aplicativos na web. Esta ferramenta pode ser utilizada por projetos que utilizem Babel, TypeScript, Node.js, React, Angular, Vue.js e Svelte.
- **Unittest** – é uma estrutura de teste de unidade que foi originalmente inspirada pelo JUnit, e é semelhante às principais estruturas de teste de unidade em outras linguagens. Ele suporta automação de teste, compartilhamento de configuração e código de desligamento para testes, agregação de testes em coleções e independência dos testes da estrutura de relatórios para linguagem Python.
- **PyUnit** - é uma biblioteca para realização de testes unitários em Python baseada na arquitetura xUnit. É a forma mais difundida para realizar a prática de testes unitários pela comunidade Python.

1.1.2 – Ferramentas para automação de testes não funcionais

Os testes não funcionais dizem respeito a verificar se seus requisitos não funcionais do sistema estão de acordo com o esperado. Estes testes estão diretamente ligados com a perspectiva de que o produto de software foi feito com a melhor qualidade. Portanto, seguem alguns exemplos de ferramentas utilizadas para este fim:

- **JMeter** - também conhecido como Apache Jmeter, possui código aberto e é construído completamente na linguagem Java. Muito utilizado para medir o desempenho e criar cenários de testes para verificar o comportamento funcional do teste. Hoje, é uma das principais ferramentas que realizam testes de carga e de estresse em recursos estáticos ou dinâmicos oferecidos por sistemas computacionais.
- **Gatling** – é uma ferramenta que pode ser utilizada para testes de carga e desempenho, tem código aberto e é baseada em Scala, Akka e Netty. Ela foi lançada pela primeira vez em sua forma estável no dia 13 de janeiro de 2012. O criador desta ferramenta, conhecido como Stéphane Landelle, criou em 2015 uma empresa exclusiva para desenvolver a ferramenta para código aberto.
- **Security AppScan** – em tempos passados, era conhecido como IBM AppScan, e confere uma linha de ferramentas para monitorar e realizar testes de segurança em aplicativos desktop e web.
- **Acunetix Vulnerability Scanner** - é uma ferramenta concebida para replicar técnicas utilizadas por hackers na localização de vulnerabilidades críticas em aplicações web.

Voltar ao sumário

1.1.3 – Ferramentas para testes funcionais

Essas ferramentas proporcionam realizar os testes de forma automatizada para verificar os requisitos e especificações funcionais:

- **Appium** - é uma estrutura para automatizar testes que executam scripts de aplicativos nativos, móveis-web ou até mesmo aplicativos híbridos no Android ou iOS utilizando algum webdriver. Além disso, esta ferramenta possui código aberto e funciona com praticamente todas as linguagens de programação e pode ser conectada a nuvem ou a servidor local.
- **Robotium** - é uma ferramenta para testes de software que possui código aberto e utilizado para escrever casos de testes automáticos para diferentes tipos de teste, a exemplo dos testes de caixa-cinza voltados para Android. Com esta ferramenta, é possível trabalhar em diversas modalidades de testes, fazendo com que o desenvolvedor e o analista de teste possam variar a automação dos seus testes funcionais, sistema e a aceitação, permitindo verificar comportamento de telas e de *timings*, por exemplo.
- **Selenium** - esta é uma ferramenta amplamente utilizada pelo mercado corporativo para testar sistemas web. O selênio é utilizado para criar scripts de testes funcionais sem que seja necessário aprender uma linguagem de script de teste. É compatível com várias linguagens como: Java, Ruby, PHP, Python, C#, Groovy, etc (GONÇALVEZ *et al*., 2019).
- **Telerik Test Studio** - é uma estrutura para realizar automação de teste que suporta testes funcionais, de desempenho, de carga e testes de API RESTful. A empresa que desenvolveu essa ferramenta se chama Telerik, e é a mesma indicada para sistemas web e desktop.
- **TestComplete** - é uma ferramenta para construir testes automatizados funcionais e foi elaborada pela SmartBear Software. Esta ferramenta permite aos profissionais de teste a possibilidade de elaborar testes automatizados para sistemas que funcionam de forma desktop no sistema operacional Microsoft Windows. Também pode ser direcionada para sistemas web, Android e iOS. Algumas de suas possibilidades são: gravação e reprodução, visualização dos testes, avaliação dos scripts, reconhecimento de objetos, atualização da interface do usuário automática, suporte para diversas linguagens e tecnologias.
- **Testing-Whiz** - esta ferramenta de testes de software pode ser usada para bancos de dados, aplicativos mobile e API, dependendo do pacote adquirido. Ela é facilmente integrada a ferramentas de gerenciamento e de busca de erros. Além disso, sua arquitetura sem código é um diferencial que facilita o uso e os testes podem ser guiados por riscos, palavras-chave ou dados.
- **WatiR** - é uma família de bibliotecas Ruby de código aberto para automatizar navegadores da web. Ele atua com o Internet Explorer, Firefox, Chrome, Opera e Safari, e está disponível no RubyGems (gerenciador de pacotes para a linguagem de programação Ruby). Houve uma contribuição muito grande do Bret Pettichord e Paul na construção.
- **Cucumber -** é uma ferramenta utilizada para realizar testes de aceitação automatizados que foram criados em um formato BDD.
- **Cypress** - é uma ferramenta de teste de front-end para aplicativos da web. É executado no Windows, Linux e macOS. O aplicativo Cypress é um software de código aberto lançado sob a licença MIT, enquanto o Cypress Cloud é um aplicativo da web.

4. Ferramentas de teste: processos de verificação e validação de software; gestão de testes, defeitos, indicadores de qualidade

Nos circuitos anteriores aprendemos sobre os conceitos **fundamentais para automação de teste**, também aprendemos sobre os **critérios e as áreas** que são necessários serem tratados durante a migração dos testes manuais para os testes automatizados, e vimos como realizar um estudo de viabilidade para implantar automação.

No primeiro Circuito deste Percurso de Aprendizagem, aprendemos sobre algumas ferramentas utilizadas para realizar os testes de forma automática, citamos ferramentas que podem ser empregadas nos testes de unidade, **testes de desempenho e de carga**, **teste de segurança**, além de falarmos sobre as ferramentas que são utilizadas para desenvolver os **testes funcionais**.

Neste Circuito, abordaremos as ferramentas que podem ser utilizadas para realizar o **gerenciamento do teste**. Estas ferramentas proporcionam o controle da criação e versionamento dos casos de testes, bem como as suas execuções e possíveis falhas. Através delas é possível **gerar relatórios para análise** do que foi testado, além disso, há ferramentas que podem ser exclusivamente para execuções manuais e também existem aquelas em que é possível conectar com outros sistemas para realizar o monitoramento e execuções de testes automatizados.

1.3 – Ferramentas para gestão de testes

- **Testlink** – o teste link é um sistema web direcionado para teste de software que tem por objetivo promover a fluidez de testes e garantir a qualidade de software. Este sistema foi construído e sustentado por muitas equipes no decorrer de sua existência. O sistema proporciona o suporte para casos de teste, plano de teste, teste de unidade e, dentre outros, é possível encontrar também funcionalidades que dão estrutura para gerar relatórios e estabelecer estatística. O principal fomento desta ferramenta é o acesso pela web, depois melhora a acessibilidade e a execução de testes por equipes que atuem remotamente. O teste link possui estrutura nativa para os bancos de dados MySQL e PostgreeSQL. Porém, o seu adequado funcionamento está intrinsecamente ligado à arquitetura web.
- **AzureDevops** – o Azure DevOps trabalha a perspectiva de colaboração entre as equipes sobre um determinado conjunto de processos que podem incluir desenvolvedores, analista de qualidade, gerente de projeto e outras partes interessadas na construção do desenvolvimento de um sistema. Ele proporciona que as empresas possam criar e melhorar produtos de forma mais eficiente e eficaz do que outras ferramentas que possuem abordagem tradicional do ciclo de vida de desenvolvimento de software, pois a mesma trabalha numa perspectiva ágil. Esta ferramenta pode ser utilizada em nuvem, usando o Azure DevOps Services, ou de forma local, utilizando o Azure DevOps Server. O Azure DevOps são recursos de forma integrada no qual é possível acessar utilizando um browser ou um cliente IDE. É possível utilizar todos os serviços que integram a ferramenta ou a organização pode optar por escolher algo específico que ela precise. São exemplos de serviços fornecidos pelo Azure DevOps os seguintes: Azure Boards, que oferece serviços

Voltar ao sumário

utilizando para suportar o planejamento e o acompanhamento, os defeitos de código e os problemas usando o método Kankan e Scrum; Azure Repos estabelece repositórios como GIT ou Controle de Versão do Time Fundation para controlar o código-fonte do código; Azure Pipelines, que dispõe de serviços relacionados à build e versão para suportar a integração contínua e a entrega dos aplicativos; Azure Test Plans permite múltiplas funcionalidades para testar os sistemas, incluindo testes manuais, exploratórios e testes contínuos e, por fim, há o Azure Artifacts, que possibilita as equipes compartilharem informações de pacote como Maven, npm, NuGet, outras fontes públicas e privadas, que possam integrar o trabalho colaborativo em pacotes de suas pipelines.

- **HipTest** – esta ferramenta tem forte viés para automação de teste. É possível criar cenários editando-os de forma prática e rápida, e os passos podem ser reutilizados dentro da ferramenta com Actionwords e, a partir disso, é possível colocar esforço de sustentação do teste em apenas um lugar. Além disso, a ferramenta proporciona o teste de refatoração, que objetiva manter o negócio com facilidade. As execuções de teste podem ser feitas de forma manual ou automática, de forma que possa acompanhar os resultados de forma simples. É possível integrar também outras ferramentas com o HipTest, sendo alguns exemplos Jenkins, Bamboo, TravisCI e Shippable, promovendo a integração de pipeline CI/CD. Isso proporciona gerir de forma descomplicada a documentação do sistema. Dentro da ferramenta, é possível criar uma história de recurso para que possa ser acompanhada, além de estabelecer um resumo do produto através de snapshot, de forma que também contemple as novas atualizações e possíveis recursos que foram excluídos.
- **Tarantula** – é um sistema para realizar gerenciamento de teste de forma contínua seguindo os procedimentos dos projetos de metodologias ágeis. O profissional de teste consegue criar de forma rápida um plano de teste e realizar o gerenciamento dos casos de teste utilizando tags inteligentes. Ele aborda as diferentes metodologias e tipos de testes e garante acesso a uma interface com usabilidade adequada aos usuários. É possível extrair relatórios da ferramenta que podem proporcionar aos envolvidos tomarem decisões referentes ao projeto, pois conseguem visualizar o status das atividades de teste. Além disso, é uma ferramenta de código aberto e totalmente gratuito.
- **Kiwi TCMS** – esta é mais uma opção de ferramenta que pode ser utilizada no gerenciamento de teste e possui algumas funcionalidades semelhantes à anterior, dentre as quais o objetivo de realizar o monitoramento do Progresso de teste. Esta é ainda uma ferramenta que pode ser muito utilizada em equipes que utilizam metodologias ágeis com o intuito de ser produtivas e consolidar entregas de software com alta qualidade.
- **Accela test** – é uma ferramenta que pode ser utilizada tanto para gerenciar testes manuais como também automatizados. Portanto, ela dá suporte à escrita de casos de testes de forma manual, e é possível escrever scripts de teste dentro da ferramenta. É um diferencial dessa ferramenta unir as duas práticas.
- **Qase** – Qase é uma ferramenta de gerenciamento de caso de teste baseada em nuvem e de código aberto gratuito (com os principais recursos disponíveis). Existem também versões que precisam ser pagas, as quais são conhecidas como Business & Enterprise, que disponibilizam algumas funcionalidades a mais, como, por exemplo, Single Sign-On, mais armazenamento de arquivos/anexos etc. Dentro

da ferramenta existem os Test Suites, que proporcionam arranjar os casos de testes em alguns grupos lógicos. É possível atribuir informações sobre o caso de teste, como gravidade e prioridade, pré-condições, pós-condições, que são o conjunto de passos a serem executados. O Smart Wizard permite a criação e a verificação dos planos de testes, além de permitir ver log de erros, taxa de sucesso, tempo de execução de cada caso etc. Tem compartilhamento simultâneo de informações e o acesso é baseado em permissões.

- **Quality Center – HP Quality Center (QC)** – atualmente é conhecido como ALM/ Quality Center, que é desenvolvido pela empresa Micro Focus. A ferramenta funciona como um painel de controle de forma unificada para gerenciar a qualidade do software. Ela dá apoio ao analista de qualidade e demais profissionais envolvidos no desenvolvimento do software para controlar o gerenciamento do ciclo de vida dos sistemas e consegue suportar processos rígidos e auditáveis do ciclo de vida. É possível acompanhar e reunir todos os esforços de teste, promovendo a governança e a visualização dos processos de teste, através do gerenciamento do planejamento de testes, além de ser possível demonstrar as execuções de testes, cadastrar e rastrear defeitos, atribuindo responsabilidades e monitorando a qualidade de forma geral. Essa ferramenta utiliza uma perspectiva baseada em riscos, e é orientada a requisitos, podendo tratar as ideias mais complexas desses temas. Ela também suporta a criação de testes automatizados coletando os resultados e gerenciando os ativos reutilizáveis. Dentro dessa ferramenta, é possível fazer de forma minuciosa a rastreabilidade de ponta a ponta, acompanhando as relações entre requisitos, defeitos e testes durante todo o ciclo de vida do sistema. É possível utilizar as análises de requisitos e cobertura dos testes para verificar se todos os requisitos foram atendidos. Além disso, dentro dessa mesma visão, a ferramenta gera uma matriz de rastreabilidade para que se possa focar no que está sendo testado. É possível, ainda, obter uma visão holística das versões para que se possam tomar decisões bem embasadas. Com ela podemos analisar profundamente projetos individuais para coletar *insigths* novos sobre o sistema. Os gráficos gerados nesta ferramenta possuem atualização real time, e estão prontos para serem utilizados e compartilhados com as demais partes interessadas. Estes gráficos podem ser estruturados a partir de ferramentas avançadas de identificação das correlações, tendências e anomalias encontradas.
- **TestMaster** – permite a configuração de um sistema de teste no local ou bancada de teste para o engenheiro de teste. Módulos de função TestMaster versáteis e interfaces de programação estão disponíveis para a integração econômica de sistemas de testes simples até bancos de testes complexos com componentes HIL em uma rede de computadores.
- **XQual Studio** – estrutura os lançamentos em componentes, definindo os requisitos do seu negócio, é possível escrever suas especificações funcionais e técnicas, gerenciar projetos usando práticas Agile e modelo em V. Nesta ferramenta, podem-se criar testes e estabelecer ações de caso de teste, bem como realizar as execuções, de forma que seja possível o arquivamento e a comparação dos resultados dos testes para que possam ser extraídos relatórios em relação à cobertura, progresso, qualidade etc. Além disso, nesta ferramenta, é possível registrar e acompanhar os defeitos que possam ser integrados ou de terceiros. Toda união destas funcionalidades estabelece a visibilidade do momento em que o produto poderá ser lançado.

Voltar ao sumário

1.4 – Ferramentas para gestão de defeitos

Além das ferramentas que havíamos mencionado para que se possa realizar a gestão de todos os procedimentos relacionados à etapa de teste, é importante também falar sobre as ferramentas que são direcionadas para acompanhar os defeitos que possam ser encontrados durante essa etapa. Portanto, algumas das ferramentas que podemos citar são:

- **Jira** – muito conhecida mundialmente, esta é uma ferramenta de código aberto utilizada para planejar e acompanhar os projetos utilizando a metodologia ágil. O Jira foi elaborado por uma empresa chamada Atlassian (TRICENTS, 2022). Atualmente a ferramenta disponibiliza vários recursos para realizar o rastreamento de bugs e projetos, contando inclusive com componentes personalizados. O poderoso mecanismo de fluxo de trabalho do Jira garante que os bugs, uma vez capturados, sejam atribuídos e priorizados automaticamente.
- **Mantis** – é outra ferramenta de código aberto muito utilizada para rastrear defeitos em código. O intuito é fornecer uma interface amigável que possa combinar funcionalidade simples e poderosa. É possível que os usuários desenvolvam as habilidades necessárias para manusear a ferramenta em questão de minutos (AGBON, 2017). Ela foi construída em PHP e suporta Linux, Windows e macOS no lado do servidor. É compatível com Chrome, Firefox, Safari, Opera e IE 10+ e foi lançada sob os termos da GNU General Public License (GPL).
- **Bugzilla** – também utilizada para rastrear bugs, ela permite que os desenvolvedores e analistas de testes possam rastrear todos os defeitos que não foram resolvidos. É possível conectá-la com outras ferramentas como JIRA, QC ou ALM etc. Foi codificada em Perl e roda em servidor MySQL.
- **BugNet** – O BugNET oferece a oportunidade de aproveitar os benefícios das plataformas SQL Server, ASP.NET Framework e Microsoft Server mais recentes. É utilizado para rastrear defeitos e gerenciar projetos. Foi escrito em C# e ASP.NET Ele permite registrar e gerenciar erros sem esforço. A ferramenta permite inserir prazos para um bug encontrado; anexar diferentes arquivos; fornecer comentários e descrições através do editor de texto HTML; comunicar problema de link; realizar a exportação de erros para Excel ou RSS; atribuir de forma personalizada tipo, status, prioridade e algum outro atributo relacionado ao defeito registrado e verificar todo o fluxo de trabalho dentro do projeto (TESTMATICK, 2022a).
- **TRAC** – esta também é uma ferramenta de código aberto e que foi projetada para rastrear defeitos e gerenciar projetos pela Edgewall Software. Foi codificada em pyton e possui suporte para bancos de dados como SQLite, PostgreSQL, MySQL e MariaDB. O TRAC dispõe de uma interface web amigável e minimalista, que possui como essência a tecnologia wiki.
- **Redmine** – é um sistema desenvolvido em Ruby on Rrails para gerenciar projeto, possuindo código aberto. É indicado para rastrear bug e realizar a execução simultânea de diversos projetos. Entre as principais funções do Redmine estão: controle de acesso baseado em função flexível; notificações sobre mudanças no projeto por meio de RSS e e-mail; rastreamento de tempo; campos opcionais personalizáveis para vários problemas; interface multilíngue; capacidade de registrar defeitos e adicionar novos usuários automaticamente e suporte de vários bancos de dados populares (TESTMATICK, 2022b).

Voltar ao sumário

Como vimos durante este Percurso de Aprendizagem, os testes automatizados são suportados por ferramentas específicas para executar um conjunto de teste de software, de uma forma que **não haja intervenções humanas** durante a execução dos procedimentos de teste. O objetivo principal dessas ferramentas é executar teste, demonstrar resultados e realizar comparações com testes anteriores para contribuir na obtenção da qualidade de software, reduzindo os riscos de falha durante o ciclo de desenvolvimento e operação do sistema. Um ponto interessante sobre os testes automatizados é que eles **podem ser realizados de forma repetida e a qualquer hora do dia**, para oferecer a confiança adequada na liberação dos sistemas para os ambientes posteriores.

IMPORTANTE

É importante que as ferramentas utilizadas para automatizar testes do tipo funcional ou não funcional possam ser integradas às ferramentas para gerenciamento de teste, com o intuito de controlar e mapear todos os casos de testes que foram executados durante as releases existentes durante o desenvolvimento do sistema.

ATENÇÃO

De modo geral, é importante ressaltar que **não há uma ferramenta que seja mais indicada do que a outra**, todas as ferramentas abordadas nesta disciplina são sugestões as quais podem ser utilizadas para aprimorar os processos de gestão de teste. Deste modo, é de demasiada sensatez pontuar que podem existir outras ferramentas que possam ser mais aplicáveis para cada organização e sua cultura.

Voltar ao sumário

REFERÊNCIAS

AGBON, Jerry. **THIS WEEK'S OPEN SOURCE APPLICATION IS MANTISBT**. [*S. l.*]: Alt Choice Tech Advisers, 1 mar. 2017. Disponível em: https://altchoicetech.com/newsletters/this-weeks-open-source-application-is-mantisbt/. Acesso em: 24 jan. 2023.

BRITO, Douglas. **Testes de performance: O que são e como utilizamos.** Vortxlab, 23 ago. 2021. Disponível em: https://lab.vortx.com.br/testes-de-performance-o-que-sao-e-como-utilizamos. Acesso em: 22 jan. 2023.

GONÇALVEZ, Priscila de F.; BARRETO, Jeanine dos S.; ZENKER, Aline M.; *et al*. **Testes de software e gerência de configuração**. 2019. E-book. ISBN 9788595029361. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595029361. Acesso em: 16 jan. 2023.

KARDASHOV, Mark. **KDT (keyword driven testing) Vs. Test Automation Scripts**. [*S. l.*], 15 set. 2015. Disponível em: https://blog.testproject.io/2015/09/15/kdt-keywords-driven-testing-vs-test-automation-scripts. Acesso em: 23 jan. 2023.

LINKCONSULTING (Brasil, São Paulo, SP). **Automação de testes: Por quê, o quê e como?** São Paulo, SP, 2022. Disponível em: https://linkconsulting.com/sqa/automacao-de-testes-porque-o-que-e-como. Acesso em: 19 jan. 2023.

ISTQB, International Software Testing Qualifications Board. **Certified Tester Syllabus**: **Test Automation Engineer.** 1.0. Brasil: WG-Traduções do BSTQB, 21 out. 2016. Disponível em: https://bcr.bstqb.org.br/docs/syllabus_ct_tae_1.0br.pdf. Acesso em: 18 jan. 2023.

JETBRAINS. **Linguagens específicas de domínio**. Praga, República Tcheca, 2000. Disponível em: https://www.jetbrains.com/pt-br/mps/concepts/domain-specific-languages. Acesso em: 23 jan. 2023.

OLIVEIRA, Saymon C. A. **Data-Driven Testing (DDT) e o reaproveitamento dos testes automatizados**. Medium, 15 set. 2020. Disponível em: https://medium.com/@saymowan/data-driven-testing-ddt-e-o-reaproveitamento-dos-testes-automatizados-8c8d67cc211c. Acesso em: 15 jan. 2023.

TESTMATICK (New York, NY). **BUGNET**. New York, NY, 2022a. Disponível em: https://altchoicetech.com/newsletters/this-weeks-open-source-application-is-mantisbt. Acesso em: 24 jan. 2023.

TESTMATICK (New York, NY). **BUGNET**. New York, NY, 2022b. Disponível em: https://testmatick.com/testing-tools/redmine. Acesso em: 24 jan. 2023.

TRICENTS. **Jira testing tool**. [*S. l.*], 2022. Disponível em: https://www.tricentis.com/learn/jira-testing-tool. Acesso em: 24 jan. 2023.

UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO

UNIVERSIDADE DE FORTALEZA (UNIFOR)

Presidência
Lenise Queiroz Rocha

Vice-Presidência
Manoela Queiroz Bacelar

Reitoria
Fátima Maria Fernandes Veras

Vice-Reitoria de Ensino de Graduação e Pós-Graduação
Maria Clara Cavalcante Bugarim

Vice-Reitoria de Pesquisa
José Milton de Sousa Filho

Vice-Reitoria de Extensão
Randal Martins Pompeu

Vice-Reitoria de Administração
José Maria Gondim Felismino Júnior

Diretoria de Comunicação e Marketing
Ana Leopoldina M. Quezado V. Vale

Diretoria de Planejamento
Marcelo Nogueira Magalhães

Diretoria de Tecnologia
José Eurico de Vasconcelos Filho

Diretoria do Centro de Ciências da Comunicação e Gestão
Danielle Batista Coimbra

Diretoria do Centro de Ciências da Saúde
Lia Maria Brasil de Souza Barroso

Diretoria do Centro de Ciências Jurídicas
Katherinne de Macêdo Maciel Mihaliuc

Diretoria do Centro de Ciências Tecnológicas
Jackson Sávio de Vasconcelos Silva

AUTORA
ANA KLYSSIA MARTINS VASCONCELOS

Mestra em Sistemas de Informação pela Universidade de São Paulo. Especialista em UX (Experiência do Usuário) pela Universidade Anhembi Morumbi. Bacharela em Sistemas de Informação pela Universidade Federal do Ceará. Possui experiência na área de Sistemas de Informação com ênfase em Engenharia de Software, Gestão de Projetos, Governança de TI e Qualidade de Software. Atuou como professora de graduação e pós-graduação na instituição de ensino superior Anhembi Morumbi. Atualmente é analista de qualidade de software na Wipro Brasil, com atuação na M. Dias Branco, e atua como professora desta instituição.

RESPONSABILIDADE TÉCNICA

UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO
VRE
Vice-Reitoria de Ensino de Graduação e Pós-Graduação

UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO
EAD
Educação a Distância

COORDENAÇÃO DA EDUCAÇÃO A DISTÂNCIA

Coordenação Geral
Andrea Chagas Alves de Almeida

Supervisão de Planejamento EAD
Ana Flávia Beviláqua Melo

Supervisão de Recursos EAD
Francisco Weslley Lima

Analista Educacional - Pedagógico
Lara Meneses Saldanha Nepomuceno

Analista Educacional - Mídias
Emanoel Alves Cavalcante

Projeto Instrucional
Ana Lucia Do Nascimento
Maria Mirislene Vasconcelos

Revisão Gramatical
Janaína de Mesquita Bezerra
José Ferreira Silva Bastos

Identidade Visual / Arte
Francisco Cristiano Lopes de Sousa
Paulo Renato Mendes Almeida

Editoração / Diagramação
Emanoel Alves Cavalcante
Rafael Oliveira de Souza
Rebeka Melo Peres
Régis da Silva Pereira

Produção de Áudio e Vídeo
José Moreira de Sousa
Pedro Henrique de Moura Mendes

Programação / Implementação
Renan Alves Diniz
Thais Rozas Teixeira