Certified Tester Advanced Level

Test Automation Enginnering Syllabus

v2.0

International Software Testing Qualifications Board

Direitos autorais

Histórico de revisões

Versão	Data	Observações
2016	21/10/2016	Liberação do CTAL-TAE GA
v2.0	03/05/2024	Versão GA do CTAL-TAE v2.0

Histórico de Revisão da versão na Língua Portuguesa

Versão	Data	Observações
RC	14/06/2024	Tradução para a língua portuguesa
1	29/07/2024	Lançamento da versão na Língua Portuguesa
2	06/01/2025	Adequação ao novo layout do ISTQB

Índice

Agradecimentos

Este documento foi formalmente divulgado pela Assembleia Geral do ISTQB® em 03 de maio de 2024.

Ele foi produzido pela Test Automation Task Force de Testes do Specialist Working Group do International Software Testing Qualifications Board: Graham Bath (Presidente do Specialist Working Group), Andrew Pollner (Vice-Presidente do Specialist Working Group e Presidente da Test Automation Task Force), Péter Földházi, Patrick Quilter, Gergely Ágnecz, László Szikszai. Os revisores da Test Automation Task Force incluíram: Armin Beer, Armin Born, Geza Bujdoso, Renzo Cerquozzi, Jan Giesen, Arnika Hryszko, Kari Kakkonen, Gary Mogyorodi, Chris van Bael, Carsten Weise, Marc-Florian Wendland.

Revisor técnico: Gary Mogyorodi

As seguintes pessoas participaram da revisão, dos comentários e da votação deste syllabus:

Horváth Ágota, Laura Albert, Remigiusz Bednarczyk, Jürgen Beniermann, Armin Born, Alessandro Collino, Nicola De Rosa, Wim Decoutere, Ding Guofu, Istvan Forgacs, Elizabeta Fourneret, Sudhish Garg, Jan Giesen, Matthew Gregg, Tobias Horn, Mattijs Kemmink, Hardik Kori, Jayakrishnan Krishnankutty, Ashish Kulkarni, Vincenzo Marrazzo, Marton Matyas, Patricia McQuaid, Rajeev Menon, Ingvar Nordström, Arnd Pehl, Michaël Pilaeten, Daniel Polan, Nishan Portoyan, Meile Posthuma, Adam Roman, Pavel Sharikov, Péter Sótér , Lucjan Stapp, Richard Taylor, Giancarlo Tomasig, Chris Van Bael, Koen Van Belle, Johan Van Berkel, Carsten Weise, Marc-Florian Wendland, Ester Zabar.

ISTQB Working Group Advanced Level Test Automation Engineer (Edição 2016): Andrew Pollner (presidente), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker.

Agradecimentos do BSTQB

O BSTQB® agradece à equipe do **BSTQB WGT** (Grupo de Trabalho de Traduções do BSTQB) pelo empenho em traduzir este material. Atuaram na tradução e revisão: Eduardo Medeiros Rodrigues, George Fialkovitz, Irene Nagase, Osmar Higashi, Paula Oliveira, Rogério Athaide de Almeida, Stênio Viveiros, Thiago Cesar Andrade.

O BSTQB® também agradece aos **ISTQB® Accredited Training Providers** no Brasil que contribuíram na revisão da tradução com sugestões e questionamentos. No momento deste trabalho os seguintes provedores estavam credenciados: Conecteseaqui, Exseed, Iterasys, Keeggo.

O BSTQB® também agradece às empresas brasileiras credenciadas no **ISTQB® Partner Program** que contribuíram na revisão da tradução com sugestões e questionamentos. No momento deste trabalho as seguintes empresas estavam credenciadas: Deal, Deltapoint, Itriad, Keeggo, Venturus.

0 Introdução

0.1 Objetivo deste syllabus

Este syllabus forma a base para a Qualificação Internacional de Teste de Software para a certificação de nível avançado em Test Automation Engineering (CTAL-TAE). O ISTQB® fornece esse syllabus da seguinte forma:

1. Aos Conselhos Membros, para traduzir para seu idioma local e credenciar os Provedores de Treinamento. Os Conselhos Membros podem adaptar o syllabus às suas necessidades específicas de idioma e modificar as referências para adaptá-las às suas publicações locais.
2. Aos Órgãos de Certificação, para que elaborem questões de exame em seu idioma local, adaptadas aos objetivos de aprendizagem deste syllabus.
3. Aos Provedores de Treinamento, para produzir material didático e determinar métodos de ensino adequados.
4. Para os Candidatos à certificação, para se preparar para o exame de certificação (como parte de um curso de treinamento ou de forma independente).
5. Para a Comunidade Internacional de Engenharia de Software e Sistemas, para promover a profissão de teste de software e sistemas, e como base para livros e artigos.

0.2 O Test Automation Engineering em Teste de Software

A certificação de nível avançado em Test Automation Engineering destina-se a qualquer pessoa envolvida em testes de software e automação de teste. Isso inclui pessoas em funções como testadores, analistas de teste, engenheiros de automação de teste, consultores de teste, arquitetos de teste, gerentes de teste e desenvolvedores de software. Essa certificação de nível avançado também é adequada para qualquer pessoa que queira ter uma compreensão básica da automação de teste, como gerentes de projeto, gerentes de qualidade, gerentes de desenvolvimento de software, analistas de negócios, diretores de TI e consultores de gerenciamento.

O syllabus de Test Automation Engineering (CTAL-TAE) é voltado para o Engenheiro de Teste que deseja implementar ou aprimorar a automação de teste. Ele define métodos e práticas que podem dar suporte a uma solução sustentável.

Outras diretrizes e modelos de referência relacionados à automação de teste As soluções são padrões de engenharia de software para os ciclos de vida de desenvolvimento de software, tecnologias de programação e padrões de formatação selecionados. Este syllabus não ensina engenharia de software. Entretanto, espera-se que um engenheiro de automação de teste deve ter habilidades, experiência e especialização em engenharia de software.

Além disso, um Engenheiro de Automação de Teste precisa estar ciente dos padrões de programação e documentação, e das práticas recomendadas do setor para usá-los ao desenvolver uma solução de automação de teste. Essas práticas podem aumentar a capacidade de manutenção, a confiabilidade e a segurança da solução de automação de teste. Esses padrões geralmente se baseiam em características de qualidade.

0.3 Carreira para Testadores e Engenheiros de Automação de Teste

O esquema ISTQB® fornece suporte para profissionais de teste em todos os estágios de suas carreiras, oferecendo amplitude e profundidade de conhecimento. Os indivíduos que obtiverem a certificação de nível avançado ISTQB® Test Automation Engineering também podem se interessar pela certificação Test Automation Strategy (CT-TAS).

As pessoas que obtiverem a certificação de nível avançado ISTQB® Certified Tester Test Automation Engineering também podem se interessar pelos níveis Core Advanced (Test Analyst, Technical Test Analyst, e Test Management) e, posteriormente, pelo nível Expert (Test Management ou Improving the Test Process). Quem quiser desenvolver habilidades em práticas de teste em uma área de ambiente ágil pode considerar as certificações Agile Technical Tester ou Agile Test Leadership at Scale. O fluxo de Especialista oferece um mergulho profundo em áreas que têm abordagens e atividades de teste específicas, por exemplo, em Test

Automation Strategy, Performance Testing, Security Testing, AI Testing, e Mobile Application Testing, ou onde o conhecimento específico do domínio é necessário (p. ex., Automotive Software Testing ou Game Testing). Acesse www.istqb.org para obter as informações mais recentes sobre o esquema ISTQB® Certified Tester.

0.4 Resultados de Negócio

Esta seção lista os resultados de negócio esperados de um candidato que tenha obtido a certificação de nível avançado Test Automation Engineering.

Um testador certificado em Test Automation Engineering pode:

TAE-B01	Descrever a finalidade da automação de teste
TAE-B02	Entender a automação de teste através do ciclo de vida de desenvolvimento de software
TAE-B03	Entender a configuração de uma infraestrutura para permitir a automação de teste
TAE-B04	Aprender o processo de avaliação para selecionar as ferramentas e estratégias corretas
TAE-B05	Compreender os conceitos de design para criar soluções de automação de teste modulares e dimensionáveis
TAE-B06	Selecionar uma abordagem, incluindo um piloto, para planejar a automação de teste no ciclo de vida de desenvolvimento de software
TAE-B07	Modelar e desenvolver soluções de automação de teste (novas ou modificadas) que atendam às necessidades técnicas
TAE-B08	Considerar o escopo e a abordagem da automação de teste e manutenção do material de teste
TAE-B09	Entender como os testes automatizados se integram aos pipelines de CI/CD
TAE-B10	Entender como coletar, analisar e gerar relatórios sobre os dados de automação de teste para informar aos stakeholders
TAE-B11	Verificar a infraestrutura de automação de teste
TAE-B12	Definir oportunidades de melhoria contínua para automação de teste

0.5 Objetivos de Aprendizagem e Nível Cognitivo de Conhecimento

Os objetivos de aprendizagem apoiam os resultados de negócio e são usados para criar os exames Certified Tester, Test Automation Engineering.

Em geral, todo o conteúdo deste syllabus pode ser examinado nos níveis K2, K3 e K4, exceto a Introdução e os Apêndices. Ou seja, o candidato pode ser solicitado a reconhecer, lembrar ou recordar uma palavra-chave ou conceito mencionado em qualquer um dos oito capítulos. Os níveis dos objetivos específicos de aprendizado são mostrados no início de cada capítulo e classificados da seguinte forma:

- K2: Compreender
- K3: Aplicar
- K4: Analisar

Mais detalhes e exemplos de objetivos de aprendizagem são fornecidos no Apêndice A.

Todos os termos listados como palavras-chave logo abaixo dos títulos dos capítulos devem ser lembrados, mesmo que não sejam explicitamente mencionados nos objetivos de aprendizagem.

0.6 Exame de Certificação Test Automation Engineering

O exame de certificação de nível avançado em Test Automation Engineering será baseado neste syllabus. As respostas às perguntas do exame podem exigir o uso de material baseado em mais de uma seção deste syllabus. Todas as seções do syllabus são passíveis de exame, exceto a Introdução e os Apêndices. Normas e livros são incluídos como referências, mas seu conteúdo não é passível de exame, além do que está resumido no próprio syllabus a partir dessas normas e livros.

Consulte o documento *Exam Structures and Rules v1.1 Compatible with Syllabus Foundation and Advanced Levels and Specialists Modules* para obter mais detalhes sobre o exame *Test Automation Engineering*.

O critério de entrada para fazer o exame de certificação Test Automation Engineering é que os candidatos tenham interesse em testes de software e automação de teste. Entretanto, é altamente recomendável que os candidatos também tenham:

- Pelo menos um histórico mínimo em desenvolvimento e teste de software, como seis meses de experiência como Engenheiro de Teste de Software ou como Desenvolvedor de Software.
- Fazer um curso que tenha sido credenciado de acordo com os padrões do ISTQB® (por um de seus Conselhos Membros).

Observação sobre requisitos de entrada: o certificado ISTQB® Foundation Level deve ser obtido antes de fazer o exame de certificação ISTQB® Test Automation Engineering.

0.7 Credenciamento

Um Conselho Membro do ISTQB® pode credenciar Provedores de Treinamento cujo material do curso siga este syllabus. Os Provedores de Treinamento devem obter as diretrizes de credenciamento do Conselho Membro ou do órgão que realiza o credenciamento. Um curso credenciado é reconhecido como estando em conformidade com este syllabus e pode ter um exame ISTQB® como parte do curso.

As diretrizes de credenciamento para este syllabus seguem as diretrizes *Accreditation Guidelines* publicadas pelo *Processes Management and Compliance Working Group*.

0.8 Manuseio de Normas

Há padrões referenciados no syllabus Test Automation Engineering (p. ex., IEEE e ISO). O objetivo dessas referências é fornecer uma estrutura (como nas referências à ISO 25010 com relação às características de qualidade) ou fornecer uma fonte de informações adicionais, se o leitor desejar. Observe que o syllabus usa os documentos normativos como referência. Os documentos normativos não se destinam a exame. Consulte o Capítulo 9 Referências para obter mais informações sobre as normas.

0.9 Mantendo-se atualizado

O setor de software muda rapidamente. Para lidar com essas mudanças e fornecer aos stakeholders acesso a informações relevantes e atuais, os grupos de trabalho do ISTQB® criaram links no site www.istqb.org, que se referem a documentos de apoio e a mudanças nos padrões. Essas informações não são passíveis de exame no syllabus Test Automation Engineering.

0.10 Nível de detalhe

O nível de detalhamento desse syllabus permite cursos e exames consistentes em nível internacional. Para atingir esse objetivo, o syllabus consiste em:

- Objetivos gerais de instrução que descrevem a intenção do Test Automation Engineering;
- Uma lista de termos que os alunos devem ser capazes de lembrar;
- Objetivos de aprendizagem para cada área de conhecimento, descrevendo o resultado cognitivo de aprendizado a ser alcançado;
- Uma descrição dos principais conceitos, incluindo referências a fontes, como literatura ou padrões aceitos.

O conteúdo do syllabus não é uma descrição de toda a área de conhecimento de testes de software; ele reflete o nível de detalhes a ser abordado nos cursos de treinamento Test Automation Engineering. Ele se concentra nos conceitos e nas técnicas de teste que podem ser aplicados a todos os projetos de software, incluindo aqueles que seguem os métodos Ágeis. Este syllabus não contém nenhum objetivo de aprendizagem específico relacionado a testes ágeis, mas discute como esses conceitos se aplicam a projetos ágeis e a outros tipos de projetos.

0.11 Como este syllabus está organizado

Há oito capítulos com conteúdo passível de exame. O título de nível superior de cada capítulo especifica o tempo para o capítulo; o tempo não é fornecido para subcapítulos. Para cursos de treinamento credenciados, o syllabus exige um mínimo de 21 horas de instrução, distribuídas pelos oito capítulos da seguinte forma:

- Capítulo 1: 45min - Introdução e objetivos da automação de teste.
  - O testador aprende sobre os benefícios da automação de teste e suas limitações;
  - A automação de teste em diferentes modelos de ciclo de vida de desenvolvimento de software é abordada;
  - O testador aprende como um sistema em teste (SUT) afeta a adequação das ferramentas de teste.
- Capítulo 2: 180min – Preparando-se para a automação de teste.
  - Projeto para testabilidade do SUT por meio da observação, controle e uma arquitetura claramente definida;
  - Um testador aprende sobre automação de teste em diferentes ambientes;
  - Fatores necessários para avaliar uma solução apropriada de automação de teste de teste são abordados;
  - Um testador aprenderá sobre as considerações técnicas necessárias para desenvolver recomendações sobre automação de teste.
- Capítulo 3: 210min - Arquitetura de automação de teste.
  - Arquitetura de automação de teste e seus componentes que levam a uma solução de automação de teste de testes é abordada;
  - Um testador aprenderá sobre camadas e sua aplicação em uma estrutura de automação de teste de testes;
  - Várias abordagens para o uso de ferramentas de automação de teste serão abordadas;
  - Um testador aprenderá como os princípios e padrões de design podem ser aplicados à automação de teste;
- Capítulo 4: 150min - Implementação da automação de teste.
  - Como planejar e implementar com eficácia um projeto piloto de automação de teste será abordado;
  - Um testador aprenderá sobre os riscos de implementação e estratégias de mitigação;
  - Fatores que melhoram a capacidade de manutenção da automação de teste será abordado.
- Capítulo 5: 90min - Estratégias de implementação e implantação para automação de teste.
  - Um testador aprenderá sobre pipelines de CI/CD e execução de testes automatizados em todos os níveis de teste;
  - Gerenciamento de configuração para componentes de automação de teste será abordado;
  - Um testador aprenderá sobre dependências aplicadas a testes de API e contratos.
- Capítulo 6: 150min - Relatórios e métricas de automação de teste.
  - Um testador aprenderá sobre onde os dados podem ser coletados de um SUT e sobre a automação de teste para análise e geração de relatórios;
  - Análise de dados de relatórios de SUT e automação de teste para descobrir as causas das falhas serão abordadas;
  - Será abordado o uso de relatórios e painéis de teste para informar os stakeholders.
- Capítulo 7: 135min - Verificação da solução de automação de teste.

  - O testador aprenderá a examinar e verificar a operação correta dos componentes de automação de teste componentes e ambiente;
  - A garantia de que os scripts de teste e os conjuntos de teste sejam executados corretamente será abordada;
  - Um testador entenderá quando realizar a análise da causa raiz;
  - Técnicas para analisar a automação de teste para qualidade serão abordadas.
- Capítulo 8: 210min - Melhoria contínua.
  - Serão abordadas áreas adicionais de análise de dados para aprimoramento de casos de teste;
  - Um testador aprenderá maneiras de fazer melhorias e atualizações em uma solução de automação de teste de testes e seus componentes;
  - Identificação de maneiras de consolidar e otimizar a automação de teste será abordado;
  - Um testador aprenderá como as ferramentas de automação de teste podem ajudar nas necessidades de suporte e configuração de testes.

1 Introdução e objetivos da Automação de Teste - 45 min

Palavras-chave

sistema em teste, automação de teste, engenheiro de automação de teste

Objetivos de aprendizagem:

1.1 Objetivo da automação de teste

TAE-1.1.1 (K2) Explicar as vantagens e desvantagens da automação de teste.

1.2 Automação de teste no ciclo de vida de desenvolvimento de software

TAE-1.2.1 (K2) Explicar como a automação de teste é aplicada em diferentes modelos de ciclo de vida de desenvolvimento de software

TAE-1.2.2 (K2) Selecionar ferramentas de automação de teste adequadas para um determinado sistema em teste.

1.1 Objetivo da Automação de Teste

1.1.1 Explicar as vantagens e desvantagens da Automação de Teste

A automação de teste, que inclui a execução de testes automatizados e a geração de relatórios de testes, é uma ou mais das seguintes atividades:

- Usar ferramentas de software criadas especificamente para controlar e configurar conjuntos de testes para a execução de testes;
- Execução de testes de forma automatizada;
- Comparação dos resultados reais com os resultados esperados.

A automação de teste oferece recursos e capacidades significativos que podem interagir com um sistema em teste (SUT). A automação de teste pode abranger uma ampla área de software. As soluções abrangem muitos tipos de software (p. ex., SUT com uma interface de usuário (UI), SUT sem uma UI, aplicativos móveis, protocolos de rede e conexões).

A automação de teste possui muitas vantagens:

- Permite a execução de mais testes por compilação em comparação com os testes manuais;
- Oferece a capacidade de criar e executar testes que não podem ser executados manualmente (p. ex., resposta em tempo real, testes remotos e testes paralelos);
- Permite a realização de testes mais complexos do que os testes manuais;
- Executa mais rapidamente do que os testes manuais;
- Está menos sujeito a erros humanos;
- É mais eficaz e eficiente no uso dos recursos de teste;
- Fornece feedback mais rápido sobre a qualidade do SUT;
- Ajuda a melhorar a confiabilidade do sistema (p. ex., disponibilidade e capacidade de recuperação);
- Melhora a consistência da execução do teste em todos os ciclos de teste.

No entanto, a automação de teste tem possíveis desvantagens, incluindo:

- Custos adicionais para o projeto, pois pode ser necessário contratar um Engenheiro de Automação de teste (TAE), comprar um novo hardware e organizar o treinamento;
- Exigência de investimento inicial para configurar uma solução de automação de teste;
- Tempo para desenvolver e manter uma solução de automação de teste;
- Requisitos de automação de teste claros para garantir o sucesso;
- Rigidez dos testes e menor adaptabilidade às mudanças no SUT;
- Introdução de defeitos adicionais por meio da automação de teste.

Há limitações para a automação de teste que precisam ser levadas em conta:

- Nem todos os testes manuais podem ser automatizados;
- Verificar apenas o que os testes automatizados estão programados para fazer;
- A automação de teste só pode verificar resultados de testes interpretáveis por máquinas, o que significa que algumas características de qualidade podem não ser testáveis com a automação;
- A automação de teste só pode verificar os resultados dos testes que podem ser verificados por um oráculo de teste automatizado.

1.2 Automação de Teste no Ciclo de Vida de Desenvolvimento de Software

1.2.1 Explicar como a automação de teste é aplicada em diferentes modelos de ciclo de vida de desenvolvimento de software

Cascata (Waterfall)

O modelo em cascata é um modelo de SDLC que é linear e sequencial. Esse modelo tem fases distintas (ou seja, requisitos, modelagem, implementação, verificação e manutenção) e cada fase geralmente termina com uma documentação que deve ser aprovada. A implementação da automação de teste normalmente ocorre em paralelo ou após a fase de implementação. As execuções de teste geralmente ocorrem durante a fase de verificação, pois os componentes do software não estão prontos para o teste.

Modelo V

O modelo V é um modelo de SDLC em que um processo é executado de forma sequencial. Como um projeto é definido a partir de requisitos de alto nível para requisitos de baixo nível, as atividades de teste e integração correspondentes são definidas para validar esses requisitos. É daí que derivam os níveis de teste tradicionais: componente, integração de componentes, sistema, integração de sistemas e aceite, conforme descrito na Seção 2.2 do CTFL. Para cada nível de teste é possível e recomendado o fornecimento de uma estrutura de automação de teste (TAF, Test Automation Framework).

Desenvolvimento Ágil de Software

No desenvolvimento ágil de software, há inúmeras possibilidades de automação de teste. Ao contrário do modelo em cascata ou em V, no método de Desenvolvimento Ágil de Software, os TAE e os representantes de negócio podem decidir sobre o roteiro, o cronograma e a entrega planejada dos testes. Nesse método, há práticas recomendadas, como revisões de código, programação em pares e execução frequente de testes automatizados. A eliminação de áreas (ou seja, garantir que desenvolvedores, testadores e outros stakeholders trabalhem juntos) permite que as equipes cubram todos os níveis de teste com a quantidade e a profundidade adequadas de automação de teste, atingindo uma meta chamada automação in-sprint. Mais detalhes podem ser encontrados no ISTQB CT-TAS Syllabus, Seção 3.2.

1.2.2 Selecionar ferramentas de automação de teste adequadas para um determinado sistema em teste

Para identificar as ferramentas de teste mais adequadas para um determinado projeto, o SUT deve ser analisado primeiro. Os TAE precisam identificar os requisitos do projeto que podem ser usados como baseline para a seleção da ferramenta.

Como diferentes recursos de ferramentas de automação de teste são usados para software de interface do usuário e, por exemplo, serviços da Web, é importante entender o que o projeto deseja alcançar ao longo do tempo. Não há limite para o número de ferramentas e recursos de automação de teste que podem ser usados ou selecionados, mas os custos sempre devem ser considerados. Usar uma ferramenta comercial pronta para uso ou implementar uma solução personalizada baseada em tecnologia de código aberto pode ser um processo complexo.

O próximo tópico a ser avaliado é a composição e a experiência da equipe em automação de teste. No caso dos testadores terem pouca ou nenhuma experiência em programação, o uso de uma solução com pouco ou nenhum código pode ser uma opção viável.

Para testadores técnicos com conhecimento de programação, pode ser útil selecionar ferramentas cuja linguagem corresponda à do SUT. Isso oferece vantagens, incluindo a capacidade de trabalhar com os desenvolvedores na depuração de defeitos de automação de teste de forma mais eficiente e treinamento cruzado de membros entre equipes.

2 Preparando-se para a Automação de Teste – 180 min

Palavras-chave

teste de API, teste de GUI, testabilidade

Objetivos de aprendizagem:

2.1 Compreender a configuração de uma infraestrutura para permitir a automação de teste

TAE-2.1.1 (K2) Descrever as necessidades de configuração de uma infraestrutura que permita a implementação da automação de teste.

TAE-2.1.2 (K2) Explicar como a automação de teste é aproveitada em diferentes ambientes.

2.2 Processo de avaliação para selecionar as ferramentas e estratégias corretas

TAE-2.2.1 (K4) Analisar um sistema em teste para determinar a solução apropriada de automação de teste.

TAE-2.2.2 (K4) Ilustrar os resultados técnicos de uma avaliação de ferramenta.

2.1 Compreender a configuração de uma infraestrutura que permita a automação de teste.

2.1.1 Descrever as necessidades de configuração de uma infraestrutura que permita a implementação da automação de teste.

A testabilidade do SUT (ou seja, a disponibilidade de interfaces de software que suportam testes, por exemplo, para permitir o controle e a observação do SUT) deve ser projetada e implementada paralelamente ao projeto e à implementação dos outros recursos do SUT. Esse trabalho geralmente é realizado por um Arquiteto de Software, pois a testabilidade é um requisito não funcional do sistema, muitas vezes com o envolvimento de um TAE para identificar as áreas específicas em que podem ser feitas melhorias.

Para melhorar a testabilidade do SUT, há diferentes soluções que podem ser utilizadas e que têm diferentes necessidades de configuração, por exemplo:

- Identificadores de acessibilidade.
  - As diferentes estruturas de desenvolvimento podem gerar esses identificadores automaticamente ou os desenvolvedores podem defini-los manualmente.
- Variáveis de ambiente do sistema.
  - Alguns parâmetros do aplicativo podem ser alterados para facilitar os testes por meio da administração.
- Variáveis de implantação.
  - Semelhante às variáveis do sistema, mas pode ser definido antes de iniciar a implantação.

O projeto para testabilidade de um SUT consiste nos seguintes aspectos:

- **Observabilidade**: O SUT precisa oferecer interfaces que forneçam informações sobre o SUT. Os casos de teste podem então usar essas interfaces para determinar se os resultados reais são iguais aos resultados esperados.
- **Controle**: O SUT precisa fornecer interfaces que possam ser usadas para executar ações no SUT. Podem ser elementos de interface do usuário, chamadas de função, elementos de comunicação (p. ex., protocolos TCP/IP (Transmission Control Protocol/Internet Protocol) e USB (Universal Serial Bus)) ou sinais eletrônicos para comutações físicas ou lógicas nas diferentes variáveis de ambiente.
- **Transparência da arquitetura**: A documentação de uma arquitetura precisa fornecer componentes e interfaces claros e compreensíveis que proporcionem a observação e o controle em todos os níveis de teste, promovendo a qualidade.

2.1.2 Explicar como a automação de teste é aproveitada em diferentes ambientes.

Diferentes tipos de testes automatizados podem ser executados em diferentes ambientes. Esses ambientes podem diferir entre projetos e metodologias, e a maioria dos projetos tem um ou mais ambientes a serem utilizados para testes. Do ponto de vista técnico, esses ambientes podem ser criados a partir de contêineres, virtualização de software ou usando outras abordagens.

Um conjunto de possíveis ambientes a serem considerados inclui:

Ambiente de desenvolvimento local

Um ambiente de desenvolvimento local é onde o software é criado inicialmente e os componentes são testados com automação para verificar a adequação funcional. Vários tipos de teste podem ocorrer no ambiente de desenvolvimento local, incluindo testes de componentes, testes de GUI e testes de interface de programação de aplicativos (API). Também é importante observar que, com o uso de um ambiente de desenvolvimento integrado (IDE) em um determinado computador, é possível realizar testes caixa-branca para identificar problemas de codificação e de qualidade ruins o mais cedo possível.

Criar ambiente

Seu principal objetivo é criar o software e executar testes que verifiquem a exatidão da criação resultante em um ecossistema de DevOps. Esse ambiente pode ser um ambiente de desenvolvimento local ou um agente de integração contínua/entrega contínua (CI/CD) em que os testes de baixo nível (ou seja, testes de componentes e testes de integração de componentes) e a análise estática podem ser realizados sem a implantação real em outros ambientes.

Ambiente de integração

Depois de realizar testes de baixo nível e análise estática o próximo estágio é um ambiente de integração do sistema. Aqui, há um candidato à versão do SUT totalmente integrado a outros sistemas que podem ser testados. Nesse ambiente, é possível executar um conjunto de testes totalmente automatizado, seja de testes de interface do usuário ou de API. Nesse ambiente, não há teste caixa-branca, apenas teste caixa-preta (ou seja, integração do sistema e/ou teste de aceite). É importante observar que esse é o primeiro ambiente em que o monitoramento deve estar presente para ver o que acontece em segundo plano durante o uso do SUT para permitir a investigação eficiente de defeitos/falhas.

Ambiente de pré-produção

Um ambiente de pré-produção é usado principalmente para avaliar características de qualidade não funcionais (p. ex., eficiência de performance). Embora os testes não funcionais possam ser realizados em qualquer ambiente, há um foco extra na pré-produção porque ela se assemelha o máximo possível à produção. Muitas vezes, o teste de aceite do usuário pode ser realizado pelos stakeholders de negócio para verificar o produto final, e é possível executar o conjunto de testes automatizados existentes aqui também, se necessário. Esse ambiente também é monitorado.

Ambiente de produção/operacional

Um ambiente de produção pode ser usado para avaliar características de qualidade funcionais e não funcionais em tempo real, enquanto os usuários interagem com um sistema implantado com monitoramento e determinadas práticas recomendadas que permitem testes em produção (p. ex., lançamento canário, implantação azul/verde e testes A/B).

2.2 Processo de avaliação para selecionar as ferramentas e as estratégias corretas.

2.2.1 Analisar um sistema em teste para determinar a solução apropriada de automação de teste.

Cada SUT pode ser diferente de outro, mas há vários fatores e características que podem ser analisados para se ter uma solução de automação de teste (TAS) bem-sucedida. Durante a investigação de um SUT, os TAE precisam reunir requisitos considerando seu escopo e os recursos fornecidos. Diferentes tipos de aplicativos (p. ex., serviço da Web, móvel e Web) precisam de diferentes tipos de automação de teste do ponto de vista técnico. A investigação pode ser feita - e é recomendada - em colaboração com outros stakeholders (p. ex., testadores manuais, stakeholders de negócio e analistas de negócios) para identificar o maior número possível de riscos e suas mitigações, a fim de obter uma solução de automação de teste benéfica para o futuro.

Os requisitos para uma abordagem e arquitetura de automação de teste devem considerar o seguinte:

- Quais atividades do processo de teste devem ser automatizadas (p. ex., gerenciamento de teste, projeto de teste, geração de teste e execução de teste);
- Quais níveis de teste devem ser suportados;
- Quais tipos de teste devem ser suportados;
- Quais funções de teste e conjuntos de habilidades devem ser suportados;
- Quais produtos de software, linhas de produtos e famílias devem ser suportados (p. ex., para definir a extensão e a vida útil do TAS implementado);

- Quais tipos de SUT precisam ser compatíveis com o TAS;
- Disponibilidade de dados de teste e sua qualidade;
- Possíveis métodos e maneiras de emular casos inacessíveis (p. ex., aplicativos de terceiros envolvidos).

2.2.2 Ilustrar os resultados técnicos de uma avaliação de ferramenta

Depois que a SUT for analisada e os requisitos forem coletados de todos os stakeholders, é provável que existam ferramentas de automação de teste que atendam a esses requisitos e que possam ser consideradas. Pode não haver uma única ferramenta que atenda a todos os requisitos identificados, e os stakeholders devem reconhecer essa possibilidade.

É útil capturar as descobertas sobre as possíveis ferramentas e refletir sobre os vários requisitos diretos e indiretos em uma tabela de comparação. O objetivo da tabela de comparação é permitir que os stakeholders vejam as diferenças entre as ferramentas com base em requisitos específicos. A tabela de comparação lista as ferramentas nas colunas e os requisitos nas linhas. As células contêm informações sobre as propriedades de cada ferramenta em relação a cada requisito e sobre as prioridades.

Em geral, as ferramentas de automação de teste devem ser avaliadas para determinar se atendem aos requisitos identificados na seção anterior (2.2.1). Os requisitos a serem considerados ao avaliar e comparar as ferramentas incluem:

- A linguagem/tecnologia da ferramenta e as ferramentas do IDE.
- A capacidade de configurar uma ferramenta, se ela suporta diferentes ambientes de teste, executa configurações e usa valores de configuração dinâmicos ou estáticos.
- A capacidade de gerenciar dados de teste dentro da ferramenta. O gerenciamento de dados de teste pode ser integrado a um repositório central para controle de versão.
- Para diferentes tipos de teste, diferentes ferramentas de automação de podem precisar ser selecionadas.
- A capacidade de fornecer recursos de relatório. Isso é importante para alinhar-se com os requisitos de relatório de teste do projeto.
- A capacidade de se integrar a outras ferramentas usadas em um projeto ou na organização, como CI/CD, rastreamento de tarefas, gerenciamento de testes, relatórios ou outras ferramentas.
- A capacidade de expandir a arquitetura geral de testes e avaliar a escalabilidade, a capacidade de manutenção, a modificabilidade, a compatibilidade e a confiabilidade das ferramentas.

Essa tabela de comparação é uma boa fonte para determinar uma ferramenta proposta ou um conjunto de ferramentas a ser usado para a automação de teste do SUT.

O processo pode variar quanto à forma como a decisão é tomada sobre a(s) ferramenta(s) a ser(em) usada(s), mas a proposta deve ser demonstrada aos stakeholders apropriados para aprovação.

3 Arquitetura de Automação de Teste – 210 min

Palavras-chave

desenvolvimento orientado pelo comportamento, captura/reprodução, teste orientado por dados, arquitetura de automação de teste genéricos, teste orientado por palavra-chave, scripting linear, teste baseado em modelos, scripting estruturado, camada de adaptação de teste, estrutura de automação de teste, solução de automação de teste, estrutura de teste, script de teste, testware, etapa de teste, desenvolvimento orientado por teste

Objetivos de aprendizagem:

3.1 Conceitos de design aproveitados na automação de teste.

TAE-3.1.1 (K2) Explicar os principais recursos em uma arquitetura de automação de teste.

TAE-3.1.2 (K2) Explicar como projetar uma solução de automação de teste.

TAE-3.1.3 (K3) Aplicar camadas de automação de teste de testes.

TAE-3.1.4 (K3) Aplicar diferentes abordagens para automatizar casos de teste.

TAE-3.1.5 (K3) Aplicar princípios e padrões de design na automação de teste.

3.1 Conceitos de design aproveitados na automação de teste

3.1.1 Explicar os principais recursos em uma arquitetura de automação de teste

Arquitetura Genérica de Automação de Teste (gTAA)

A gTAA é um conceito de projeto de alto nível que fornece uma visão abstrata da comunicação entre a automação de teste e os sistemas aos quais a automação de teste está conectada, ou seja, o SUT, o gerenciamento de projetos, o gerenciamento de testes e o gerenciamento de configurações (veja a Figura 1). Ele também fornece os recursos que são necessários para cobrir ao projetar uma arquitetura de automação de teste (TAA).

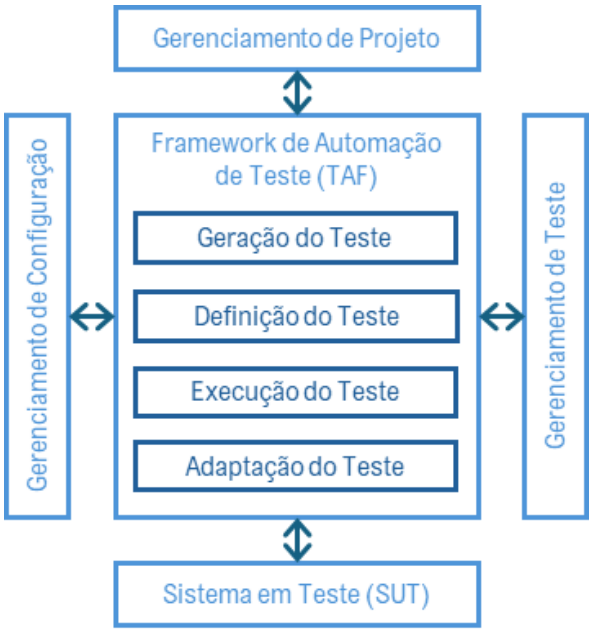

Figura 1 – Diagrama gTAA

As interfaces do gTAA descrevem:

- A interface do SUT descreve a conexão entre a SUT e a TAF (consulte a seção 3.1.3 sobre a estrutura de automação de teste).
- A interface de gerenciamento de projetos descreve o progresso do desenvolvimento.
- A interface de gerenciamento de testes descreve o mapeamento das definições dos casos de teste e dos casos de teste automatizados.
- A interface de gerenciamento de configuração descreve os pipelines de CI/CD, os ambientes e o software de teste.

Recursos fornecidos pelas ferramentas de automação de teste ferramentas e bibliotecas

Os principais recursos de automação de teste devem ser identificados e selecionados entre as ferramentas disponíveis, conforme necessário para um determinado projeto.

Geração do teste: Oferece suporte ao projeto automatizado de casos de teste com base em um modelo de teste. As ferramentas de teste baseadas em modelos podem ser aproveitadas no processo de geração (consulte o ISTQB CT-MBT Syllabus). A geração de testes é um recurso opcional.

Definição do teste: Oferece suporte à definição e à implementação de casos de teste e/ou conjuntos de teste, que, opcionalmente, podem ser derivados de um modelo de teste. Separa a definição de teste do SUT e/ou das ferramentas de teste. Contém os meios para definir testes de alto e baixo nível, que são tratados nos dados de teste, nos casos de teste e nos componentes da biblioteca de testes ou em suas combinações.

Execução do teste: Oferece suporte à execução de testes e ao seu registro. Fornece uma ferramenta de execução de testes para executar os testes selecionados automaticamente e um componente de registro e relatório destes testes.

Adaptação do testes: Fornece a funcionalidade necessária para adaptar os testes automatizados para os vários componentes ou interfaces do SUT. Fornece diferentes adaptadores para conexão com a SUT por meio de APIs, protocolos e serviços.

3.1.2 Explicar como projetar uma solução de automação de teste

Uma Solução de Automação de Teste (TAS) é definida por uma compreensão dos requisitos funcionais, não funcionais e técnicos do SUT e das ferramentas existentes ou necessárias para implementar uma solução. Uma TAS é implementada com ferramentas comerciais ou de código aberto e pode precisar de adaptadores adicionais específicos para a SUT.

O TAA define o projeto técnico para o TAS geral, e deve abordar:

- A seleção de automação de teste e bibliotecas específicas de ferramentas;
- O desenvolvimento de plug-ins e/ou componentes;
- A identificação dos requisitos de conectividade e interface (p. ex., firewalls, banco de dados, URLs/conexões, emuladores/simuladores, filas de mensagens e protocolos);
- A conexão com as ferramentas de gerenciamento de testes e de defeitos;
- A utilização de um sistema de controle de versão e repositórios.

3.1.3 Aplicar camadas de estruturas de automação de teste

Framework de Automação de teste

O TAF é a base de um TAS. Ela geralmente inclui um conjunto de testes, também conhecido como executor de testes, e bibliotecas de teste, scripts de teste e suítes de teste.

Camadas do TAF

As camadas do TAF definem uma borda distinta de classes que têm finalidades semelhantes, como casos de teste, relatórios de teste, registro de teste, criptografia e estruturas de teste. Ao introduzir uma camada para cada finalidade, o projeto pode se tornar complicado. Portanto, é recomendável manter o número de camadas de TAF baixo.

Scripts de teste

Sua finalidade é fornecer um repositório de casos de teste das anotações do SUT e do conjunto de testes. Ele chama os serviços da camada de lógica do negócio que podem envolver as etapas de teste, fluxos de usuário ou chamadas de API. No entanto, não devem ser feitas chamadas diretas para as bibliotecas essenciais a partir de scripts de teste.

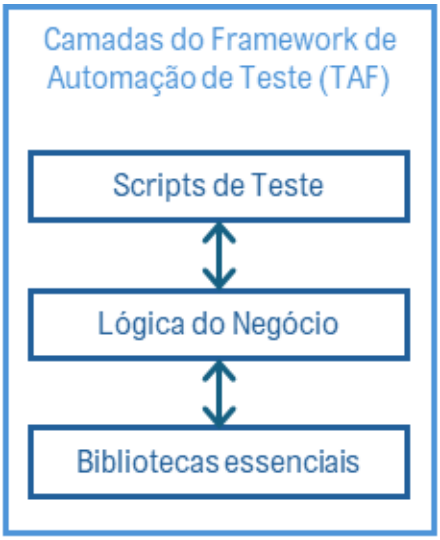

Figura 2 – Camadas do TAF

Lógica do negócio

Todas as bibliotecas dependentes do SUT são armazenadas nessa camada. Essas bibliotecas herdarão os arquivos de classe das bibliotecas principais ou usarão as fachadas fornecidas por elas (consulte a seção 3.1.5 sobre herança e fachadas). A camada de lógica do negócio é usada para configurar a TAF para execução na SUT e nas configurações adicionais.

Bibliotecas essenciais

Todas as bibliotecas que são independentes de qualquer SUT são armazenadas nessa camada. Essas bibliotecas essenciais podem ser reutilizadas em qualquer tipo de projeto que compartilhe a mesma pilha de desenvolvimento.

Dimensionamento da automação de teste

O exemplo a seguir (Figura 3) mostra como as bibliotecas essenciais fornecem uma base reutilizável para várias TAF. No Projeto #1, há dois TAF criados com base nas bibliotecas essenciais, e um projeto separado aproveita as bibliotecas essenciais já existentes para criar seu TAF para testar o Aplicativo #3. Um TAE cria os TAF para o Projeto #1, enquanto um segundo TAE cria o TAF para o Projeto #2.

Figura 3 – Aplicação das Bibliotecas essenciais em múltiplos TAFs

3.1.4 Aplicar diferentes abordagens para automatizar casos de teste

Há várias abordagens de desenvolvimento que as equipes podem escolher para produzir casos de teste automatizados. Elas podem incluir linguagens de script interativas ou linguagens de programação compiladas. As diferentes abordagens oferecem diferentes benefícios de automação e podem ser aproveitadas em diferentes circunstâncias. Embora o desenvolvimento orientado por testes (TDD) e o desenvolvimento orientado por comportamento (BDD) sejam metodologias de desenvolvimento, se seguidas corretamente, elas resultam no desenvolvimento de casos de teste automatizados.

Captura/Reprodução

A captura/reprodução é uma abordagem que captura as interações com o SUT enquanto uma sequência de ações é executada manualmente. Essas ferramentas produzem scripts de teste durante a captura e, dependendo da ferramenta usada, o código de automação de teste pode ser modificável. As ferramentas que não expõem o código às vezes são chamadas de automação de teste sem código, enquanto as ferramentas que expõem o código são chamadas de automação de teste com pouco código.

Prós

- Inicialmente fácil de configurar e usar.

Contras

- Difícil de manter, dimensionar e evoluir.
- O SUT precisa estar disponível durante a captura de um caso de teste.
- Só é viável para um escopo pequeno e uma SUT que raramente muda.
- A execução do SUT capturada depende muito da versão do SUT da qual a captura foi feita.
- O registro de cada caso de teste individual, em vez de reutilizar blocos de construção existentes, consome muito tempo.

Scripting linear

A criação de scripts lineares é uma atividade de programação que não exige bibliotecas de teste personalizadas criadas por um TAE e é usada para escrever e executar os scripts de teste. Um TAE pode aproveitar quaisquer scripts de teste gravados por uma ferramenta de captura/reprodução que podem ser modificados.

Prós

- Fácil de configurar e de começar a escrever scripts de teste.
- Em comparação com a captura/reprodução os scripts de teste podem ser modificados mais facilmente.

Contras

- Difícil de manter, dimensionar e evoluir.
- O SUT precisa estar disponível durante a captura de um caso de teste.
- Só é viável para um escopo pequeno e uma SUT que raramente muda.
- Em comparação com a Captura/Reprodução, é necessário algum conhecimento de programação.

Scripting estruturado

As bibliotecas de teste são introduzidas com elementos reutilizáveis, etapas de teste e/ou jornadas do usuário. Nessa abordagem, é necessário conhecimento de programação para a criação e a manutenção de scripts de teste.

Prós

- Fácil de manter, dimensionar, portar, adaptar e evoluir.
- A lógica de negócio pode ser separada dos scripts de teste.

Contras

- É necessário ter conhecimento de programação.
- O investimento inicial no desenvolvimento do TAF e na definição do software de teste é demorado.

Desenvolvimento orientado por teste

Os casos de teste são definidos como parte do processo de desenvolvimento antes que um novo recurso do SUT seja implementado. A abordagem TDD é testar, codificar e refatorar ou também conhecida como vermelho, verde e refatorar. Um desenvolvedor identifica e cria um caso de teste que falhará (vermelho). Em seguida, ele desenvolve a funcionalidade que satisfará o caso de teste (verde). O código é então refatorado para otimizá-lo e obedecer aos princípios de código limpo. O processo continua com o próximo teste e o próximo incremento de funcionalidade.

Prós

- Simplifica o desenvolvimento de casos de teste em nível de componente.
- Melhora a qualidade do código e a estrutura do código.
- Melhora a capacidade de teste.
- Facilita a obtenção de uma cobertura de código desejada.
- Reduz a propagação de defeitos para níveis de teste mais altos.
- Melhora a comunicação entre desenvolvedores, representantes do negócio e testadores.
- Histórias de usuários que não são verificadas por meio de testes de GUI e testes de API podem atingir rapidamente os critérios de saída seguindo o TDD.

Contras

- Inicialmente, leva mais tempo para se acostumar com o TDD.
- Não seguir o TDD corretamente pode resultar em falsa confiança na qualidade do código.

Testes orientado por dados

O teste orientado por dados (DDT) baseia-se na abordagem de scripting estruturado. Os scripts de teste são fornecidos com os dados de teste (p. ex., arquivos .csv, arquivos .xlsx e dumps de banco de dados). Isso permite a execução dos mesmos scripts de teste várias vezes com dados de teste diferentes.

Prós

- Permite a expansão rápida e fácil do caso de teste por meio de feeds de dados.
- O custo de adicionar novos testes automatizados pode ser reduzido significativamente.

- Os analistas de teste podem especificar testes automatizados preenchendo um ou mais arquivos de dados de teste que descrevem os testes. Isso dá aos analistas de testes mais liberdade para especificar testes automatizados com menos dependência dos analistas de testes técnicos.

Contras

- Pode ser necessário um gerenciamento adequado dos dados de teste.

Teste orientado por palavras-chave

Os casos de teste do Teste Orientado por Palavras-chave (KDT) são uma lista ou tabela de etapas de teste derivadas de palavras-chave e dos dados de teste nos quais as palavras-chave operam. As palavras-chave são definidas a partir da perspectiva do usuário. Essa técnica geralmente é baseada no DDT.

Prós

- Os analistas de teste e os analistas de negócios podem se envolver na criação de casos de teste automatizados seguindo a abordagem KDT.
- O KDT também pode ser usado para testes manuais, independentemente da automação de teste (consulte a norma ISO/IEC/IEEE 29119-5 sobre teste orientado por palavras-chave).

Contras

- A implementação e a manutenção de palavras-chave é uma tarefa complexa que os TAE precisam cobrir, o que pode se tornar um desafio quando o escopo aumenta.
- Isso gera um enorme esforço para sistemas menores.

Desenvolvimento orientado pelo comportamento

O BDD utiliza um formato de linguagem natural (ou seja, dado, quando e então) para formular critérios de aceite que podem ser usados como casos de teste automatizados e armazenados em arquivos. Uma ferramenta de BDD pode então entender a linguagem e executar os casos de teste.

Prós

- Melhora a comunicação entre desenvolvedores, representantes de negócio e testadores.
- Os cenários automatizados de BDD funcionam como casos de teste e garantem a cobertura das especificações.
- O BDD pode ser aproveitado na produção de vários tipos de teste em diferentes níveis da pirâmide de testes.

Contras

- Casos de teste adicionais, normalmente condições de teste negativos e casos limítrofes, ainda precisam ser definidos pela equipe, normalmente por um analista de teste ou um TAE.
- Muitas equipes confundem o BDD como sendo apenas uma forma de escrever casos de teste em uma linguagem natural e não envolvem os representantes negócio e os desenvolvedores em toda a abordagem.
- Implementar e manter as etapas de teste de linguagem natural é uma tarefa complexa para os TAE cobrirem.
- Etapas de teste excessivamente complexas transformarão a depuração em uma atividade difícil e cara.

3.1.5 Aplicar princípios e padrões de design na automação de teste.

A automação de teste é uma atividade de desenvolvimento de software. Portanto, os princípios e padrões de design são tão importantes para um TAE quanto para um desenvolvedor de software.

Princípios de programação orientado a objetos

Há quatro princípios principais de programação orientada a objetos: encapsulamento, abstração, herança e polimorfismo.

Princípios SOLID

É um acrônimo de responsabilidade única (S), aberto-fechado (O), substituição de Liskov (L), substituição de interface (I) e inversão de dependência (D). Esses princípios melhoram a legibilidade, a capacidade de manutenção e o dimensionamento do código.

Padrões de design

Entre os vários padrões de design, três são mais importantes para as TAE.

O *padrão de fachada oculta* detalhes de implementação para expor apenas o que os testadores precisam criar nos casos de teste, e o *padrão singleton* é frequentemente usado para garantir que haja apenas um driver que se comunique com o SUT.

No *modelo de objeto de página*, um arquivo de classe é criado e chamado de modelo de página. Sempre que a estrutura do SUT for alterada, o TAE precisará fazer atualizações em apenas um local, o localizador dentro de um modelo de página, em vez de atualizar os localizadores em cada caso de teste.

O *padrão de modelo de fluxo* é uma expansão do modelo de objeto de página. Ele introduz uma fachada adicional sobre os modelos de objeto de página, que armazena todas as ações do usuário que interagem com os objetos de página. Ao introduzir um design de fachada dupla, o padrão de modelo de fluxo fornece uma abstração e uma capacidade de manutenção aprimoradas, pois as etapas de teste podem ser reutilizadas em vários scripts de teste.

4 Implementação da Automação de Teste – 150 min

Palavras-chave

risco, dispositivo de teste

Objetivos de aprendizagem:

4.1 Desenvolvimento de Automação de Teste

TAE-4.1.1 (K3) Aplicar diretrizes que apoiem atividades piloto e de implementação eficazes de automação de teste.

4.2 Riscos associados ao desenvolvimento da Automação de Teste

TAE-4.2.1 (K4) Analisar os riscos da implantação e planejar estratégias de mitigação para a automação de teste.

4.3 Manutenção da solução de Automação de Teste

TAE-4.3.1 (K2) Explicar quais fatores apoiam e afetam a manutenção da solução de automação de teste.

4.1 Desenvolvimento de Automação de Teste

4.1.1 Aplicar diretrizes que apoiem atividades piloto e de implementação eficazes de automação de teste

É importante definir o escopo de validação para um piloto de automação de teste. Um projeto piloto não leva muito tempo para conduzir, mas o resultado pode ter um impacto significativo na direção que o projeto toma.

Com base nas informações coletadas sobre o SUT e os requisitos do projeto, o seguinte deve ser avaliado para estabelecer diretrizes para otimizar os esforços de automação de teste:

- Linguagem(ns) de programação que será(ão) usada(s).
- Ferramentas comerciais prontas para uso ou de código aberto adequadas.
- Níveis de teste a serem cobertos.
- Casos de teste selecionados.
- Abordagem de desenvolvimento de casos de teste.

Com base nos pontos listados acima, os TAE podem definir uma abordagem inicial a ser seguida. Com base nos requisitos, vários protótipos iniciais diferentes podem ser criados para mostrar os prós e os contras das diferentes abordagens. A partir daí, os TAE podem decidir qual caminho seguir.

Definir cronogramas é uma parte importante dos horários das reuniões e garantir o sucesso do piloto. Uma recomendação comum é verificar periodicamente o progresso do projeto piloto para identificar quaisquer riscos e mitigá-los.

Durante o piloto, também é recomendável tentar integrar a solução e o código já implementado na CI/CD. Isso pode antecipar problemas, seja no SUT, no TAS ou na integração geral de diferentes ferramentas dentro da organização.

À medida que o número de casos de teste aumenta, os TAE podem pensar em alterar a configuração inicial de CI/CD para executar os testes de diferentes maneiras e em momentos diferentes.

Além disso, durante o projeto piloto, é necessário avaliar outros aspectos não técnicos, como:

- O conhecimento e a experiência dos membros da equipe.
- A estrutura da equipe.
- Regras de licenciamento e organização.
- O tipo de teste planejado e os níveis de teste visados a serem cobertos durante a automação dos casos de teste.

Após a conclusão do projeto piloto, o esforço deve ser avaliado pelos TAE e gerentes de teste para avaliar o sucesso ou o fracasso e tomar uma decisão apropriada.

4.2 Riscos associados ao desenvolvimento da Automação de Teste

4.2.1 Analisar os riscos de implantação e planejar estratégias de mitigação para a automação de teste

A interface do TAF com o SUT precisa ser considerada como parte do projeto arquitetônico. Em seguida, o empacotamento, o registro de testes e as ferramentas de podem ser selecionadas.

Durante a implementação do piloto, a expansão e a manutenção do código de automação de teste precisam ser consideradas. Esses são fatores cruciais da fase de avaliação do piloto e podem afetar seriamente a decisão final.

Diferentes riscos de implantação podem ser identificados a partir do piloto:

- Aberturas de firewall
- Utilização de recursos (p. ex., CPU e RAM)

É preciso se preparar para os riscos da implantação, como problemas de firewall, utilização de recursos, conexão de rede e confiabilidade. Esses fatores não estão estritamente ligados à automação de teste, mas as TAE precisam garantir que todas as condições sejam atendidas para fornecer portas de qualidade confiáveis e benéficas em seu processo de desenvolvimento.

O uso de dispositivos reais para automação de teste móveis é um exemplo. Os dispositivos móveis devem estar ligados, ter bateria suficiente para funcionar durante o teste, estar conectados a uma rede e ter acesso à SUT.

Os riscos de implementação técnica incluem:

- Empacotamento.
- Registro em log.
- Estruturação dos testes.
- Atualização.

Empacotamento

O empacotamento precisa ser considerado como controle de versão da automação de teste é tão importante quanto para o SUT. O Testware pode precisar ser carregado em um repositório para ser compartilhado em uma organização, seja no local ou na nuvem.

Registro em log

O registro de testes fornece a maior parte das informações sobre os resultados dos testes. Há vários níveis de registro de testes e todos eles são úteis na automação de teste por vários motivos:

- *Fatal*: esse nível é usado para registrar eventos de erro que podem levar à interrupção da execução do teste.
- *Erro*: esse nível é usado quando uma condição ou interação falha e, portanto, também falha no caso de teste.
- *Alerta*: Esse nível é usado quando ocorre uma condição/ação inesperada, mas não interrompe o fluxo do caso de teste.
- *Informativo*: Esse nível é usado para mostrar informações básicas sobre um caso de teste e o que acontece durante a execução do teste.
- *Depuração*: Esse nível é usado para armazenar detalhes específicos da execução que geralmente não são necessários para os registros básicos, mas são úteis durante a investigação de uma falha no teste.
- *Rastreamento*: Esse nível é semelhante ao Debug, mas contém ainda mais informações

Estruturação dos testes

A parte mais importante do TAS é o conjunto de testes e os dispositivos de teste incluídos nele, itens que devem estar disponíveis para que os testes sejam executados. Os dispositivos de teste oferecem liberdade no controle de um ambiente e dos dados de teste. As pré-condições e pós-condições podem ser definidas para a execução do teste e os casos de teste podem ser agrupados em conjuntos de teste de várias maneiras. Esses aspectos também são importantes para avaliar durante um projeto piloto. Além disso, os dispositivos de teste permitem a criação de testes automatizados que são repetíveis e atômicos.

Atualização

Um dos riscos técnicos mais comuns são as atualizações automáticas nas estruturas de teste (p. ex., agentes) e as alterações de versão nos dispositivos. Esses riscos podem ser atenuados com fontes de alimentação adequadas, conexões de rede apropriadas e planos de configuração de dispositivos adequados.

4.3 Manutenção da solução de Automação de Teste

4.3.1 Explicar quais fatores apoiam e afetam a capacidade de manutenção da solução de automação de teste

A capacidade de manutenção é altamente afetada pelos padrões de programação e pelas expectativas dos TAE entre si.

Uma regra de ouro é tentar seguir os princípios de código limpo de Robert C. Martin (Robert C Martin, "Clean Code: A Handbook of Agile Software Craftsmanship", 2008).

Resumidamente, princípios de código limpo enfatizam os seguintes pontos:

- Usar uma convenção de nomenclatura comum para classes, métodos e variáveis para obter nomes significativos.
- Usar uma estrutura de projeto lógica e comum.
- Evitar hardcoding.
- Evitar o excesso de parâmetros de entrada para métodos.
- Evitar métodos longos e complexos.
- Usar registro em log.
- Usar padrões de design onde eles forem benéficos e necessários.
- Focar na capacidade de teste.

As convenções de nomenclatura são muito úteis para identificar o destino de uma determinada variável. Ter nomes de variáveis compreensíveis, como "loginButton", "resetPasswordButton", ajuda os TAE a entender qual componente usar.

Hardcoding é o processo de incorporação de valores no software sem a possibilidade de alterá-los diretamente. Isso pode ser evitado com o uso de testes orientados por dados para que os dados de teste venham de uma fonte comum que possa ser mantida facilmente. O hardcoding reduz o tempo de desenvolvimento, mas não é recomendável usá-lo, pois os dados podem mudar com frequência, o que pode consumir muito tempo de manutenção. Também é recomendável usar constantes para as variáveis que não se espera que mudem com frequência. Dessa forma, os códigos-fonte e os locais que precisam ser atualizados podem ser reduzidos.

O aproveitamento dos padrões de projeto também é altamente recomendado. Os padrões de projeto, conforme descrito em 3.1.5, permitem a implementação de um código de automação de teste estruturado e de manutenção adequada, desde que os padrões de projeto sejam usados corretamente.

Para garantir a alta qualidade do código de automação, recomenda-se o uso de analisadores estáticos. Os formatadores de código, como os comumente usados em IDEs, melhorarão a legibilidade do código de automação de teste.

Além dos princípios de código limpo, é recomendável usar uma estrutura e uma estratégia de ramificação acordadas no controle de versão. O uso de diferentes ramificações para recursos, versões e correções de defeitos é útil para entender o conteúdo das ramificações.

5 Estratégias de implementação e implantação para Automação de Teste – 90 min

Palavras-chave

teste de contrato

Objetivos de aprendizagem:

5.1 Integração aos pipelines de CI/CD

TAE-5.1.1 (K3) Aplicar automação de teste em diferentes níveis de teste nos pipelines.

TAE-5.1.2 (K2) Explicar o gerenciamento de configuração para testware.

TAE-5.1.3 (K2) Explicar as dependências de automação de teste para uma infraestrutura de API.

5.1 Integração com pipelines de CI/CD

5.1.1 Aplicar a automação de teste em diferentes níveis de teste nos pipelines

Um dos principais benefícios da automação de teste é que os testes implementados podem ser executados sem supervisão, o que os torna candidatos ideais para execução em pipelines. Isso pode ser feito por meio de pipelines de CI/CD ou do pipeline usado para executar os testes regularmente.

Os níveis de teste geralmente são integrados da seguinte forma:

- Os testes de configuração para TAF/TAS, durante a compilação, podem ser considerados como uma subcategoria de testes de componentes. Esses testes são executados durante a compilação de um projeto de automação de (TAF/TAS) e verificam se todos os caminhos para os arquivos usados nos scripts de teste estão corretos, se os arquivos realmente existem e se estão localizados nos caminhos especificados.
- Os testes de componentes fazem parte da etapa de compilação do pipeline, pois são executados nos componentes individuais (p. ex., classes de biblioteca e componentes da Web). Eles atuam como portas de qualidade para o pipeline e, portanto, são uma parte essencial de um pipeline de integração contínua.
- Os testes de integração de componentes podem fazer parte do pipeline de integração contínua se forem testes de componentes de baixo nível ou do SUT. Nesses casos, esses testes e os testes de componentes são executados juntos.
- Os testes de sistema geralmente podem ser integrados a um pipeline de implantação contínua, onde atuam como a última porta de qualidade do SUT entregue.
- Os testes de integração do sistema entre diferentes componentes do sistema geralmente fazem parte de um pipeline de entrega contínua como portas de qualidade. Esses testes de integração do sistema garantem que os componentes do sistema desenvolvidos separadamente estejam funcionando juntos.

Muitos sistemas modernos de integração contínua diferenciam as fases de compilação e implantação dos pipelines de entrega contínua. Nesses casos, os testes de componentes e os testes de integração de componentes fazem parte da primeira fase de compilação. Quando essa primeira fase é bem-sucedida (ou seja, compilação e teste juntos), os componentes/SUT são implantados.

No caso de testes de integração de sistemas, testes de sistemas e testes de aceite, há duas abordagens principais para integrá-los a esses pipelines:

1. Os casos de teste são executados como parte da fase de implementação após a implementação do componente. Isso pode ser benéfico, pois, com base nos resultados do teste, a implementação pode falhar e também ser revertida. No entanto, nesse caso, se os testes precisarem ser executados novamente, será necessário fazer uma reimplantação.
2. Os casos de teste são executados como um pipeline separado, acionado pela implantação bem-sucedida. Isso pode ser vantajoso se for esperado que diferentes conjuntos de testes e vários códigos de automação de teste sejam serão executados em cada implementação. Nesse caso, os testes não funcionam como uma porta de qualidade. Portanto, são necessárias outras ações, geralmente manuais, para reverter uma implementação malsucedida. Nesse caso, alguns scripts de teste automatizados simples, como verificações de implantação, podem ser usados para garantir que o SUT seja implantado, mas esses scripts de teste automatizados não verificam a adequação funcional de forma ampla.

Os pipelines também podem ser usados para outros fins de automação de teste como, por exemplo:

- Executar diferentes conjuntos de testes periodicamente: Um conjunto de testes de regressão pode ser executado todas as noites (ou seja, a regressão noturna), especialmente para conjuntos de testes de execução mais longa, para que a equipe tenha uma visão clara da qualidade do SUT pela manhã.
- Executar testes não funcionais: Parte de um pipeline de implantação contínua ou separadamente, para monitorar periodicamente determinadas características de qualidade não funcionais do sistema, como a eficiência do performance.

5.1.2 Explicar o gerenciamento de configuração do software de teste

O gerenciamento de configuração é parte integrante da automação de teste pois a automação geralmente é executada em vários ambientes de teste e versões do SUT.

Gerenciamento de configuração na automação de teste inclui:

- Configuração do ambiente de teste.
- Dados de teste.
- Suíte de teste/casos de teste.

Configuração do ambiente de teste

Cada ambiente de teste usado no pipeline de desenvolvimento pode ter configurações diferentes, como vários URLs ou credenciais. A configuração do ambiente de teste geralmente é armazenada com o testware. No entanto, no caso da automação de teste usada em vários projetos ou em vários TAF para o mesmo projeto, a configuração do ambiente de teste pode fazer parte da biblioteca de núcleo comum ou em um repositório compartilhado.

Dados de teste

Os dados de teste também podem ser específicos para o ambiente de teste ou para a versão e o conjunto de recursos do SUT. Assim como na configuração do ambiente de teste, os dados de teste geralmente são armazenados em TAF menores, mas também podem ser usados sistemas de gerenciamento de dados de teste.

Suíte de teste/casos de teste

Uma prática comum é configurar diferentes suítes de teste dos casos de teste, com base em sua finalidade, como *smoke test* ou teste de regressão. Esses conjuntos de testes geralmente são executados em níveis de teste separados, aproveitando diferentes pipelines e ambientes de teste.

Cada versão do SUT determina um conjunto de recursos que inclui casos de teste e suítes de teste que avaliam a qualidade da versão em questão. Há diferentes opções no testware para lidar com isso:

- Uma configuração de alternância de recursos pode ser definida para cada versão ou ambiente de teste. Há casos de teste e suítes de teste para testar cada recurso. A alternância de recursos pode ser usada no testware para identificar quais suítes de teste devem ser executadas em uma determinada versão/ambiente de teste.
- O software de teste também pode ser lançado com o SUT usando a mesma versão de lançamento. Dessa forma, há uma correspondência exata entre a versão do SUT e o testware que pode testá-lo. Essa versão geralmente é implementada por meio de um sistema de gerenciamento de configuração que usa marcas ou ramificações.

5.1.3 Explicar as dependências da automação de teste para uma infraestrutura de API.

Ao realizar a automação de teste de API é fundamental ter as seguintes informações sobre dependências para criar uma estratégia adequada:

- Conexões de API: Entender a lógica de negócio que pode ser testada automaticamente e a relação entre as APIs.
- Documentação da API: Serve como baseline para a automação de teste com todas as informações relevantes (p. ex., parâmetros, cabeçalhos e tipos distintos de solicitação-resposta de objetos)

Testes de API automatizados integrados podem ser feitos pelos desenvolvedores ou pelos TAE. No entanto, com o *shift-left*, recomenda-se apoiar e dividir o teste entre diferentes níveis. No ISTQB CTFL Syllabus, testes de integração de componentes e testes de integração de sistemas são mencionados, que podem ser estendidos com uma prática recomendada chamada de teste de contrato.

Teste de contrato

O teste de contrato é um tipo de teste de integração que verifica se os serviços podem se comunicar uns com os outros e se os dados compartilhados entre os serviços são consistentes com um conjunto especificado de regras. O uso de testes de contrato fornece compatibilidade entre dois sistemas separados (p. ex., dois microsserviços) para que se comuniquem entre si. Ele vai além da validação do esquema, exigindo que ambas as partes cheguem a um consenso sobre o conjunto de interações permitidas e, ao mesmo tempo, permitindo a evolução ao longo do tempo. Ele captura as interações que são trocadas entre cada serviço, armazenando-as em um contrato, que pode ser usado para verificar se ambas as partes aderem a ele. Uma das principais vantagens desse tipo de teste é que os defeitos que ocorrem nos serviços subjacentes podem ser encontrados mais cedo no SDLC e a origem desses defeitos pode ser identificada com mais facilidade.

Na abordagem orientada para o consumidor dos testes contratuais, o consumidor define sua expectativa, determinando como o provedor deve responder às solicitações provenientes desse consumidor. Na abordagem orientada para o provedor para testes contratuais, o provedor cria o contrato, que mostra como seus serviços estão operando.

6 Relatórios e métricas de Automação de Teste – 150 min

Palavras-chave

Medição, métrica, registro de teste, relatório de progresso do teste, etapa de teste

Objetivos de aprendizagem:

6.1 Coleta, análise e relatório de dados de automação de teste

TAE-6.1.1 (K3) Aplicar métodos de coleta de dados da solução de automação de teste e do sistema em teste.

TAE-6.1.2 (K4) Analisar os dados da solução de automação de teste e do sistema em teste para entender melhor os resultados.

TAE-6.1.3 (K2) Explicar como um relatório de progresso de teste é construído e publicado

6.1 Coleta, análise e relatório de dados de Automação de Teste

6.1.1 Aplicar métodos de coleta de dados da solução de automação de teste e do sistema em teste

Os dados podem ser coletados das seguintes fontes:

- Registros do SUT.
  - UI da Web/móvel;
  - APIs;
  - Aplicativos;
  - Servidores da Web;
  - Servidores de banco de dados.
- Registros de TAF para fornecer uma trilha de auditoria.
- Registros de construção.
- Registros de implantação.
- Logs de produção para monitorar dados em produção (consulte o syllabus ISTQB CT-PT Syllabus, seção 2.3).
  - Monitoramento de performance na produção para realizar análise de tendências;
  - Registros de teste de eficiência de performance em um ambiente de teste de performance (p. ex., testes de carga, estresse e picos);
- Capturas de tela e gravações de tela (nativas da ferramenta de automação ou de terceiros).

Como um TAS tem um testware automatizado em seu núcleo, o testware automatizado pode ser aprimorado para registrar informações sobre seu uso. Os aprimoramentos do testware feitos no testware subjacente podem ser usados por todos os scripts de teste automatizados de nível superior. Por exemplo, o aprimoramento do testware subjacente para registrar a hora de início e de término da execução do teste pode ser aplicado a todos os testes.

Recursos de automação de teste que suportam a medição e a geração de relatórios de teste

As linguagens de script de muitas ferramentas de teste oferecem suporte à medição e relatórios por meio de recursos que podem ser usados para gravar e registrar informações antes, durante e depois da execução de testes individuais e de conjuntos de testes inteiros.

O relatório de teste de cada uma das séries de execuções de teste precisa ter um recurso de análise para considerar os resultados dos testes anteriores, de modo que possa destacar tendências, como alterações na taxa de sucesso do teste.

A automação de teste normalmente requer a automatização da execução e da verificação do teste, sendo esta última obtida pela comparação de elementos específicos dos resultados reais com os resultados esperados. A melhor maneira de fazer essa comparação é por meio de uma ferramenta de teste que usa afirmações. Deve-se considerar o nível de informação que é relatado como resultado dessa comparação. É importante que o status do teste seja determinado corretamente (ou seja, aprovado ou reprovado). No caso de status de falha, serão necessárias mais informações sobre a causa da falha (p. ex., capturas de tela).

As diferenças entre os resultados reais e os resultados esperados de um teste nem sempre são claras, e o suporte da ferramenta pode ajudar muito na definição de comparações que ignoram as diferenças esperadas, como datas e horários, ao mesmo tempo em que destaca as diferenças inesperadas.

Registro de testes

Os registros de teste são uma fonte usada com frequência para analisar possíveis defeitos no TAS e no SUT. Na seção a seguir, há exemplos de registro de testes, categorizados por TAS e SUT.

Registro da TAS

O contexto determina se a TAF ou a execução do teste é responsável pelo registro de informações que devem incluir:

- Qual caso de teste está sendo executado no momento, incluindo a hora de início e de término
- O status da execução do teste, pois, embora as falhas possam ser facilmente identificadas nos registros de teste, o TAF também deve ter essas informações e deve informar por meio de um painel. O status da execução do teste pode ser aprovado, reprovado ou uma falha do TAS. Às vezes, o status pode ser inconclusivo e é importante que a organização possua definições claras e consistentes para eles. Uma falha de TAS é aplicada a situações em que o defeito não está no SUT.
- Detalhes de baixo nível do registro de teste (p. ex., registro de etapas de teste significativas), incluindo informações de tempo.
- Informações dinâmicas sobre o SUT (p. ex., vazamentos de memória) que o caso de teste foi capaz de identificar com a ajuda de ferramentas de terceiros. Os resultados reais e as falhas devem ser registrados com o conjunto de testes que foi executado quando a falha foi detectada.
- No caso de testes de confiabilidade ou de estresse em que são realizados vários ciclos de teste, um contador deve ser registrado para determinar facilmente quantas vezes os casos de teste foram executados.
- Quando os casos de teste tiverem elementos aleatórios (p. ex., parâmetros aleatórios ou etapas de teste aleatórias no teste de transição de estado), o número/escolhas aleatórias devem ser registrados.
- Todas as ações que um caso de teste executa devem ser registradas de forma que os registros de teste, ou partes deles, possam ser reproduzidos para executar novamente o teste com as mesmas etapas de teste e o mesmo tempo. Isso é útil para reproduzir uma falha identificada e capturar informações adicionais. As informações sobre a ação do caso de teste também podem ser registradas pelo SUT para uso na reprodução de falhas identificadas pelo cliente. Se um cliente executar um conjunto de testes, as informações do registro de testes são capturadas e podem ser reproduzidas pela equipe de desenvolvimento ao solucionar um defeito.
- As capturas de tela podem ser salvas durante a execução do teste para uso posterior durante a análise da causa raiz.
- Sempre que um conjunto de testes inicia uma falha, o TAS deve se certificar de que todas as informações necessárias para analisar o defeito estejam disponíveis/armazenadas, bem como qualquer informação referente à continuação do teste, se aplicável. O TAS deve salvar todos os dumps de falhas e rastreamentos de stacks associadas. Além disso, todos os registros de teste que possam ser sobrescritos (p. ex., buffers cíclicos são usados com frequência para registros de teste no SUT) devem ser armazenados em um local onde estarão disponíveis para análise posterior.
- O uso de cores pode ajudar a distinguir diferentes tipos de informações de registro de teste (p. ex., defeitos em vermelho e informações de progresso em verde).

Registro de SUT

Correlação dos resultados da automação de teste com os registros do SUT para ajudar a identificar a causa raiz dos defeitos no SUT e no TAS.

- Quando um defeito é identificado no SUT, todas as informações necessárias para analisar o defeito devem ser registradas, incluindo carimbos de data e hora, local de origem do defeito e mensagens de erro.
- Na inicialização de um sistema, as informações de configuração devem ser registradas em um arquivo, que consiste, por exemplo, nas diferentes versões de software/firmware, na configuração do SUT e na configuração do sistema operacional.
- Usando a automação de teste, os registros de SUT podem ser facilmente pesquisáveis. Uma falha identificada no registro de teste pelo TAS deve ser facilmente identificada no registro de teste do SUT e vice-versa, com ou sem ferramentas adicionais. A sincronização de vários registros de teste com um registro de data e hora facilita a correlação do que ocorreu quando uma falha foi relatada.

Integração com outras ferramentas de terceiros (p. ex., planilhas, XML, documentos, bancos de dados e ferramentas de relatórios)

Quando as informações da execução de casos de teste automatizados são usadas em outras ferramentas para rastreamento e relatório (p. ex., atualização de informações de rastreabilidade), é possível fornecer as informações em um formato adequado para ferramentas de terceiros. Isso geralmente é obtido por meio da funcionalidade da ferramenta de teste existente (p. ex., formatos de exportação para relatórios de teste) ou pela criação de relatórios personalizados que são produzidos em um formato consistente com outros softwares.

Visualização dos resultados dos testes

Os resultados dos testes podem se tornar visíveis por meio de gráficos. Considere o uso de ícones coloridos, como semáforos, para indicar o status geral da execução do teste/automação do teste para que as decisões possam ser tomadas com base nas informações relatadas. A gerência está particularmente interessada em resumos visuais para ver os resultados dos testes, o que ajuda na tomada de decisões. Se forem necessárias mais informações, eles ainda poderão se aprofundar nos detalhes.

6.1.2 Analisar os dados da solução de automação de teste e do sistema em teste para entender melhor os resultados.

Após a execução do teste, é importante analisar os resultados do teste para identificar possíveis falhas tanto no SUT quanto no TAS. Para essa análise, os dados coletados do TAS são primários e os dados coletados do SUT são secundários.

- Analisar os dados do ambiente de teste para dar suporte ao dimensionamento adequado da automação de teste (p. ex., na nuvem)
    - Clusters e recursos (p. ex., CPU e RAM)
    - Execução de testes com um ou vários navegadores (ou seja, entre navegadores)
- Comparar os resultados das execuções de testes anteriores
- Determinar como usar os registros da Web para monitorar o uso do software

As falhas na execução do teste precisam ser analisadas, pois há possíveis problemas:

1. Verifique se a mesma falha ocorreu nas execuções de teste anteriores. Isso pode ser um defeito conhecido no SUT ou no TAS. O TAS pode ser criado para registrar o histórico dos resultados dos casos de teste, o que ajuda ainda mais a análise.
2. Se o defeito não for conhecido, identifique o caso de teste e o que ele está testando. O teste pode ser autoexplicativo ou o caso de teste pode ser identificado no sistema de gerenciamento de testes, com base em sua ID registrada com a execução do teste.
3. Descubra em qual etapa do caso de teste ocorreu a falha. O TAS registra isso.
4. Analisar no registro de teste Informações sobre o estado do SUT e se ele corresponde aos resultados esperados usando capturas de tela, API e registros de rede ou qualquer registro que mostre o estado do SUT.
5. Se o estado do SUT não for o esperado, registre um defeito no sistema de gerenciamento de defeitos. Certifique-se de incluir todas as informações necessárias sobre o defeito e os dados que justificam que se trata de um defeito.

Quando há uma falha, é possível que o resultado real e o resultado esperado do SUT não coincidam. Nesse caso, é muito provável que o TAS contenha um defeito que precisa ser corrigido ou que haja uma incompatibilidade invisível.

Outra situação que pode ocorrer é se o ambiente de teste não estiver disponível durante a execução do teste, ou estiver disponível apenas parcialmente. Nesse caso, todos os casos de teste podem falhar, seja com o mesmo defeito ou se partes do sistema estiverem inativas, com falhas aparentemente reais. Para identificar a causa raiz de tais defeitos, os registros do SUT podem ser analisados, o que mostrará se houve alguma interrupção no ambiente de teste no momento da execução do teste.

Se o SUT implementar registros de auditoria para interações de usuários (ou seja, sessões de interface do usuário ou chamadas de API), isso ajudará a analisar os resultados dos testes. Geralmente, há uma ID exclusiva adicionada à interação com a mesma ID para cada chamada e integração subsequentes no sistema. Dessa forma, conhecendo a ID exclusiva de uma solicitação/interação, o comportamento do sistema pode ser observado e rastreado.

Essa ID exclusiva geralmente é chamada de ID de correlação ou ID de rastreamento. Ela pode ser registrada pelo TAS para ajudar a analisar os resultados do teste.

6.1.3 Explicar como um relatório de progresso do teste é construído e publicado.

Os registros de teste fornecem informações detalhadas sobre as etapas de teste, as ações a serem tomadas e as respostas esperadas de um caso de teste e/ou conjunto de testes. No entanto, os registros de teste por si só não podem fornecer uma boa visão geral dos resultados gerais do teste. Para isso, é necessário ter uma funcionalidade de relatório de teste. Após a execução de um conjunto de testes, um relatório conciso do progresso do teste deve ser criado e publicado. Um gerador de relatórios pode ser usado para isso.

Conteúdo de um relatório de progresso do teste

O relatório de progresso do teste deve conter os resultados do teste, as informações do SUT e a documentação do ambiente de teste no qual os testes foram executados em um formato apropriado para cada uma dos stakeholders.

É necessário saber quais testes falharam e os motivos da falha. Para facilitar a solução de problemas, é importante conhecer o histórico de execução do teste e quem o relatou (ou seja, geralmente a pessoa que o criou ou o atualizou pela última vez). A pessoa responsável precisa investigar a causa da falha, relatar o defeito relacionado a ela, acompanhar a correção do defeito e testar se a correção foi implementada corretamente.

O relatório de testes também é usado para diagnosticar falhas nos componentes do TAF.

Publicação dos relatórios de teste

O relatório de teste deve ser publicado para todos os stakeholders relevantes. Ele pode ser carregado em um site, na nuvem ou nas instalações, enviado para uma lista de discussão ou carregado em outra ferramenta, como uma ferramenta de gerenciamento de testes. Isso ajuda a garantir que os relatórios sejam revisados e analisados se as pessoas estiverem inscritas para recebê-los por e-mail ou por meio de mensagens de bate-papo publicadas por um chatbot.

Uma opção é identificar partes problemáticas do SUT e manter um histórico dos relatórios de teste, de modo que as estatísticas sobre casos de teste ou conjuntos de teste com regressões frequentes possam ser reunidas para análise de tendências.

Os stakeholders a serem informados incluem:

- stakeholders de gestão
    - Funções típicas: arquiteto de soluções ou corporativo, gerente de projeto/entrega, gerente de programa, gerente de teste ou diretor de teste
- stakeholders operacionais
    - Funções típicas: proprietário/gerente do produto, representante comercial ou analista de negócio
- stakeholders técnicos
    - Funções típicas: líder de equipe, mestre de scrum, administrador da Web, desenvolvedor/administrador de banco de dados, líder de teste, TAE, testador ou desenvolvedor

Os relatórios de teste podem variar em conteúdo ou detalhes, dependendo dos destinatários. Enquanto os stakeholders técnicos podem estar mais interessados em detalhes de nível inferior, a gerência se concentrará em tendências, como quantos casos de teste foram adicionados desde a última execução de teste, alterações na taxa de aprovação/reprovação e a confiabilidade do TAS e do SUT. Os stakeholders operacionais geralmente dão mais ênfase às métricas relacionadas ao uso do produto.

Criação de painéis de controle

As ferramentas modernas de geração de relatórios oferecem várias opções de relatórios por meio de painéis, gráficos coloridos, coleções detalhadas de registros e análise automatizada de registros de testes. automatizados. Há muitas ferramentas disponíveis no mercado para você escolher.

Essas ferramentas oferecem suporte à agregação de dados de fontes como registros de testes de execução de pipeline, ferramentas de gerenciamento de projetos e repositórios de código. A visualização de dados fornecida por essas ferramentas ajuda os stakeholders a ver as tendências e a tomar decisões de acordo com elas. Essas tendências podem incluir grupos de defeitos, aumento/diminuição da propagação de defeitos para determinados ambientes de teste, degradação do performance do SUT e confiabilidade das compilações.

Análise de inteligência artificial/aprendizado de máquina dos registros de teste

Nos últimos anos, algumas ferramentas de automação de teste incluem ou são baseadas em algoritmos de aprendizado de máquina (ML). A análise automatizada de grandes quantidades de dados em registros de teste ajuda o TAE a reduzir o tempo gasto para encontrar localizadores quebrados, analisar o motivo das falhas de teste (ou seja, é um defeito no SUT ou no TAS?) e agrupar defeitos comuns para relatórios de teste (consulte o ISTQB CT-AI Syllabus).

7 Verificação da Solução de Automação de Teste – 135 min

Palavras-chave

análise estática

Objetivos de aprendizagem:

7.1 Verificação da infraestrutura de Automação de Teste

TAE-7.1.1 (K3) Planejar a verificação do ambiente de automação de teste, incluindo a configuração da ferramenta de teste.

TAE-7.1.2 (K2) Explicar o comportamento correto de um determinado script de teste automatizado e/ou conjunto de testes.

TAE-7.1.3 (K2) Identificar onde a automação de teste produz resultados inesperados.

TAE-7.1.4 (K2) Explicar como a análise estática pode ajudar na qualidade do código da automação de teste.

7.1 Verificação da Infraestrutura de Automação de Teste

7.1.1 Planejar a verificação do ambiente de automação de teste, incluindo a configuração da ferramenta de teste

Se o ambiente de automação de teste O ambiente de automação de teste e todos os outros componentes do TAS funcionam conforme o esperado ainda requerem verificação. Essas verificações são feitas, por exemplo, antes de iniciar a automação de teste. Podem ser adotadas etapas para verificar os componentes do ambiente de automação de teste. Cada uma delas é explicada em mais detalhes a seguir.

Instalação, configuração e personalização da ferramenta de teste

O TAS é composto por muitos componentes. É preciso contar com cada um deles para garantir uma performance confiável e repetível. No centro de um TAS estão os componentes executáveis, as bibliotecas de funções correspondentes e os arquivos de dados e configuração de suporte. O processo de configuração de um TAS pode variar desde o uso de scripts de instalação automatizados até a colocação manual de arquivos nas pastas correspondentes. As ferramentas de teste, assim como os sistemas operacionais e outros softwares, têm regularmente pacotes de serviços ou podem ter suplementos opcionais ou necessários para garantir a compatibilidade com qualquer ambiente SUT específico.

A instalação automatizada ou a cópia de um repositório tem vantagens. Ela pode garantir que os testes em diferentes SUT tenham sido realizados com a mesma versão do TAS e a mesma configuração do TAS, quando apropriado. As atualizações do TAS podem ser feitas por meio do repositório. O uso do repositório e o processo de atualização para uma nova versão do TAS devem ser os mesmos das ferramentas de desenvolvimento padrão.

Repetibilidade na configuração/desmontagem do ambiente de teste

Um TAS será implementado em uma variedade de sistemas, servidores e para dar suporte a pipelines de CI/CD. Para garantir que o TAS funcione corretamente em cada ambiente de teste, é necessário ter uma abordagem sistemática para carregar e descarregar o TAS de qualquer ambiente de teste específico. Isso é alcançado com sucesso quando a construção e a reconstrução do TAS não oferecem nenhuma diferença perceptível na forma como ele opera dentro e entre vários ambientes de teste. O gerenciamento de configuração dos componentes do TAS garante que uma determinada configuração possa ser criada de forma confiável. Quando isso for feito, a documentação dos vários componentes que compõem o TAS fornecerá o conhecimento necessário sobre os aspectos do TAS que podem ser afetados ou exigir mudanças quando o ambiente do SUT for alterado.

Conectividade com sistemas/interfaces internos e externos

Depois que um TAS é instalado em um determinado ambiente de SUT e antes de usar o SUT, um conjunto de verificações ou pré-condições deve ser administrado para garantir que a conectividade com sistemas internos, sistemas externos e interfaces esteja disponível. Por exemplo, é uma boa prática fazer login nos servidores, iniciar as ferramentas de automação de teste, verificar se as ferramentas de automação de teste estão disponíveis e se o SUT está disponível. verificar se as ferramentas de automação de teste podem acessar o SUT, inspecionar manualmente as definições de configuração e garantir que as permissões sejam definidas corretamente para o registro e o relatório de testes entre os sistemas. Estabelecer condições prévias para a automação de teste é essencial para garantir que o TAS tenha sido instalado e configurado corretamente.

Teste de componentes TAF

Assim como em qualquer projeto de desenvolvimento de software, os componentes da TAF precisam ser testados e verificados individualmente. Isso pode incluir testes funcionais e não funcionais (p. ex., eficiência de performance e utilização de recursos). Por exemplo, os componentes que fornecem verificação de objetos em sistemas GUI precisam ser testados em uma ampla gama de classes de objetos para estabelecer que a verificação de objetos funciona corretamente. Da mesma forma, os registros e relatórios de testes devem produzir informações precisas sobre o status da automação de teste e o comportamento do SUT. e do comportamento do SUT. Exemplos de testes não funcionais podem incluir a compreensão da degradação do performance da TAF, a utilização de recursos do sistema que podem indicar defeitos, como vazamentos de memória, e a falta de interoperabilidade de componentes dentro e/ou fora da TAF.

7.1.2 Explicar o comportamento correto de um determinado script de teste automatizado e/ou conjunto de testes.

Os conjuntos de testes automatizados precisam ser testados quanto à integridade, à consistência e ao comportamento correto. Diferentes tipos de verificações podem ser aplicados para garantir que o conjunto de testes automatizados esteja disponível em um determinado momento ou para determinar se ele está apto para uso.

Podem ser adotadas etapas para verificar o conjunto de testes automatizados. Essas etapas incluem:

- Verificar a composição do conjunto de testes;
- Verificar os novos testes que se concentram em novos recursos do TAF;
- Considerar a repetibilidade dos testes;
- Considerar a intrusividade das ferramentas de teste automatizadas.

Cada um deles é explicado em mais detalhes a seguir.

Verificar a composição do conjunto de testes

Verificar a integridade (p. ex., se todos os casos de teste possuem um resultado esperado e seus dados de teste estão presentes) e a versão correta do TAF e do SUT.

Verificar os novos testes que se concentram em novos recursos do TAF

Na primeira vez em que um novo recurso do TAF for usado em casos de teste, ele deverá ser verificado e monitorado de perto para garantir que esteja funcionando corretamente.

Considerar a repetibilidade dos testes

Ao repetir os testes, os resultados devem ser sempre os mesmos. Os casos de teste no conjunto de testes que não fornecerem um resultado de teste confiável (p. ex., devido a condições de execução) devem ser movidos do conjunto de testes automatizados ativos e analisados separadamente para encontrar a causa raiz. Caso contrário, o tempo será gasto repetidamente nesses testes para analisar a falha.

Considerar a intrusividade das ferramentas de teste automatizadas

O TAS geralmente é fortemente acoplado ao SUT. Isso é planejado para que haja melhor compatibilidade no que diz respeito ao nível de interações. Entretanto, essa estreita integração também pode levar a resultados adversos. Por exemplo, quando o TAS está localizado dentro do ambiente do SUT, a funcionalidade do SUT pode ser diferente de quando os testes são realizados manualmente, o que também pode afetar a performance.

Um alto nível de intrusão pode mostrar falhas durante os testes que não são evidentes na produção. Se isso causar falhas nos testes automatizados, a confiança no TAS pode cair drasticamente. Os desenvolvedores podem exigir que as falhas identificadas pela automação de teste sejam reproduzidas manualmente, se possível, para ajudar na análise.

7.1.3 Identificar onde a automação de teste produz resultados inesperados

Quando um script de teste falha ou passa inesperadamente, a análise da causa raiz deve ser realizada. Isso incluirá a inspeção de registros de teste, dados de performance, configuração e desmontagem do script de teste.

Também é útil executar alguns testes isolados. As falhas intermitentes são mais difíceis de analisar. O defeito pode estar no caso de teste, no SUT, no TAF, no hardware ou na rede. O monitoramento dos recursos do sistema pode fornecer pistas sobre a causa raiz. A análise do arquivo de registro de teste do caso de teste, do SUT e do TAF pode ajudar a identificar a causa raiz do defeito. A depuração também pode ser necessária. Para ajudar a identificar a causa raiz, pode ser necessário o apoio de um analista de testes, analista de negócios, desenvolvedor ou engenheiro de sistemas.

Verifique se todas as afirmações estão em vigor. A falta de afirmações pode causar resultados de teste inconclusivos.

7.1.4 Explicar como a análise estática pode ajudar na qualidade do código de automação de teste.

A análise de código estático pode ajudar a encontrar vulnerabilidades e defeitos no código do programa. Isso pode incluir o SUT ou o TAF.

As varreduras automatizadas podem inspecionar o código para reduzir os riscos. Isso proporciona uma revisão do SUT em busca de defeitos e garante que os padrões de qualidade do código estejam sendo cumpridos e aplicados. Isso também pode ser considerado uma técnica proativa de detecção de defeitos, que desempenha uma função importante nas implementações de DevSecOps (ou seja, DevOps com ênfase em Segurança). Essas varreduras ocorrem no início do SDLC por meio de pipelines para fornecer feedback imediato às equipes de desenvolvimento.

O resultado referente aos defeitos geralmente é categorizado como de gravidade crítica, alta, média ou baixa, para que as equipes de desenvolvimento possam priorizar os defeitos que desejam corrigir. Algumas ferramentas de análise estática também têm a capacidade de fornecer sugestões de correções de código para solucionar os defeitos encontrados. Elas apresentarão às equipes de desenvolvimento uma cópia das linhas de código problemáticas e oferecerão uma possível solução a ser implementada pelos desenvolvedores. Além disso, essas ferramentas auxiliam as TAE medindo a qualidade, sugerindo áreas onde comentar o código, aprimorando o design do código para otimizar o manuseio de recursos (p. ex., utilizando blocos try-catch e melhores estruturas de looping) e removendo chamadas de biblioteca ruins.

Como as ferramentas de automação de teste usam linguagens de programação, há o risco de que um código de automação de teste inadequado possa ser introduzido no SDLC. Como exemplo simples, uma prática comum na automação de teste é ter um nome de usuário e uma senha. É possível que um TAE inclua incorretamente a senha em texto simples em um ou vários scripts de teste.

As ferramentas de análise estática podem beneficiar o código de automação de teste. Elas podem ser usadas para analisar o código de automação de teste em busca de violações de segurança, como uma senha de texto simples dentro do código. As ferramentas de análise estática são compatíveis com muitas linguagens de programação, inclusive as usadas pelo software de automação de teste. Portanto, é imperativo que o TAE estenda as práticas recomendadas de verificação de código para incluir também o código de automação de teste. Mesmo que o código de automação de teste não seja necessariamente implantado com o software geral, há claramente possíveis vulnerabilidades se a senha for descoberta em um script de teste de automação que o acompanha (consulte o ISTQB CT-SEC Syllabus).

8 Melhoria contínua – 210 min

Palavras-chave

validação do esquema, histograma de teste

Objetivos de aprendizagem:

8.1 Oportunidades de melhoria contínua para Automação de Teste

TAE-8.1.1 (K3) Descobrir oportunidades para melhorar os casos de teste por meio da coleta e análise de dados.

TAE-8.1.2 (K4) Analisar os aspectos técnicos de uma solução de automação de teste implantada e recomendar melhorias.

TAE-8.1.3 (K3) Reestruturar o software de teste automatizado para alinhar com as atualizações do sistema em teste.

TAE-8.1.4 (K2) Resumir as oportunidades de uso de ferramentas de automação de teste.

8.1 Oportunidades de melhoria contínua para Automação de Teste

8.1.1 Descobrir oportunidades para melhorar os casos de teste por meio da coleta e análise de dados

A coleta e a análise de dados podem ser aprimoradas ao considerar vários tipos de dados com as abordagens descritas abaixo.

Histograma de teste

Um relatório visual dos dados de teste representados como um histograma de teste fornece possíveis áreas de melhoria com relação às tendências dos dados do caso de teste. Os TAE podem decidir sobre as possíveis áreas de melhoria, pois muitas ferramentas de CI/CD e de relatório de teste têm a capacidade de mostrar diferentes resultados de teste e seus respectivos dados de teste (p. ex., logs de exceção, mensagens de erro e capturas de tela). O histograma de teste também permite que os TAE identifiquem e selecionem os casos de teste que são frágeis e os refaçam com melhorias adicionais ou repensando a implementação real.

Inteligência Artificial

Outra oportunidade recente é a utilização da Inteligência Artificial (IA) para dar suporte a testes e automação de teste. Por exemplo, nos casos de teste de interface do usuário, os dados também incluem valores de localizador de interface do usuário que podem ser tratados como entradas. Ferramentas de ponta recentes mostram a capacidade de detectar se um determinado localizador foi alterado em relação ao usado. Com base em ML e reconhecimento de imagem, elas podem identificar os novos seletores e usar um algoritmo de autocorreção para corrigir o caso de teste e incluir os localizadores alterados no relatório de teste. Isso pode acelerar as etapas de acompanhamento, como controle de versão alterações e manutenção de código.

Validação do esquema

A validação de esquema pode ser aplicada na análise de dados de API (p. ex., propriedades derivadas de pontos de extremidade de destino) e na análise de banco de dados (p. ex., regras de validação para campos de software, como tipos de dados e intervalos de valores permitidos).

Com a validação do esquema o TAS é capaz de verificar se uma resposta corresponde à especificação real de negócio. Esse tipo de verificação pode ser usado para determinar se os elementos obrigatórios da resposta estão presentes na resposta do serviço e se o tipo de objeto corresponde ao definido no esquema. Em caso de quebra do esquema, a solução retorna à validação real que ajuda os TAE a identificar a causa raiz do problema.

Exemplo: uma API tem seis elementos de resposta obrigatórios que devem ser cadeias de caracteres na resposta. Com as ferramentas de validação de esquema não é necessário escrever afirmações individuais para verificar se esses tipos são strings e se seus valores não são nulos. A validação do esquema cuidará dessas verificações, tornando o código de automação de teste implementado muito mais curto e aumentando a eficiência da detecção de defeitos no serviço de backend.

8.1.2 Analisar os aspectos técnicos de uma solução de automação de teste implantada e recomendar melhorias.

Além das tarefas de manutenção contínua necessárias para manter o TAS sincronizado com o SUT, há muitas oportunidades de aprimoramento do TAS. Essas melhorias podem ser feitas para obter uma série de benefícios, inclusive maior eficiência (p. ex., reduzindo ainda mais a intervenção manual), maior facilidade de uso, recursos adicionais e suporte aprimorado para testes. A decisão de como o TAS será aprimorado é influenciada por quais recursos agregam mais valor a um projeto.

As áreas específicas de um TAS que podem ser consideradas para aprimoramento incluem scripts, execução de testes, verificação, TAA, TAF, configuração e desmontagem, documentação, recursos do TAS e atualizações e upgrades do TAS. Essas áreas são descritas em mais detalhes abaixo.

Scripting

As técnicas de script variam desde o scripting linear ao teste orientado por dados e até a abordagem mais sofisticada de teste orientado por palavras-chave conforme descrito na seção 3.1.4. Pode ser apropriado atualizar a técnica de script atual do TAS para todos os novos testes automatizados. A técnica pode ser adaptada a todos os testes automatizados existentes ou, pelo menos, àqueles que envolvem o maior esforço de manutenção.

Outra área de aprimoramento do TAS para scripts de teste pode se concentrar em sua implementação. Por exemplo:

- Avaliar o script de teste/caso de teste/etapa de teste para consolidar os testes automatizados. Os casos de teste que contêm sequências de ações semelhantes não devem implementar essas etapas de teste várias vezes. Essas etapas de teste devem ser transformadas em uma função e adicionadas a uma biblioteca, para que possam ser reutilizadas. Essas funções de biblioteca podem então ser usadas por diferentes casos de teste. Isso aumenta a capacidade de manutenção do material de teste. Quando as etapas de teste não são idênticas, mas semelhantes, a parametrização pode ser necessária. Observação: essa é uma abordagem típica em testes orientados por palavras-chave.
- Estabelecer um processo de recuperação de falhas para o TAS e o SUT. Quando ocorre uma falha durante a execução de um conjunto de testes, o TAS deve ser capaz de se recuperar dessa condição para continuar com o próximo teste possível. Quando ocorre uma falha no SUT, o TAS precisa executar as ações de recuperação necessárias no SUT (p. ex., uma reinicialização do SUT) quando for viável e prático.
- Avaliar os mecanismos de espera para garantir que o melhor tipo esteja sendo usado. Há três mecanismos de espera comuns:
    - Esperas codificadas (ou seja, esperar um determinado número de milissegundos), que podem ser a causa raiz de muitos defeitos de automação de teste de automação de teste, dada a imprevisibilidade dos tempos de resposta do software.
    - A espera dinâmica por sondagem (p. ex., verificar se uma determinada mudança de estado ou ação ocorreu) é muito mais flexível e eficiente:
        - O TAS aguarda apenas o tempo necessário e não há desperdício de tempo de teste
        - Quando o processo demorar mais do que o esperado, a sondagem aguardará até que a condição seja verdadeira. Lembre-se de incluir um mecanismo de tempo limite, caso contrário, o teste poderá esperar para sempre se houver um defeito.
    - Uma maneira ainda melhor é assinar o mecanismo de eventos do SUT. Isso é muito mais confiável do que as outras duas opções, mas a linguagem de script de teste precisa oferecer suporte à assinatura de eventos e o SUT precisa oferecer esses eventos ao TAS. Também é necessário um mecanismo de tempo limite, ou o teste poderá esperar para sempre se houver um defeito.

Execução de teste

Quando um conjunto de testes de regressão automatizado não é concluído porque a execução do teste demora muito, pode ser necessário se possível, testar simultaneamente em diferentes ambientes de teste, se isso for possível. Quando sistemas caros são usados para testes, pode ser uma restrição que todos os testes sejam feitos em um único sistema de destino. Pode ser necessário dividir o conjunto de testes de regressão em várias partes, cada uma executada em um período de tempo definido (p. ex., em uma única noite). Uma análise mais aprofundada da cobertura da automação de teste pode revelar duplicação. A remoção da duplicação pode reduzir o tempo de execução do teste e gerar mais eficiência. No caso de CI/CD, uma boa prática é executar trabalhos em lote em paralelo para otimizar o tempo de execução do teste. Além disso, é uma boa prática programar trabalhos em lote automatizados para executar os diferentes pipelines em um determinado momento, por exemplo, todas as manhãs, para reduzir as interações manuais e acelerar o processo de desenvolvimento.

Verificação

Antes de criar novas funções de verificação, adote um conjunto de métodos de verificação padrão para uso em todos os testes automatizados. Isso evitará a reimplementação de ações de verificação em vários testes. Quando

os métodos de verificação não forem idênticos, mas semelhantes, o uso da parametrização ajudará a permitir que uma função seja usada em vários tipos de objetos.

TAA

Pode ser necessário alterar o TAA para dar suporte a melhorias na testabilidade do SUT. Essas alterações podem ser feitas na arquitetura do SUT e/ou no TAA do TAS. Isso pode proporcionar uma grande melhoria na automação do teste, mas pode exigir mudanças e investimentos significativos no SUT/TAS. Por exemplo, se o SUT for alterado para fornecer APIs para teste, o TAS também deverá ser refatorado de acordo. Adicionar esses tipos de recursos mais tarde no SDLC pode ser bastante caro; é muito melhor pensar nisso no início da automação de teste e nas fases iniciais do SDLC do SUT.

TAF

Frequentemente, há novas versões das bibliotecas principais usadas em uma TAF. Às vezes, essas são atualizações importantes, e a versão mais recente não pode ser referenciada imediatamente na lista de dependências da TAF, pois isso interromperia os testes de muitas das equipes que as utilizam. Portanto, é melhor realizar primeiro um piloto e uma análise de impacto. Depois disso, é possível criar um plano de adoção. Todas as equipes adotam a nova versão das bibliotecas essenciais ao mesmo tempo, atualizando a dependência no arquivo de compilação da camada de bibliotecas essenciais, ou cada equipe decide individualmente quando atualizá-la em sua camada de lógica de negócios. Eventualmente, quando todas as equipes estiverem prontas para aceitar a nova versão das bibliotecas centrais, as dependências poderão ser atualizadas na camada de bibliotecas centrais (consulte a seção 3.1.3).

Configuração e desmontagem

As ações e configurações que se repetem antes ou depois de cada script de teste ou conjunto de testes devem ser movidas para métodos de configuração ou desmontagem. Dessa forma, todas as alterações que afetam o código podem ser atualizadas em um único local, o que resulta em menos esforços de manutenção. Por exemplo, as chamadas de serviço da Web podem ser usadas para atender às pré-condições ou pós-condições dos testes de UI (p. ex., registro de usuário, limpeza de usuário e configuração de perfil).

Documentação

Isso abrange todas as formas de documentação da automação de teste (p. ex., o que o código de automação de teste faz e como ele deve ser usado), a documentação do usuário para o TAS e os relatórios de teste e registros de teste produzidos pelo TAS.

Recursos da TAS

Adicione recursos e funções do TAS, como relatórios de teste detalhados, registros de teste e integração com outros sistemas. Adicione apenas novos recursos que serão usados. A adição de recursos não utilizados só aumenta a complexidade e diminui a confiabilidade e a capacidade de manutenção.

Atualizações e upgrades do TAF

Ao atualizar ou fazer o upgrade para novas versões do TAF, novas funções podem ficar disponíveis e ser usadas pelos casos de teste ou falhas podem ser corrigidas. O risco é que a atualização da TAF por meio do upgrade das ferramentas de teste existentes ou da introdução de novas ferramentas pode ter um impacto adverso sobre os casos de teste existentes. Teste a versão mais recente da ferramenta de teste executando uma amostra de testes antes de implementar a nova versão da ferramenta. A amostra de teste deve ser representativa dos testes automatizados de diferentes SUT, diferentes tipos de teste e, quando apropriado, diferentes ambientes de teste.

8.1.3 Reestruturar o software de teste automatizado para se alinhar com as atualizações do sistema em teste

Após um determinado conjunto de alterações em um SUT existente, serão necessárias atualizações no TAS, incluindo o TAF e as bibliotecas de componentes. Qualquer alteração, por mais trivial que seja, pode ter um impacto adverso abrangente sobre a confiabilidade e o performance do TAS.

Identificar alterações nos componentes do ambiente de teste

Avalie quais mudanças e melhorias precisam ser feitas. Elas exigem alterações no testware, nas bibliotecas de funções personalizadas ou no sistema operacional? Cada um desses aspectos tem um impacto sobre o performance do TAS. O objetivo geral é garantir que os testes automatizados continuem a ser executados de maneira eficiente. As alterações devem ser feitas de forma incremental, com uma mentalidade de produto mínimo viável, de modo que o impacto no TAS possa ser medido por meio de uma execução limitada de scripts de teste. Quando for constatado que não há efeito colateral, as alterações poderão ser totalmente implementadas. Uma execução de regressão completa é a última etapa para verificar se a alteração não afetou negativamente os scripts de teste automatizados. Durante a execução desses scripts de teste de regressão, podem ser encontradas falhas. A identificação da causa raiz dessas falhas (p. ex., por meio de relatórios de teste, registros de teste e análise de dados de teste) fornecerá um meio de garantir que elas não resultem da atividade de melhoria da automação de teste.

Aumentar a eficiência e a eficácia das principais bibliotecas de funções da TAS

À medida que um TAS amadurece, são descobertas novas maneiras de executar tarefas com mais eficiência. Essas novas técnicas (p. ex., otimização de código em funções e uso de bibliotecas de sistemas operacionais mais recentes) precisam ser incorporadas às bibliotecas de funções principais usadas pelo projeto atual e por projetos futuros.

Direcionar várias funções que atuam no mesmo tipo de controle para consolidação

Uma grande parte do que ocorre durante a execução de um teste automatizado é a interrogação de controles na GUI. Essa interrogação serve para fornecer informações sobre um controle (p. ex., visível/não visível, ativado/não ativado, tamanho e dimensões e dados). Com essas informações, um teste automatizado pode selecionar um item de uma lista suspensa, inserir dados em um campo e ler um valor de um campo. Há várias funções que podem agir sobre os controles para obter essas informações. Algumas funções são extremamente especializadas, enquanto outras são de natureza mais geral. Por exemplo, pode haver uma função específica que funcione apenas em listas suspensas. Como alternativa, pode haver uma função que funcione com várias funções, especificando uma delas como um de seus parâmetros. Portanto, um TAE pode usar várias funções que podem ser consolidadas em menos funções, obtendo os mesmos resultados e minimizando a manutenção.

Refatorar o TAA para acomodar as alterações no SUT

Durante o ciclo de vida de um TAS, será necessário fazer alterações para acomodar as mudanças no SUT. À medida que o SUT evolui e amadurece, o TAA subjacente também terá de evoluir para garantir que a capacidade esteja presente para dar suporte ao SUT. Deve-se tomar cuidado ao estender os recursos para que eles não sejam implementados de forma fixa, mas sim analisados e alterados na TAA. Isso garantirá que, à medida que a nova funcionalidade do SUT exigir scripts de teste adicionais, haverá componentes compatíveis para acomodar esses novos testes automatizados.

Convenções de nomenclatura e padronização

À medida que as mudanças são introduzidas, as convenções de nomenclatura para o novo código de automação de teste e bibliotecas de funções e bibliotecas de funções precisam ser consistentes com os padrões definidos anteriormente (consulte a seção 4.3.1).

Avaliação de scripts de teste existentes para revisão/eliminação do SUT

O processo de mudança e aprimoramento também inclui uma avaliação dos scripts de teste existentes, seu uso e valor contínuo. Por exemplo, se determinados testes forem complexos e demorados para serem executados, decompô-los em testes menores pode ser mais viável e eficiente. A eliminação dos testes que não são executados com frequência ou que não são executados de forma alguma reduzirá a complexidade do TAS e deixará mais claro o que precisa ser mantido.

8.1.4 Resumir as oportunidades de uso de ferramentas de automação de teste

Além do teste real, a automação de teste pode ajudar em atividades de teste não específicas, como:

Configuração e controle do ambiente

Certos scripts de teste (p. ex., criação de dados de teste) podem ser aproveitados em um método de configuração para criar dados de teste diferentes em um novo ambiente de teste. Em uma situação em que usuários com vários perfis precisam ser criados com base em diferentes entradas de dados, uma equipe pode usar um script de teste automatizado para chamar o encerramento de serviço da Web que registre esses usuários. Os scripts de teste podem ter controle sobre a configuração da infraestrutura de teste e podem ser aproveitados em uma limpeza após o processo. Isso ajuda a economizar tempo e a garantir que os usuários adequados estejam presentes em cada novo ambiente de teste. Por exemplo, diferentes registros de teste e outros materiais de teste podem ser removidos de um ambiente de teste automaticamente, tornando a limpeza e o uso do ambiente de teste mais eficientes.

Envelhecimento de dados

A automação de teste pode ser usada para manipular os dados de teste no ambiente de teste. Por exemplo, em bancos de dados, os campos de data podem ser verificados e controlados para mantê-los atualizados do ponto de vista anual.

Geração de capturas de tela e vídeos

A maioria das ferramentas modernas de automação de teste têm um recurso integrado para criar capturas de tela ou vídeos em determinadas condições e armazená-los. Com essas ferramentas de teste, as equipes podem ajudar a empresa a criar capturas de tela e vídeos de uso real para documentação de lançamento de software ou para fins de marketing.

9 Referências

Normas

As normas para Automatização de Testes incluem, entre outros:

A Automatic Test Markup Language (ATML) do IEEE (Institute of Electrical and Electronics Engineers) consiste em:

- IEEE Std 1671.1: Test Description
- IEEE Std 1671.2: Instrument Description
- IEEE Std 1671.3: UUT Description
- IEEE Std 1671.4: Test Configuration Description
- IEEE Std 1671.5: Test Adaptor Description
- IEEE Std 1671.6: Test Station Description
- IEEE Std 1641: Signal and Test Definition
- IEEE Std 1636.1: Test Results

ISO/IEC/IEEE 29119-5 (2016) Software and Systems Engineering – Software testing – Part 5: Keyword-Driven Testing

ISO/IEC 30130:2016 (E) Software engineering — Capabilities of software testing tools

A Testing and Test Control Notation (TTCN-3) do ETSI (European Telecommunication Standards Institute) e ITU (International Telecommunication Union) consiste em:

- ES 201 873-1: TTCN-3 Core Language
- ES 201 873-2: TTCN-3 Tabular Presentation Format (TFT)
- ES 201 873-3: TTCN-3 Graphical Presentation Format (GFT)
- ES 201 873-4: TTCN-3 Operational Semantics
- ES 201 873-5: TTCN-3 Runtime Interface (TRI)
- ES 201 873-6: TTCN-3 Control Interface (TCI)
- ES 201 873-7: Using ASN.1 with TTCN-3
- ES 201 873-8: Using IDL with TTCN-3
- ES 201 873-9: Using XML with TTCN-3
- ES 201 873-10: TTCN-3 Documentation
- ES 202 781: Extensions: Configuration and Deployment Support
- ES 202 782: Extensions: TTCN-3 Performance and Real-Time Testing
- ES 202 784: Extensions: Advanced Parameterization
- ES 202 785: Extensions: Behaviour Types
- ES 202 786: Extensions: Support of interfaces with continuous signals
- ES 202 789: Extensions: Extended TRI

A UML Testing Profile (UTP) do OMG (Object Management Group) que especifica os conceitos de Especificação de Teste para:

- Test Architecture
- Test Data
- Test Behavior
- Test Logging
- Test Management

Documentos ISTQB®

Identificador	Referência
ISTQB-AL-TTA	ISTQB Certified Tester, Advanced Level Syllabus, Technical Test Analyst, versão 4.0, junho de 2021, disponível em [ISTQB-Web]
ISTQB-FL	ISTQB Certified Tester, Foundation Level Syllabus, versão 4.0, abril de 2023, disponível em [ISTQB-Web]
ISTQB-MBT	ISTQB Certified Tester, Model-Based Testing Syllabus, versão 1.0, outubro de 2015, disponível em [ISTQB-Web]

ISTQB-PT	ISTQB Certified Tester, Performance Testing Syllabus, dezembro de 2018, disponível em [ISTQB-Web]
ISTQB-SEC	ISTQB Certified Tester, Security Tester Syllabus, versão 1.0, março de 2016, disponível em [ISTQB-Web]
ISTQB-TAS	ISTQB Certified Tester, Test Automation Strategy Syllabus, fevereiro de 2024, disponível em [ISTQB-Web]
ISTQB-Glossary	ISTQB Glossary of Terms, disponível online em [ISTQB-Web]

Livros

Paul Baker, Zhen Ru Dai, Jens Grabowski, e Ina Schieferdecker, “Model-Driven Testing: Using the UML Testing Profile”, Springer 2008 edition, ISBN-10: 3540725628, ISBN-13: 978-3540725626

Efriede Dustin, Thom Garrett, Bernie Gauf, “Implementing Automated Software Testing: how to save time and lower costs while raising quality”, Addison-Wesley, 2009, ISBN 0-321-58051-6

Efriede Dustin, Jeff Rashka, John Paul, “Automated Software Testing: introduction, management, and performance”, Addison-Wesley, 1999, ISBN-10: 0201432870, ISBN-13: 9780201432879

Mark Fewster, Dorothy Graham, “Experiences of Test Automation: Case Studies of Software Test Automation”, Addison-Wesley, 2012

Mark Fewster, Dorothy Graham, “Software Test Automation: Effective use of test execution tools”, ACM Press Books, 1999, ISBN-10: 0201331403, ISBN-13: 9780201331400

Robert C Martin, “Clean Code: A Handbook of Agile Software Craftsmanship”, 2008, ISBN-10: 9780132350884

James D. McCaffrey, “.NET Test Automation Recipes: A Problem-Solution Approach”, APRESS, 2006 ISBN-13:978-1-59059-663-3, ISBN-10:1-59059-663-3

Daniel J. Mosley, Bruce A. Posey, “Just Enough Software Test Automation”, Prentice Hall, 2002, ISBN-10: 0130084689, ISBN-13: 9780130084682

Manikandan Sambamurthy, “Test Automation Engineering Handbook”, January 2023, ISBN: 9781804615492

Colin Willcock, Thomas Deiß, Stephan Tobies and Stefan Keil, “An Introduction to TTCN-3” Wiley, 2nd edition 2011, ISBN-10: 0470663065, ISBN-13: 978-0470663066

Artigos

Robert V. Binder, Suzanne Miller, “Five Keys to Effective Agile Test Automation for Government Programs” August 24, 2017, Software Engineering Institute, Carnegie Mellon University, https://resources.sei.cmu.edu/asset_files/Webinar/2017_018_101_503516.pdf

DoD CIO, Modern Software Practices “DevSecOps Fundamentals Guidebook: Activities & Tools”, Version 2.2, May 2023, https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsActivitesToolsGuidebookTables.pdf?ver=_Sylg1WJB9K0Jxb2XTvzDQ%3d%3d

Péter Földházi, “Tri-Layer Testing Architecture”, https://www.pnsqc.org/docs/PROP53522057-FoldhaziDraftFinal.pdf

Thomas Pestak, William Rowell, PhD, “Automated Software Testing Practices and Pitfalls”, September 2018, https://www.afit.edu/stat/statcoe_files/Automated%20Software%20Testing%20Practices%20and%20Pitfalls%20Rev%201.pdf

Andrew Pollner, Jim Simpson, Jim Wisnowski, “Automated Software Testing Implementation Guide for Managers and Practitioners”, October 2018, https://www.afit.edu/stat/statcoe_files/0214simp%202%20AST%20IG%20for%20Managers%20and%20Practitioners.pdf

The Selenium Project, “Page object models”, https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/

Siglas usadas neste syllabus

gTTA: Arquitetura Genérica de Automação de Teste

SUT: Sistema em Teste

TAA: Arquitetura de Automação de Teste

TAE: Engenheiro de Automação de Teste

TAF: Framework de Automação de Teste

TAS: Solução de Automação de Teste

10 Apêndice A: Objetivos de Aprendizagem e Níveis Cognitivos de Conhecimento

Os seguintes Objetivos de Aprendizagem são definidos como aplicáveis a este syllabus. Cada tópico do syllabus será examinado de acordo com seu objetivo de aprendizado.

Os Objetivos de Aprendizagem começam com um verbo de ação correspondente ao seu nível cognitivo de conhecimento, conforme listado abaixo.

Nível 2: Compreender (K2)

O candidato pode selecionar as razões ou explicações para declarações relacionadas ao tópico e pode resumir, comparar, classificar e dar exemplos para o conceito de teste.

Verbos de ação: Classificar, comparar, diferenciar, distinguir, explicar, dar exemplos, interpretar, resumir

Exemplos	Notas
Classificar as ferramentas de teste de acordo com sua finalidade e as atividades de teste que elas suportam.	
Comparar os diferentes níveis de teste.	Pode ser usado para procurar semelhanças, diferenças ou ambos.
Diferenciar teste de depuração.	Procura diferenças entre os conceitos.
Distinguir entre riscos do projeto e do produto.	Permite que dois (ou mais) conceitos sejam classificados separadamente.
Explicar o impacto do contexto no processo de teste.	
Dar exemplos de por que os testes são necessários.	
Inferir a causa raiz dos defeitos a partir de um determinado perfil de falhas.	
Resumir as atividades do processo de revisão do produto de trabalho.	

Nível 3: Aplicar (K3)

O candidato pode executar um procedimento quando confrontado com uma tarefa familiar ou selecionar o procedimento correto e aplicá-lo a um determinado contexto.

Verbos de ação: Aplicar, implementar, preparar, usar

Exemplos	Notas
Aplicar a análise de valor limite para derivar casos de teste a partir de determinados requisitos.	Deve se referir a um procedimento / técnica / processo etc.
Implementar métodos de coleta de métricas para dar suporte aos requisitos técnicos e gerenciais.	
Preparar testes de instalação para aplicativos móveis.	
Usar a rastreabilidade para monitorar o progresso do teste quanto à integridade e à consistência com os objetivos, a estratégia e o plano de teste.	Pode ser usado em um objetivo de aprendizagem que deseja que o candidato seja capaz de usar uma técnica ou um procedimento. Semelhante a "aplicar".

Nível 4: Analisar (K4)

O candidato pode separar as informações relacionadas a um procedimento ou técnica em suas partes constituintes para melhor compreensão e pode distinguir entre fatos e inferências. Uma aplicação típica é analisar um documento, software ou situação de projeto e propor ações apropriadas para resolver um problema ou tarefa.

Verbos de ação: Analisar, desconstruir, delinear, priorizar, selecionar.

Exemplos	Notas
Analisar uma determinada situação de projeto para determinar quais técnicas de teste caixa-preta ou baseadas em experiência devem ser aplicadas para atingir objetivos específicos.	Examinável somente em combinação com um objetivo mensurável da análise. Deve ser do tipo 'Analyze xxxx to xxxx' (ou similar).
Priorizar os casos de teste em um determinado conjunto de testes para execução com base nos riscos relacionados ao produto.	
Selecionar os níveis de teste e os tipos de teste adequados para verificar um determinado conjunto de requisitos.	Necessário quando a seleção requer análise.

Referência

(Para os níveis cognitivos dos objetivos de aprendizagem)

Anderson, L. W. e Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching, and

Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon

11 Apêndice B: Matriz de rastreabilidade entre Resultados de Negócio e Objetivos de Aprendizagem

Esta seção lista a rastreabilidade entre os resultados de negócios e os objetivos de aprendizagem do especialista em Test Automation Engineering.

Resultados de Negócio do Test Automation Engineering			N01	N02	N03	N04	N05	N06	N07	N08	N09	N10	N11	N12
Cap.1	Introdução e objetivos da automação de teste													
1.1	Objetivo da automação de teste													
1.1.1	Explicar as vantagens e desvantagens da automação de teste	2	X											
1.2	Automação de teste no ciclo de vida de desenvolvimento de software													
1.2.1	Explicar como a automação de teste é aplicada em diferentes modelos de ciclo de vida de desenvolvimento de software	2		X										
1.2.2	Selecionar ferramentas de automação de teste ferramentas adequadas para um determinado sistema em teste	2		X										
Cap.2	Preparando-se para a automação de teste													
2.1	Descrever as necessidades de configuração de uma infraestrutura que permita a automação de teste													
2.1.1	Descrever as necessidades de configuração de uma infraestrutura que permita a implementação da automação de teste	2			X									
2.1.2	Explicar como a automação de teste é aproveitada em diferentes ambientes	2			X									
2.2	Processo de avaliação para selecionar as ferramentas e as estratégias corretas													
2.2.1	Analisar um sistema em teste para determinar a solução apropriada de automação de teste	4				X								
2.2.2	Ilustrar os resultados técnicos de uma avaliação de ferramenta	4				X								
Cap.3	Arquitetura de automação de teste													
3.1	Conceitos de design aproveitados na automação de teste													
3.1.1	Explicar os principais recursos em uma arquitetura de automação de teste	2					X							
3.1.2	Explicar como projetar uma solução de automação de teste	2					X							
3.1.3	Aplicar camadas de automação de teste	3					X							
3.1.4	Aplicar diferentes abordagens para automatizar casos de teste	3					X							

Resultados de Negócio do Test Automation Engineering			N01	N02	N03	N04	N05	N06	N07	N08	N09	N10	N11	N12
3.1.5	Aplicar princípios e padrões de design na automação de teste	3					X							
Cap.4	Implementação da automação de teste													
4.1	Desenvolvimento de automação de teste													
4.1.1	Aplicar diretrizes que apoiem atividades piloto e de implementação eficazes de automação de teste	3						X						
4.2	Riscos associados ao desenvolvimento da automação de teste													
4.2.1	Analisar riscos de implantação e planejar estratégias de mitigação para automação de teste	4							X					
4.3	Manutenção da solução de automação de teste													
4.3.1	Explicar quais fatores apoiam e afetam a capacidade de manutenção da solução de automação de teste	2								X				
Cap.5	Estratégias de implementação e implantação para automação de teste													
5.1	Integração com pipelines de CI/CD													
5.1.1	Aplicar a automação de teste em diferentes níveis de teste nos pipelines	3									X			
5.1.2	Explicar o gerenciamento de configuração do software de teste	2									X			
5.1.3	Explicar as dependências da automação de teste para uma infraestrutura de API	2									X			
Cap.6	Relatórios e métricas de automação de teste													
6.1	Coleta, análise e relatório de automação de teste													
6.1.1	Aplicar métodos de coleta de dados da solução de automação de teste e do sistema	3										X		
6.1.2	Analisar os dados da solução de automação de teste e do sistema em teste para entender melhor os resultados	4										X		
6.1.3	Explicar como um relatório de progresso do teste é construído e publicado	2										X		
Cap.7	Verificação da solução de automação de teste													
7.1	Verificação da infraestrutura de automação de teste													
7.1.1	Planejar a verificação do ambiente de automação de teste, incluindo a configuração da ferramenta de teste	3											X	
7.1.2	Explicar o comportamento correto de um determinado script de teste automatizado e/ou conjunto de teste	2											X	
7.1.3	Identificar onde a automação de teste produz resultados inesperados	2											X	

Resultados de Negócio do Test Automation Engineering			N01	N02	N03	N04	N05	N06	N07	N08	N09	N10	N11	N12
7.1.4	Explicar como a análise estática pode ajudar na qualidade do código da automação de teste	2											X	
Cap.8	Melhoria contínua													
8.1	Oportunidades de melhoria contínua para automação de teste													
8.1.1	Descobrir oportunidades para melhorar os casos de teste por meio da coleta e análise de dados	3												X
8.1.2	Analisar os aspectos técnicos de uma solução de automação de teste implantada de testes e recomendar melhorias	4												X
8.1.3	Reestruturar o software de teste automatizado para alinhá-lo às atualizações do sistema em teste	3												X
8.1.4	Resumir as oportunidades de uso de ferramentas de automação de teste	2												X

12 Apêndice C: Notas de versão

O ISTQB® Test Automation Engineering Syllabus 2024 é uma grande atualização e reescrita da versão 2016. Por esse motivo, não há notas de versão detalhadas por capítulo e seção. O conteúdo foi significativamente aprimorado para a função de Engenharia de Automação, enquanto o conteúdo estratégico foi movido para o novo ISTQB® Test Automation Strategy Syllabus 2024.

13 Apêndice D - Termos específicos do domínio

DevSecOps: Uma metodologia que combina desenvolvimento, segurança e operações e usa um elevado grau de automação.

padrão do modelo de fluxo: Uma visão de alto nível do domínio de trabalho, seus componentes e interconexões entre eles.

jornada do usuário: Uma série de etapas que mostram, a partir da perspectiva da experiência da pessoa, como um usuário interage com um sistema em teste.

padrão de objeto de página: Um padrão de design na automação de teste para aprimorar a manutenção de testes e reduzir a duplicação de código.

controle de versão: Um processo para fazer o check-in e armazenar versões específicas do código-fonte.