Certified Tester Syllabus

Test Automation Engineer

CT-TAE 1.0

BSTQB
Brazilian Software Testing
Qualifications Board

Versão traduzida, adaptada e disponibilizada na Língua Portuguesa
pelo **WG-Traduções do BSTQB** do syllabus
ISTQB Certified Tester (Advanced Level), Test Automation Engineer
versão 1.0 (2016)

Direitos Autorais

Advanced Level Test Automation Working Group: Bryan Bakker, Graham Bath, Armin Born, Mark Fewster, Jani Haukinen, Judy McKay, Andrew Pollner, Raluca Popescu, Ina Schieferdecker; 2016.

Histórico de Revisões

Versão	Data	Observação
Esboço inicial	13 de agosto de 2015	Esboço inicial
Segundo esboço	5 de novembro de 2015	Mapeamento e reposicionamento de LO
Terceiro esboço	17 de dezembro de 2015	Refinamento de LO
Esboço Beta	11 de janeiro de 2016	Esboço editado
Beta	18 de março de 2016	Versão Beta
Syllabus 2016	21 de outubro de 2016	Versão GA

Sumário

Agradecimentos

Este documento foi produzido por uma equipe do *Advanced Level Working Group* do *International Software Testing Qualifications Board*.

A equipe agradece à equipe de revisão e a todas os Conselhos Nacionais por suas sugestões.

No momento em que o nível avançado do *syllabus* para este módulo foi terminado, o *Advanced Level Working Group* para automatização do teste teve os seguintes membros: Bryan Bakker, Graham Bath (Advanced Level Working Group Chair), Armin Beer, Inga Birthe, Armin Born, Alessandro Collino, Massimo Di Carlo, Mark Fewster, Mieke Gevers, Jani Haukinen, Skule Johansen, Eli Margolin, Judy McKay (Advanced Level Working Group Vice Chair), Kateryna Nesmyelova, Mahantesh (Monty) Pattan, Andrew Pollner (Advanced Level Test Automation Chair), Raluca Popescu, Ioana Prundaru, Riccardo Rosci, Ina Schieferdecker, Gil Shekel, Chris Van Bael.

Os autores da equipe principal deste plano de estudos: Andrew Pollner (Chair), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker.

As seguintes pessoas participaram da revisão, comentários e votações deste syllabus (ordem alfabética): Armin Beer, Tibor Csöndes, Massimo Di Carlo, Chen Geng, Cheryl George, Kari Kakkonen, Jen Leger, Singh Manku, Ana Paiva, Raluca Popescu, Meile Posthuma, Darshan Preet, Ioana Prundaru, Stephanie Ulrich, Erik van Veenendaal, Rahul Verma.

O BSTQB agradece aos voluntários de seu grupo de trabalho de traduções (**WG Traduções**) pelo empenho e esforço na tradução e revisão deste material

Este documento foi formalmente divulgado pela Assembleia Geral do ISTQB em 21 de outubro de 2016.

0 Introdução ao Syllabus

0.1 Objetivo deste documento

Este *syllabus* constitui a base para a certificação do *ISTQB® Advanced Level* para *Test Automation - Engineering* (Engenharia de Automação de Teste). O ISTQB® fornece este *syllabus* da seguinte forma:

- Para os Conselhos Nacionais, para traduzir para sua língua local e para credenciar os provedores de treinamento. Os Conselhos Nacionais podem adaptar o syllabus às suas necessidades linguísticas específicas e modificar as referências para se adaptarem às suas publicações locais;
- Para os Exames, para derivar questões de exame em sua língua local adaptada aos objetivos de aprendizagem para cada módulo;
- Para os provedores de treinamento, produzir material didático e determinar métodos de ensino apropriados;
- Para os Candidatos à certificação, para se preparar para o exame (como parte de um curso de formação ou de forma independente);
- Para a Comunidade Internacional de Software e Engenharia de Sistemas, para avançar na profissão de software e testes de sistemas e como base para livros e artigos.

O ISTQB® pode permitir que outras entidades utilizem este syllabus para outros fins, desde que procurem e obtenham autorização prévia por escrito.

0.2 Escopo deste documento

Previsto

Este documento descreve as tarefas de um Engenheiro de Automação de Teste (TAE) na concepção, desenvolvimento e manutenção de soluções de automação de teste. Centra-se nos conceitos, métodos, ferramentas e processos para automatizar testes funcionais dinâmicos e a relação desses testes com o gerenciamento de testes, gerenciamento de configurações, gerenciamento de defeitos, processos de desenvolvimento de software e garantia de qualidade.

Os métodos descritos são geralmente aplicáveis a várias abordagens do ciclo de vida do software (p. ex., ágil, sequencial, incremental, iterativo), tipos de sistemas de software (p. ex., incorporados, distribuídos, móveis) e tipos de teste (teste funcional e não funcional).

Não previsto

Os seguintes aspetos estão fora do escopo para este *syllabus*:

- Gerenciamento de testes, criação automatizada de especificações de teste e geração de testes automatizados;
- Tarefas do Gerente de Automação de Teste (TAM) no planejamento, supervisão e ajuste do desenvolvimento e evolução de soluções de automação de teste;
- Especificações da automatização de testes não funcionais (p. ex., desempenho);

- Automatização de análise estática (p. ex., análise de vulnerabilidade) e ferramentas de teste estático;
- Ensino de métodos de engenharia de software e programação (p. ex., quais padrões usar e quais habilidades ter para realizar uma solução de automação de teste);
- Ensino de tecnologias de software (p. ex., quais técnicas de *scripting* usar para implementar uma solução de automação de teste);
- Seleção de produtos e serviços de teste de software (p. ex., quais produtos e serviços usar para uma solução de automação de teste).

0.3 A certificação CT-TAE Test Automation Engineer

Expetativas

A certificação de Nível Avançado destina-se a pessoas que desejam desenvolver os conhecimentos e as competências adquiridas no Nível Fundamental e desenvolver ainda mais os seus conhecimentos numa ou mais áreas específicas. Os módulos oferecidos pelo Especialista em Nível Avançado cobrem uma ampla gama de tópicos de teste.

Um Engenheiro de Automação de Teste é aquele que tem amplo conhecimento de testes em geral e uma compreensão aprofundada na área especial de automação de teste. Uma compreensão profunda é definida como ter conhecimento suficiente de teoria e prática de automação de teste para ser capaz de influenciar a direção que uma organização e/ou projeto leva ao projetar, desenvolver e manter soluções de automação de teste para testes funcionais.

O módulo Visão Geral dos Módulos de Nível Avançado [ISTQB-AL-Modules] descreve os resultados do negócio para este módulo.

Requisitos de Entrada e Renovação

Os critérios gerais de entrada para o Nível Avançado estão descritos no site do ISTQB na Web [ISTQB-Web], ou no site do BSTQB [BSTQB-web] na seção Portfólio.

Além destes critérios gerais de entrada, os candidatos devem possuir o certificado *ISTQB Foundation Level* [ISTQB-CTFL] para se realizar o exame de certificação *Advanced Level Test Automation Engineer*.

Nível de Conhecimento

Os objetivos de aprendizagem para este syllabus são apresentados no início de cada capítulo para uma identificação clara. Cada tópico do syllabus será examinado de acordo com o objetivo de aprendizagem que lhe foi atribuído.

Os níveis cognitivos atribuídos aos objetivos de aprendizagem ("K-levels") são descritos no site do ISTQB [ISTQB-Web].

Exame

O exame para a certificação de Nível Avançado será baseado neste plano de estudos mais o *Syllabus* de Nível Fundamental [ISTQB-CTFL]. As respostas às perguntas de um exame podem exigir o uso de material baseado em mais de um capítulo desse syllabus.

O formato do exame é descrito no site do BSTQB [BSTQB-Web] no capítulo Certificações. Algumas informações úteis para aqueles que fazem exames também estão incluídas no site do BSTQB.

Credenciamento

O Conselho do BSTQB pode credenciar provedores de treinamento cujos materiais de curso seguem este syllabus.

No site do BSTQB [BSTQB-Web], o capítulo Treinamento descreve as regras específicas que se aplicam aos provedores de treinamento para a acreditação de cursos e a relação de material e provedores já credenciados.

0.4 Partes normativas versus informativas

As partes normativas do *syllabus* são examináveis. Esses são:

- Palavras-chave.
- Objetivos de aprendizado.

O restante do *syllabus* é informativo e elabora sobre os objetivos de aprendizagem.

0.5 Nível de detalhe

O nível de detalhe deste syllabus permite o ensino e um exame internacionalmente consistentes. Para atingir esse objetivo, o syllabus trata:

- Objetivos de aprendizagem para cada área de conhecimento, descrevendo o resultado cognitivo e o conhecimento a ser alcançado (estes são normativos);
- Uma lista de informações para ensinar, incluindo uma descrição dos Palavras-chave, fontes como literatura ou padrões aceitos e referências a fontes adicionais se necessário (estas são informativas).

O conteúdo do syllabus não é uma descrição de toda a área de conhecimento da Engenharia de Automação de Teste, ele reflete pormenor a ser coberto em um curso de formação de nível avançado credenciado.

0.6 Como esse syllabus é organizado

Há oito capítulos principais. No título principal é apresentada a quantidade de horas que devem ser dedicadas ao capítulo entre colchetes.

Por exemplo:

Arquitetura de Automação de Teste Genérica *[270 min]*

Neste exemplo é definido um tempo mínimo de dedicação ao ensino de 270 minutos. Os objetivos de aprendizagem específicos são listados no início de cada capítulo.

0.7 Termos, definições e acrônimos

Muitos termos usados na literatura de software são intercambiáveis. As definições para este *syllabus* estão disponíveis no Glossário Termos, criado pelo ISTQB [ISTQB-Glossary], traduzido e publicado pelo [BSTQB-Web].

Cada um dos Palavras-chave listados no início de cada capítulo está definido no [ISTQB-Glossary].

Os acrônimos a seguir são usados neste documento:

CLI: *Command Line Interface*
EMTE: *Equivalent Manual Test Effort*
gTAA: *Generic Test Automation Architeture*,
Fornecendo um modelo para soluções de automação de teste.
GUI: *Graphical User Interface*
Interface gráfica de um software.
SUT: *System Under Test*
Sistema sob teste.
TAA: *Test Automation Architeture*
Instancia de gTAA para definir a arquitetura de um TAS.
TAE: *Test Automation Engineer*
Pessoa que é responsável pela concepção de um TAA, incluindo a implementação do TAS resultante, a sua manutenção e evolução técnica.
TAF: *Test Automation Framework*
Ambiente necessário para a automação de teste.
TAM: *Test Automation Manager*
Pessoa responsável pelo planejamento e supervisão do desenvolvimento e evolução de um TAS.
TAS: *Test Automation Solution*
Realização ou implementação de um TAA, artefatos como bibliotecas de teste.
UI: *User Interface*
Interface do usuário.

1 Introdução e objetivos da automação de testes [30 min]

Palavras-chave

testes de API, testes de CLI, teste de GUI, sistema sob teste, arquitetura de automação de teste, estrutura de automação de teste, estratégia de automação de teste, automação de teste, script de teste, *testware*.

Objetivos de Aprendizagem

1.1 Propósito da automação de teste

ALTA-E-1.1.1 (K2): Explicar os objetivos, vantagens, desvantagens e limitações da automação de teste

1.2 Fatores de sucesso na automação de teste

ALTA-E-1.2.1 (K2): Identificar fatores técnicos de sucesso de um projeto de automação de teste

1.1 Propósito da automação de teste

Nos testes de software, a automação de teste (que inclui a execução de teste automatizada) é uma ou mais das seguintes tarefas:

- Utilizar ferramentas de software construídas especificamente para controlar e configurar pré-condições de teste;
- Executar testes;
- Comparar os resultados reais com os resultados previstos.

Uma boa prática é separar o software usado para testar o próprio Sistema em Teste (SUT) para minimizar a interferência. Existem exceções, por exemplo, sistemas embutidos nos quais o software de teste precisa ser implantado no SUT.

É esperado da automação de teste a ajuda para executar muitos casos de teste de forma consistente e repetidamente em diferentes versões do SUT e/ou ambientes. Mas a automação de teste é mais do que um mecanismo para executar um conjunto de testes sem interação humana. Envolve um processo de criação do *testware*, incluindo:

- Software;
- Documentação;
- Casos de teste;
- Ambientes de teste;
- Dados de teste.

O *Testware* é necessário para as atividades de teste que incluem:

- Implementação de casos de teste automatizados;
- Monitoramento e controle da execução de testes automatizados;
- Interpretação, reporte e registro dos resultados dos testes automatizados.

A automação de teste tem abordagens diferentes para interagir com um SUT:

- Testar através das interfaces públicas para classes, módulos ou bibliotecas do SUT (Teste de API);
- Testar através da interface de usuário do SUT (p. ex., teste de GUI ou CLI);
- Testes através de um serviço ou protocolo.

Os objetivos da automação de teste incluem:

- Melhorar a eficiência do teste;
- Proporcionar uma cobertura de funções mais ampla;
- Reduzir o custo total do teste;
- Realizar os testes que os testadores manuais não podem;
- Reduzir o período de execução do teste;
- Aumentar a frequência de teste a partir da redução do tempo necessário nos ciclos de teste.

As vantagens da automação de teste incluem:

- Mais testes podem ser executados por compilação;

- A possibilidade de criar testes que não podem ser feitos manualmente (em tempo real, remoto, testes paralelos);
- Aumentar a complexidade dos testes;
- Aumentar a velocidade de execução dos testes;
- Reduzir os erros humanos na execução dos testes;
- Elevar a eficácia e eficiência dos recursos de teste;
- Feedback mais rápido sobre a qualidade do software;
- Maior confiabilidade do sistema (p. ex., repetibilidade, consistência);
- Maior consistência dos testes.

Desvantagens da automação de teste incluem:

- Custos adicionais envolvidos;
- Investimento inicial para configurar o TAS;
- Necessita de tecnologias adicionais;
- A equipe precisa ter habilidades de desenvolvimento e automação;
- É requerida a manutenção do TAS em andamento;
- Pode desviar-se dos objetivos de teste, por exemplo, concentrando-se na automatização de casos de testes às custas da execução dos testes;
- Os testes podem se tornar mais complexos;
- Erros adicionais podem ser introduzidos pela automação.

As limitações da automação de teste incluem:

- Nem todos os testes manuais podem ser automatizados;
- A automação só pode verificar os resultados interpretáveis pela máquina;
- A automação só pode verificar os resultados reais que podem ser verificados por um teste de oráculo automatizado;
- Não é um substituto para testes exploratórios.

1.2 Fatores de sucesso na automação de teste

Os seguintes fatores de sucesso se aplicam aos projetos de automação de teste que estão em operação e, portanto, o foco está nas influências que afetam o sucesso a longo prazo do projeto. Fatores que influenciam o sucesso de projetos de automação de teste na fase piloto não são aqui considerados.

Os principais fatores de sucesso para a automação de teste incluem:

Test Automation Architeture (TAA) - Arquitetura do Automação de Teste

A arquitetura de automação de teste (TAA) está muito alinhada com a arquitetura de um produto de software. Deve ficar claro quais requisitos funcionais e não funcionais a arquitetura deve suportar. Normalmente, este será o mais importante requisito.

Muitas vezes a TAA é projetada para manutenção, desempenho e capacidade de aprendizagem. (Veja ISO/IEC 25000: 2014 para detalhes sobre essas e outras caraterísticas não-funcionais.) É útil envolver engenheiros de software que entendam a arquitetura do SUT.

Testabilidade do SUT

O SUT precisa ser modelado para testabilidade suportando testes automatizados. No caso de testes de GUI, isso pode significar que o SUT deve desacoplar tanto quanto possível a interação GUI e os dados da aparência da interface gráfica. No caso de testes de API, isso pode significar que mais classes, módulos ou a interface de linha de comando precisam ser expostos como público para que possam ser testados.

As partes testáveis do SUT devem ser alvo primeiro. Geralmente, um fator chave no sucesso da automação de teste reside na facilidade de implementar scripts de teste automatizados. Com este objetivo em mente, e para fornecer uma prova de conceito bem-sucedida, o TAE (*Test Automation Engineer*) precisa iniciar a partir da identificação dos módulos ou componentes do SUT que são facilmente testados com automação.

Estratégia de Automação de teste

É uma estratégia prática e consistente de automação de teste que aborda a manutenção e consistência do SUT.

Pode não ser possível aplicar a estratégia de automação de teste da mesma forma a partes antigas e novas do SUT. Ao criar a estratégia de automação, considere os custos, benefícios e riscos de aplicá-lo nas diferentes partes do código.

Deve-se considerar o teste da interface do usuário e da API com casos de teste automatizados para verificar a consistência dos resultados.

Framework de Automação de Teste (TAF - Test Automation Framework)

Um TAF que é fácil de usar, bem documentado e mantido, suporta uma abordagem consistente para automatizar testes.

A fim de estabelecer um TAF fácil de usar e de manutenção, o seguinte deve ser feito:

- *Implementar instalações de reporte*: Os relatórios de teste devem fornecer informações sobre a qualidade do SUT (aprovação, reprovação, falha, erro, não processado, abortado, estatísticas etc.). Os relatórios devem fornecer as informações para os testadores envolvidos, gerentes de teste, desenvolvedores, gerentes de projeto e outros stakeholders para obter uma visão geral da qualidade.
- *Ativar facilmente a solução de problemas*: Além da execução de teste e registro, o TAF deve fornecer uma maneira fácil de solucionar falhas de testes. O fracasso pode ser devido as falhas encontradas no SUT ou no TAS, com os próprios testes ou no ambiente de teste.
- *Abordar adequadamente o ambiente de teste*: As ferramentas de teste dependem da consistência no ambiente. Ter um ambiente de teste dedicado é necessário em testes automatizados. Se não houver controle do ambiente e de seus dados, a configuração para os testes poderá não atender aos requisitos de execução, e é provável que produza resultados falsos.
- *Documentar os casos de teste automatizados*: Os objetivos da automação de teste devem ser claros, por exemplo, quais partes da aplicação serão testadas, até que ponto e quais

atributos serão testados (funcionais e não funcionais). Isso deve ser claramente descrito e documentado.
- *Rastrar o teste automatizado*: O TAF deve suportar o rastreamento para que o Engenheiro de Automação de Teste rastreie etapas individuais dos casos de teste.
- *Permitir uma fácil manutenção*: Idealmente, os casos de teste automatizados devem ser armazenados para que a manutenção não consuma uma parte significativa do esforço de automação de teste. Além disso, o esforço de manutenção deve ser proporcional à escala das mudanças feitas no SUT. Para isso, os casos devem ser facilmente analisáveis, mutáveis e expansíveis. Além disso, a reutilização automatizada de testes deve ser alta para minimizar o número de itens que exigem alterações.
- *Manter os testes automatizados atualizados*: Quando os requisitos novos ou alterados fazem com que os testes ou conjuntos de testes inteiros falhem, não os desative, corrija-os.
- *Planejar a implantação*: Certifique-se de que os scripts de teste podem ser facilmente implantados, alterados e reimplantados.
- *Retirar os testes conforme necessário*: Certifique-se de que os scripts de teste automatizados podem ser facilmente retirados se não forem mais úteis ou necessários.
- *Monitorar e restaurar o SUT*: Na prática, para executar continuamente um caso de teste ou conjunto de casos de teste, o SUT deve ser monitorado continuamente. Se o SUT encontrar um erro fatal (como uma falha), o TAF deve ter a capacidade de recuperar, ignorar o caso atual e continuar os testes com o próximo caso.

O código de automação de teste pode ser complexo de se manter. Não é incomum ter tanto código para testar quanto ao código do SUT. É por isso que é de extrema importância que o código de teste possa ser armazenado. Isto é devido às diferentes ferramentas de teste que estão sendo usadas, aos diferentes tipos de verificação que são usados e aos diferentes artefatos de teste que precisam ser mantidos (como dados de entrada de teste, oráculos de teste, relatórios de teste).

Com estas considerações de manutenção em mente, além dos itens importantes que devem ser feitos, há outros que devem ser desprezados, da seguinte forma:

- Não crie código que seja sensível à interface (isto é, seria afetado por mudanças na interface gráfica ou em partes não essenciais da API).
- Não crie automação de teste que seja sensível a alterações de dados ou tenha uma alta dependência de valores de dados específicos (p. ex., entrada de teste dependendo de outras saídas de teste).
- Não crie um ambiente de automação que seja sensível ao contexto (p. ex., data e hora do sistema operacional, parâmetros de localização do sistema operacional ou o conteúdo de outro aplicativo). Neste caso, é melhor usar simuladores de teste conforme necessário para que o ambiente possa ser controlado.

Quanto mais fatores de sucesso forem atendidos, mais provável será o sucesso do projeto de automação de teste. Nem todos os fatores são necessários, e na prática raramente todos os fatores são atendidos. Antes de iniciar o projeto de automação de teste, é importante analisar as chances de sucesso para o projeto, considerando os fatores existentes e os fatores que faltam, mantendo os riscos da abordagem escolhida, bem como o contexto do projeto. Uma vez que a TAA está no lugar, é importante investigar quais itens estão faltando ou ainda precisam de trabalho.

2 Preparando-se para a automação de teste [165 min]

Palavras-chave:

testabilidade, controlador, nível de intrusão, simulador, ferramenta de execução de teste, teste de gancho, gerente de automação de teste.

Objetivos de Aprendizagem:

2.1 Fatores do SUT que influenciam a automação de teste

ALTA-E-2.1.1 (K4) Analisar um sistema em teste para determinar a solução apropriada de automação.

2.2 Avaliação e seleção de ferramentas

ALTA-E-2.2.1 (K4) Analisar as ferramentas de automação de teste para um determinado projeto e relatar resultados técnicos e recomendações

2.3 Modelagem para testabilidade e automação

ALTA-E-2.3.1 (K2) Compreender os métodos "modelagem para testabilidade " e "modelagem para automação de teste" aplicáveis ao SUT.

2.1 Fatores do SUT que influenciam a automação de teste

Ao avaliar o contexto do SUT e seu ambiente, os fatores que influenciam a automação de teste precisam ser identificados para determinar uma solução apropriada. Estes podem incluir o seguinte:

Interfaces do SUT

Os casos de teste automatizados invocam ações no SUT. Para isso, o SUT deve fornecer interfaces através das quais o SUT pode ser controlado. Isso pode ser feito através de controles de interface gráfica (GUI), mas também através de interfaces de software de nível inferior. Além disso, alguns casos de teste podem ser capazes de interagir no nível de comunicação, por exemplo, usando TCP/IP, USB ou interfaces de mensagens proprietárias.

A decomposição do SUT permite que a automação de teste interfira com o SUT em diferentes níveis de teste. É possível automatizar os testes em um nível específico, por exemplo o nível do componente e do sistema, mas somente quando o SUT apoia isso adequadamente. Por exemplo, no nível do componente, pode não haver interface do usuário que possa ser usada no teste, assim precisarão estar disponíveis interfaces de software diferenciadas, possivelmente personalizadas, também chamadas ganchos de teste.

Software de terceiros

Muitas vezes, o SUT não consiste apenas em software escrito na organização doméstica, mas também pode incluir software fornecido por terceiros. Em alguns contextos, este software pode necessitar de testes e, se a automação de teste for justificada, ela pode precisar de uma solução de automação de teste diferenciada, como o uso de uma API.

Níveis de intrusão

Diferentes abordagens de automação de teste (usando diferentes ferramentas) têm diferentes níveis de intrusão. Quanto maior o número de alterações que são necessárias serem construídas especificamente para testes automatizados para o SUT, maior será o nível de intrusão. O uso de interfaces de software dedicadas requer um alto nível de intrusão, enquanto o uso de elementos de interface do usuário existentes tem um nível de intrusão menor. Usando elementos de hardware do SUT (como teclados, *hand-switches*, *touchscreens*, interfaces de comunicação) têm um nível ainda maior de intrusão.

O problema com níveis mais altos de intrusão é o risco de alarmes falsos. O TAS pode apresentar falhas que podem ser decorrentes do nível de intrusão imposta pelos testes, mas isso não é provável que aconteça quando o sistema de software está sendo usado em um ambiente real. Testar com um alto nível de intrusão é geralmente uma solução mais simples para a abordagem de automação de teste.

Arquiteturas diferentes do SUT

Diferentes arquiteturas do SUT podem exigir diferentes soluções de automação de teste. Uma abordagem diferente é necessária para um SUT escrito em C++ usando a tecnologia COM do

que para um SUT escrito em Python. Pode ser possível que essas diferentes arquiteturas sejam tratadas pela mesma estratégia de automação de teste, mas isso requer uma estratégia híbrida com capacidade de suportá-las.

Tamanho e complexidade do SUT

Considerar o tamanho e a complexidade do SUT atual e os planos para o desenvolvimento futuro. Para um SUT pequeno e simples, uma abordagem de automação de teste complexa e flexível pode não ser garantida. Uma abordagem simples pode ser mais adequada. Inversamente, pode não ser racional implementar uma abordagem pequena e simples para um SUT muito grande e complexo. Às vezes, porém, é apropriado começar pequeno e simples, mesmo para um SUT complexo, mas isso deve ser uma abordagem temporária (veja o Capítulo 3 para mais detalhes).

Vários fatores aqui descritos são conhecidos (p. ex., tamanho e complexidade, interfaces de software disponíveis) quando o SUT já está disponível, mas a maior parte do tempo o desenvolvimento da automação de teste deve ser iniciado antes do SUT estar disponível. Quando isso acontece, várias coisas precisam ser estimadas ou o TAE pode especificar as interfaces de software necessárias. (Ver Capítulo 2.3 para mais detalhes).

Mesmo quando o SUT ainda não existe, o planejamento de automação de teste pode ser iniciado. Por exemplo:

- Quando os requisitos (funcionais ou não funcionais) são conhecidos, os candidatos à automação podem ser selecionados a partir desses requisitos, juntamente com a identificação dos meios para testá-los. O planejamento para a automação pode começar para esses candidatos, incluindo também a identificação dos requisitos para a automação e a determinação da melhor estratégia de automação dos testes.
- Quando a arquitetura e a modelagem técnica estiverem sendo desenvolvidas, o projeto de interfaces de software para suportar o teste pode ser realizado.

2.2 Avaliação e seleção de ferramentas

A responsabilidade primária pelo processo de seleção e avaliação da ferramenta pertence ao TAM (gerente de automação de teste). No entanto, o TAE estará envolvido no fornecimento de informações ao TAM e na realização de muitas das atividades de avaliação e seleção. O conceito do processo de avaliação e seleção de ferramentas foi introduzido no Nível Fundamental (CTFL) e mais detalhes desse processo são descritos no *syllabus* CTAL-TM [ISTQB-CTAL-TM].

O TAE estará envolvido ao longo do processo de avaliação e seleção de ferramentas, mas terá contribuições específicas para as seguintes atividades:

- Avaliar a maturidade da organização e identificar as oportunidades de suporte da ferramenta;
- Avaliar os objetivos apropriados para o suporte da ferramenta;
- Identificar e recolher informações sobre ferramentas potencialmente adequadas;
- Analisar as informações sobre a ferramenta em relação aos objetivos e restrições do projeto;
- Estimar a relação custo-benefício com base em um *business case*;

- Fazer uma recomendação sobre a ferramenta apropriada;
- Identificar a compatibilidade da ferramenta com componentes do SUT.

As ferramentas de automação de teste funcional frequentemente não atendem a todas as expetativas ou as situações que são encontradas em um projeto de automação. Um conjunto de exemplos desses tipos de problemas são (não é uma lista completa):

Encontrado	Exemplos	Soluções Possíveis
A interface da ferramenta não funciona com outras ferramentas já existentes	As informações do suporte de pré-vendas estavam erradas e nem todos os dados podem ser transferidos para a ferramenta de geração de relatórios	Preste atenção às notas de versão antes de qualquer atualização e para grandes migrações teste antes de migrar para a produção. Tente obter uma demonstração no local da ferramenta que usa o real SUT. Procurar suporte de fornecedores e/ou fóruns da comunidade de usuários
Algumas dependências SUT são alteradas para aquelas não suportadas pela ferramenta de teste	O departamento de desenvolvimento atualizou para a versão mais recente do Java	Sincronizar atualizações para o ambiente de desenvolvimento/teste e a ferramenta de automação de teste
Objeto na GUI não pôde ser capturado	O objeto é visível, mas a ferramenta de automação de teste não pode interagir com ele	Tente usar apenas tecnologias conhecidas ou objetos em desenvolvimento Faça um projeto piloto antes de comprar uma ferramenta de automação de teste Os desenvolvedores definem padrões para objetos
A ferramenta parece muito complicada	A ferramenta tem um conjunto de recursos enorme, mas apenas parte do que será usado	Tente encontrar uma forma de limitar o conjunto de recursos removendo recursos indesejados da barra de ferramentas Selecione uma licença para atender às suas necessidades. Tente encontrar ferramentas alternativas mais focadas na funcionalidade necessária.
Conflito com outros sistemas	Após a instalação de outro software a ferramenta de automação de teste não funcionará mais ou vice-versa	Leia as notas de versão ou os requisitos técnicos antes de instalar. Obter confirmação do fornecedor de que não haverá impacto para outras ferramentas. Questionar fóruns de comunidade de usuários.
Impacto sobre o SUT	Durante / após o uso da ferramenta de automação de teste, o SUT reage de forma diferente (p. ex., tempo de resposta mais longo)	Use uma ferramenta que não precise alterar o SUT (p. ex., instalação de bibliotecas etc.)
Acesso ao código	A ferramenta de automação de teste irá alterar partes do código-fonte	Use uma ferramenta que não precisará alterar o código-fonte (p. ex., instalação de bibliotecas etc.)

Encontrado	Exemplos	Soluções Possíveis
Recursos limitados (principalmente em ambientes incorporados)	O ambiente de teste tem recursos livres limitados ou fica sem recursos (p. ex., memória)	Leia as notas de lançamento e discuta o ambiente com o fornecedor da ferramenta para obter confirmação de que isso não levará a problemas. Questionar fóruns de comunidade de usuários.
Atualizações	A atualização não migrará todos os dados ou corromperá os scripts, dados ou configurações de teste automatizados existentes A atualização precisa de um ambiente (melhor)	Teste a atualização no ambiente de teste e obtenha confirmação do provedor de que a migração funcionará Ler pré-requisitos de atualização e decidir se a atualização vale o esforço Procure o suporte dos fóruns da comunidade de usuários
Segurança	Ferramenta de automação de teste requer informações que não estão disponíveis para o Engenheiro de Automação de Teste	O Engenheiro de Automação de Teste precisa ter acesso
Incompatibilidade entre diferentes ambientes e plataformas	A automação de teste não funciona em todos os ambientes / plataformas	Implementar testes automatizados para maximizar a independência da ferramenta, minimizando assim o custo do uso de várias ferramentas.

2.3 Modelagem para testabilidade e automação

A testabilidade do SUT (a disponibilidade de interfaces de software que suportam testes, por exemplo, para permitir o controle e observação do SUT) deve ser concebida e implementada em paralelo com a concepção e implementação de outras caraterísticas do SUT. Isso pode ser feito pelo analista de desenvolvimento (como testabilidade é apenas um dos requisitos não-funcionais do sistema), mas muitas vezes isso é feito por, ou com o envolvimento de um TAE.

A modelagem para testabilidade consiste em várias partes:

- **Observabilidade**: O SUT precisa fornecer interfaces que dão uma visão do sistema. Os casos de teste podem então usar essas interfaces para verificar, por exemplo, se o comportamento esperado é igual ao comportamento real.
- **Controle** (capacidade): O SUT precisa fornecer interfaces que podem ser usadas para executar ações no SUT. Isto pode ser: elementos UI, chamadas de função, elementos de comunicação (p. ex., protocolo TCP/IP ou USB), sinais eletrônicos (para comutadores físicos), entre outros.
- **Arquitetura claramente definida**: A terceira parte importante da modelagem para testabilidade é uma arquitetura que fornece interfaces claras e compreensíveis que dão controle e visibilidade em todos os níveis do teste.

O TAE considera maneiras pelas quais o SUT pode ser testado, incluindo testes automatizados, de forma eficaz (testando as áreas certas e encontrando bugs críticos) e eficiente (sem tomar muito esforço). Sempre que interfaces de software específicas são necessárias, elas devem ser

especificadas pelo TAE e implementadas pelo desenvolvedor. É importante definir testabilidade e, se necessário, interfaces de software adicionais no início do projeto, para que o trabalho de desenvolvimento possa ser planejado e orçado.

Alguns exemplos de interfaces de software que suportam testes incluem:

- A poderosa capacidade de gerar script usando as modernas planilhas;
- Aplicar simuladores ou *mocks* para simular software e/ou hardware (p. ex., transações financeiras eletrônicas, serviço de software, servidor dedicado, placa eletrônica, peça mecânica) que ainda não está disponível ou é muito caro para comprar, permite testar o software na ausência dessa interface específica;
- As interfaces de software (ou simuladores e controladores) podem ser usadas para testar condições de erro. Considere um dispositivo com uma unidade de disco rígido interna (HD). O software que controla este HD (chamada controladora de HD ou *Hard Disk Driver*) deve ser testado para detectar falhas ou desgaste no disco. Fazer isso esperando pela falha da controladora não é muito eficiente (ou confiável). A implementação de interfaces de software que simulam um disco defeituoso ou lento pode verificar se o software da controladora executa corretamente suas instruções (p. ex., fornece uma mensagem de erro, número de tentativas);
- Interfaces de software alternativas podem ser usadas para testar um SUT quando nenhuma interface do usuário (UI) está disponível ainda (e isso é muitas vezes considerado uma abordagem melhor de qualquer maneira). O software embutido em sistemas muitas vezes precisa monitorar a temperatura do dispositivo e acionar uma função de resfriamento quando a temperatura sobe acima de um determinado nível. Isso pode ser testado sem o hardware usando uma interface de software para especificar a temperatura;
- O teste de transição de estado é usado para avaliar o comportamento do estado do SUT. Uma maneira de verificar se o SUT está no estado correto é consultá-lo através de uma interface de software personalizada projetada para esse fim (embora isso também inclua um risco, consulte o nível de intrusão no Capítulo 2.1).

O projeto para automação deve considerar que:

- A compatibilidade com as ferramentas de teste existentes deve ser estabelecida desde o início;
- A questão da compatibilidade da ferramenta de teste é crítica, pois pode afetar a capacidade de automatizar testes de funcionalidade importantes (p. ex., a incompatibilidade com um controlador de grade impedindo todos os testes que usam este controlador);
- Soluções podem exigir o desenvolvimento de código de programa e chamadas para API.

A modelagem para testabilidade é de extrema importância para uma boa abordagem de automação de teste e pode beneficiar a execução de teste manual.

3 Arquitetura genérica de automação do teste [270 min]

Palavras-chave

captura e reprodução, testes de dados, arquitetura de teste genérica de automação, teste orientado por palavras-chave, scripts lineares, teste baseado em modelo, scripts de processo, scripts estruturados, camada de adaptação de teste, arquitetura de automação de teste, estrutura de automação de teste, camada de definição de teste, camada de execução de teste, camada de geração de teste

Objetivos de Aprendizagem

3.1 Introdução ao gTAA

ALTA-E-3.1.1 (K2): Explicar a estrutura do gTAA.

3.2 Modelagem TAA

ALTA-E-3.2.1 (K4): Projetar a TAA apropriada para um determinado projeto.

ALTA-E-3.2.2 (K2): Explicar o papel que as camadas desempenham dentro de um TAA.

ALTA-E-3.2.3 (K2): Compreender as considerações de projeto para um TAA.

ALTA-E-3.2.4 (K4): Analisar os fatores de implementação, uso e manutenção para um dado TAS.

3.3 Desenvolvimento TAS

ALTA-E-3.3.1 (K3): Aplicar componentes da TAA genérica (gTAA) para construir uma TAA.

ALTA-E-3.3.2 (K2): Explicar os fatores a serem considerados ao identificar a reutilização de componentes.

3.1 Introdução ao gTAA

Um Engenheiro de Automação de Teste (TAE) tem o papel de projetar, desenvolver, implementar e manter Soluções de Automação de Teste (TAS). À medida que cada solução é desenvolvida, tarefas semelhantes precisam ser feitas, questões semelhantes precisam ser respondidas, tratadas e priorizadas. Esses conceitos, etapas e abordagens recorrentes na automação de testes tornam-se a base da Arquitetura Genérica de Automação de Teste, denominada gTAA.

O gTAA apresenta as camadas, componentes e interfaces que são redefinidos na TAA para um TAS particular. Ele permite uma abordagem estruturada e modular para construir uma solução de automação de teste:

- Definindo o espaço conceitual, as camadas, os serviços e as interfaces de um TAS para permitir a realização por componentes internos, bem como por componentes desenvolvidos externamente;
- Suportando componentes simplificados para o desenvolvimento eficaz e eficiente da automação dos testes;
- Reutilizando componentes de automação de teste para diferentes TAS ou em evolução para linhas de produtos de famílias de software, além de tecnologias e ferramentas;
- Facilitando a manutenção e evolução de um TAS;
- Definindo os recursos essenciais para um usuário de um TAS.

Um TAS consiste no ambiente de teste (e seus artefatos) e nos conjuntos de testes (um conjunto de casos de teste incluindo dados de teste). Um *framework* de automação de teste (TAF) pode ser usado para realizar um TAS. Ele fornece suporte para ferramentas de teste de ambiente, ambiente preparado para teste, ou bibliotecas de suporte.

Recomenda-se que a TAA de um TAS cumpra com os seguintes princípios que suportam fácil desenvolvimento, evolução e manutenção do TAS:

- *Responsabilidade Única*: Cada componente do TAS deve ter uma única responsabilidade, e essa responsabilidade deve ser totalmente encapsulada no componente. Em outras palavras, cada componente de um TAS deve ser responsável por cumprir um único objetivo, por exemplo, gerar palavras-chave ou dados, criar cenários de teste, executar casos de teste, registrar resultados, gerar relatórios de execução.
- *Extensão* (ver, p. ex., princípio aberto/fechado por B. Myer): Cada componente TAS deve estar aberto para ampliação, mas fechado para modificação. Este princípio significa que deve ser possível modificar ou enriquecer o comportamento dos componentes sem quebrar a funcionalidade compatível com versões anteriores.
- *Substituição* (ver, p. ex., princípio de substituição por B. Liskov): Cada componente TAS deve ser substituível sem afetar o comportamento geral do TAS. O componente pode ser substituído por um ou mais componentes, mas o comportamento exibido deve ser o mesmo.
- *Segregação de componentes* (ver, p. ex., princípio de segregação de interfaces por R.C. Martin): É melhor ter componentes mais específicos do que um componente geral multiuso. Isso facilita a substituição e a manutenção eliminando dependências desnecessárias.

- *Inversão de dependência*: Os componentes de um TAS devem depender de abstrações e não de detalhes de baixo nível. Em outras palavras, os componentes não devem depender de cenários de teste automatizados específicos.

Normalmente, um TAS baseado no gTAA será implementado por um conjunto de ferramentas, seus *plugins* e/ou seus componentes. É importante notar que o gTAA é neutro ao fornecedor: não predefine nenhum método, tecnologia ou ferramenta concreta para a realização de um TAS. O gTAA pode ser implementado por qualquer abordagem de engenharia de software, por exemplo, estruturada, orientada por objetos, orientada para serviços, orientada por modelo, bem como por quaisquer tecnologias e ferramentas de software. De fato, um TAS é muitas vezes implementado usando ferramentas de prateleira, mas normalmente precisará de adicionais SUT, de adições específicas ou adaptações.

Outras diretrizes e modelos de referência relacionados ao TAS são padrões de engenharia de software para o ciclo de vida de desenvolvimento de software selecionado, tecnologias de programação, padrões de formatação etc. Não pertence ao âmbito deste *syllabus* ensinar engenharia de software em geral, no entanto, de um TAE é esperado ter habilidades, experiências e especializações em engenharia de software.

Além disso, um TAE precisa estar ciente dos padrões de codificação e documentação da indústria e das melhores práticas para usá-los ao desenvolver um TAS. Essas práticas podem aumentar a manutenção, confiabilidade e segurança do TAS. Tais padrões são tipicamente específicos do domínio.

Os padrões populares incluem:

- MISRA para C ou C++;
- Padrão de codificação JSF para C ++;
- Regras AUTOSAR para MathWorks Matlab / Simulink®.

3.1.1 Visão geral do gTAA

O gTAA é estruturado em camadas horizontais:

- Geração de teste;
- Definição de teste;
- Execução de teste;
- Adaptação de teste.

O gTAA abrange:

- A camada de geração de teste que suporta a modelagem manual ou automatizado de casos de teste. Ele fornece os meios para projetar casos de teste;
- A camada de definição de teste que suporta a definição e implementação de conjuntos de teste e/ou casos de teste. Ele separa a definição do teste das tecnologias e ferramentas do SUT e/ou do sistema de teste. Contém meios para definir testes de alto nível e de baixo nível, que são tratados nos dados de teste, casos de teste, procedimentos de teste e componentes de biblioteca de teste ou suas combinações;

- A camada de execução de teste que suporta a execução de casos de teste e o log de teste. Ele fornece uma ferramenta de execução de teste para executar os testes selecionados automaticamente e um componente de log e relatório;
- A camada de adaptação de teste que fornece o código necessário para adaptar os testes automatizados para os vários componentes ou interfaces do SUT. Ele fornece diferentes adaptadores para conexão com o SUT via API, protocolos, serviços e outros;
- Possui também interfaces para gerenciamento de projetos, gerenciamento de configurações e gerenciamento de testes em relação à automação de testes. Por exemplo, a interface entre a gestão de teste e a camada de adaptação de teste lida com a seleção e configuração dos adaptadores apropriados em relação à configuração de teste escolhida.

As interfaces entre as camadas de gTAA e os seus componentes são tipicamente específicas e, portanto, não desenvolvidas aqui.

É importante compreender que estas camadas podem estar presentes ou ausentes em qualquer TAS dado. Por exemplo:

- Se a execução do teste deve ser automatizada, é necessário utilizar a execução do teste e as camadas de adaptação do teste. Eles não precisam ser separados e podem ser realizados juntos, por exemplo, em estruturas de teste de unidade;
- Se a definição de teste deve ser automatizada, a camada de definição de teste é necessária;
- Se a geração de teste for automatizada, a camada de geração de teste é necessária.

Na maioria das vezes, um poderia começar com a implementação de um TAS de baixo para cima, mas outras abordagens, como a geração de teste automatizado para testes manuais também pode ser útil. Em geral, é aconselhável implementar o TAS em passos incrementais (p. ex., em *sprints*), a fim de utilizar o TAS o mais rapidamente possível e para provar o valor acrescentado do TAS. Além disso, são recomendadas provas de conceito como parte do projeto de automação de teste.

Qualquer projeto de automação de teste precisa ser entendido, configurado e gerenciado como um projeto de desenvolvimento de software e requer gerenciamento de projeto dedicado. A gerência de projeto para o desenvolvimento de TAF (isto é, suporte de automação de teste para uma empresa inteira, famílias de produtos ou linhas de produtos) pode ser separada da gerência de projeto para o TAS (isto é, automação de teste para um produto de concreto).

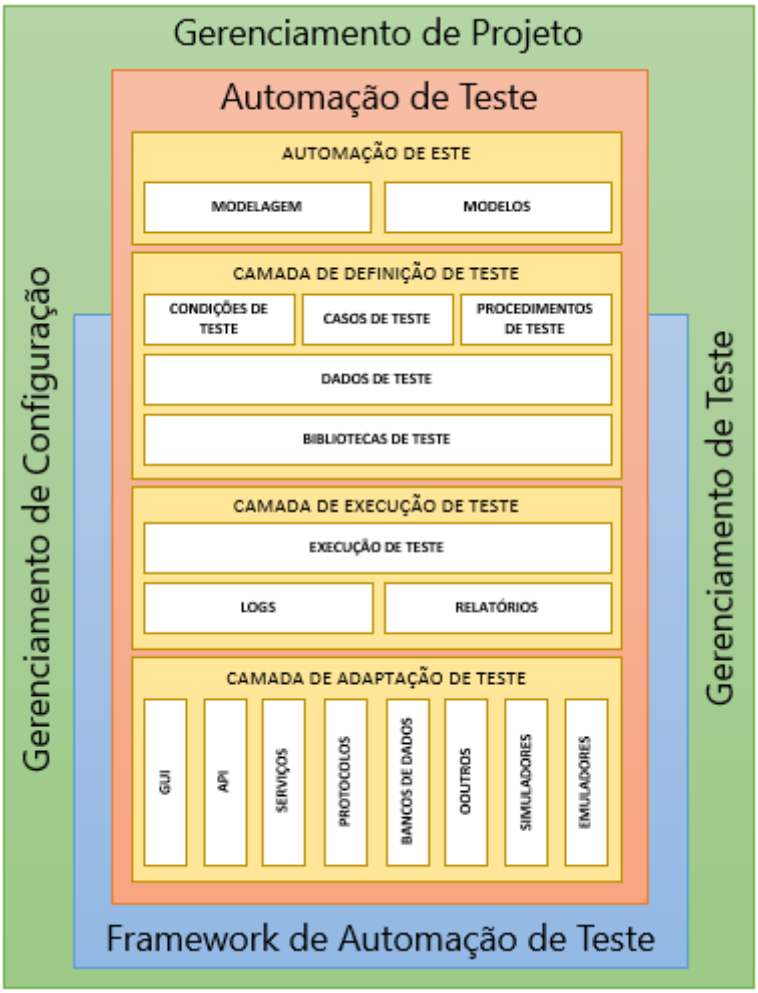


Figura 1: Arquitetura de Automação de Teste Genérica

3.1.2 Camada de geração de teste

A camada de geração de teste consiste em suporte de ferramenta para o seguinte:

- Criação manual de casos de teste;
- Desenvolvimento, captura ou obtenção de dados de teste;
- Geração automática de casos de teste a partir de modelos que definem o SUT e/ou seu ambiente (ou seja, testes automatizados baseados em modelos).

Os componentes nesta camada são usados para:

- Editar e navegar nas estruturas do conjunto de teste;
- Relacionar casos de teste com objetivos de teste ou requisitos do SUT;
- Documentar a modelagem do teste;
- Geração automatizada de testes, os seguintes recursos também podem ser incluídos:
  - Capacidade de modelar o SUT, seu ambiente ou o sistema de teste;
  - Capacidade de definir diretivas de teste e configurar algoritmos de geração de teste;
  - Capacidade de rastrear os testes gerados de volta para o modelo (elementos);

3.1.3 Camada de definição do teste

A camada de definição de teste consiste em suporte de ferramenta para:

- Especificar casos de teste (em um nível alto e/ou baixo);
- Definir dados de teste para casos de teste de baixo nível;
- Especificar procedimentos de teste para um caso de teste ou um conjunto de casos de teste
- Definir scripts de teste para a execução dos casos de teste;
- Conceder acesso às bibliotecas de teste conforme necessário (p. ex., em abordagens orientadas por palavras-chave).

Os componentes nesta camada são usados para:

- Particionar, restringir, parametrizar ou instanciar dados de teste;
- Especificar sequências de teste ou comportamentos de teste de pleno direito (incluindo declarações de controle e expressões), parametrizar e/ou agrupar;
- Documentar os dados de teste, casos de teste e procedimentos de teste.

3.1.4 Camada de execução de teste

A camada de execução de teste consiste em suporte a ferramentas para:

- Executar casos de teste automaticamente;
- Registrar as execuções de casos de teste;
- Relatar os resultados dos testes.

A camada de execução de teste pode consistir em componentes que fornecem os seguintes recursos:

- Configurar e derrubar o SUT para execução de teste;
- Configurar e desmontar conjuntos de testes (ou seja, conjunto de casos de teste incluindo dados de teste);
- Configurar e parametrizar a configuração do teste;
- Interpretar dados de teste e casos de teste e transformá-los em scripts executáveis;
- Instruir o sistema de teste e/ou o SUT para registro (filtrado) de execução de teste e/ou para injeção de falha;
- Analisar as respostas do SUT durante a execução do teste para orientar testes subsequentes;
- Validar as respostas do SUT (comparação de resultados esperados e reais) para resultados de execução de casos de teste automatizados;
- Controlar a execução do teste automatizado em tempo.

3.1.5 Camada de adaptação de teste

A camada de adaptação de teste consiste em suporte de ferramenta para:

- Controlar o ambiente de teste;
- Interagir com o SUT;
- Monitorar o SUT;

- Simular ou emular o ambiente SUT.

A camada de adaptação de teste fornece:

- Mediação entre as definições de teste de tecnologia neutra e os requisitos específicos de tecnologia do SUT e os dispositivos de teste;
- Aplicação de diferentes adaptadores específicos da tecnologia para interagir com o SUT
- Distribuição da execução do teste em vários dispositivos ou interfaces de teste ou executar testes localmente.

3.1.6 Gerenciamento de configuração de um TAS

Normalmente, um TAS está sendo desenvolvido em várias iterações ou versões e precisa ser compatível com as iterações ou versões do SUT. O gerenciamento de configuração de um TAS pode precisar incluir:

- Modelos de teste;
- Definições ou especificações de teste incluindo dados de teste, casos de teste e bibliotecas;
- Scripts de teste;
- Motores de execução de teste, ferramentas e componentes complementares;
- Adaptadores de teste para o SUT;
- Simuladores e emuladores para o ambiente SUT;
- Resultados de testes e relatórios de testes.

Esses itens constituem o *testware* e devem estar na versão correta para corresponder à versão do SUT. Em algumas situações, pode ser necessário reverter para versões anteriores do TAS, por exemplo, no caso de problemas de campo precisam ser reproduzidos com versões SUT mais antigas. Um bom gerenciamento de configuração possibilita essa capacidade.

3.1.7 Gerenciamento de projeto de um TAS

Como qualquer projeto de automação de teste é um projeto de software, ele requer o mesmo gerenciamento de projeto que qualquer outro projeto de software. Um TAE precisa executar as tarefas para todas as fases da metodologia estabelecida do ciclo de vida de desenvolvimento do software ao desenvolver o TAS. Além disso, um TAE precisa entender que o ambiente de desenvolvimento do TAS deve ser projetado de modo que as informações de status (métricas) possam ser extraídas facilmente ou automaticamente relatadas ao gerenciamento de projetos do TAS.

3.1.8 Suporte TAS para gerenciamento de testes

Um TAS deve suportar o gerenciamento de teste para o SUT. Os relatórios de teste, incluindo os logs e os resultados dos testes, precisam ser extraídos facilmente ou automaticamente pelo gerenciamento de teste (pessoas ou sistema) do SUT.

3.2 Modelagem TAA

3.2.1 Introdução à modelagem TAA

Há várias atividades principais necessárias para projetar uma TAA, que pode ser encomendada de acordo com as necessidades do projeto de automação de teste ou organização. Essas atividades são discutidas nas seções abaixo. Podem ser necessárias mais ou menos atividades, dependendo da complexidade do TAA.

Requisitos de captura necessários para definir uma TAA apropriada.

Os requisitos para uma abordagem de automação de teste precisam considerar o seguinte:

- Qual atividade ou fase do processo de teste deve ser automatizada, por exemplo, gerenciamento de teste, projeto de teste, geração de teste ou execução de teste. Observe que a automação de teste refina o processo fundamental inserindo a geração entre a modelagem e a implementação do teste;
- Qual nível de teste deve ser suportado, por exemplo, nível de componente, nível de integração, nível de sistema;
- Que tipo de teste deve ser suportado, por exemplo, teste funcional, teste de conformidade, testes de interoperabilidade;
- Que função de teste deve ser suportada, por exemplo, executor de teste, analista de teste, arquiteto de teste, gerente de teste;
- Qual produto de software, linha de produtos de software, família de produtos de software deve ser suportado, por exemplo, para definir a extensão e o tempo de vida do TAS implementado;
- Que tecnologias SUT devem ser suportadas, por exemplo, para definir o TAS em vista da compatibilidade com as tecnologias SUT.

Comparar e contrastar diferentes abordagens de projeto / arquitetura

O TAE precisa analisar os prós e contras de diferentes abordagens ao projetar camadas selecionadas do TAA. Estes incluem, mas não estão limitados a:

Considerações para a camada de geração de teste:

- Seleção de geração manual ou automática de testes;
- Seleção de, por exemplo, geração de teste baseada em requisitos, baseada em dados, baseada em cenários ou baseada em comportamento;
- Seleção de estratégias de geração de teste (p. ex., cobertura de modelo, como árvores de classificação para abordagens baseadas em dados, cobertura de caso de uso e cobertura de caso de exceção para abordagens baseadas em cenários, cobertura de transição, de estado e de caminho para abordagens baseadas em comportamento etc.);
- Escolhendo a estratégia de seleção de teste. Na prática, a geração completa de testes combinatórios é inviável, pois pode levar a uma explosão de casos de teste. Portanto, critérios

práticos de cobertura, pesos, avaliações de risco etc. devem ser usados para orientar a geração de teste e seleção de teste subsequente.

Considerações para a camada de definição de teste:

- Seleção de definição de teste orientada por dados, por palavras-chave, baseada em padrões ou por modelos;
- Seleção de notação para definição de teste, por exemplo, tabelas, notação baseada no estado, notação estocástica, notação de fluxo de dados, notação do processo de negócios, notação baseada em cenários, notação TTCN3, UML *Testing Profile* (UTP), entre outros;
- Seleção de guias de estilo e diretrizes para a definição de testes de alta qualidade;
- Seleção de repositórios de casos de teste (planilhas, bancos de dados, arquivos, ...).

Considerações para a camada de execução de teste:

- Seleção da ferramenta de execução de teste;
- Seleção de interpretação (usando uma máquina virtual) ou abordagem de compilação para implementar procedimentos de teste. Esta escolha depende tipicamente da ferramenta de execução de teste escolhida;
- Seleção da tecnologia para implementar procedimentos de teste (imperativo, como C, funcional, como Haskell ou Erlang, orientado a objeto, como C ++, C #, Java, *scripting*, como Python ou Ruby, ou uma ferramenta específica Tecnologia). Esta escolha é tipicamente dependente da ferramenta de execução de teste escolhida;
- Seleção de bibliotecas auxiliares para facilitar a execução do teste (p. ex., bibliotecas de dispositivos de teste, bibliotecas de codificação ou decodificação, ...).

Considerações para a camada de adaptação de teste:

- Seleção de interfaces de teste para o SUT;
- Seleção de ferramentas para estimular e observar as interfaces de teste;
- Seleção de ferramentas para monitorar o SUT durante a execução do teste;
- Seleção de ferramentas para rastrear a execução do teste (p. ex., incluindo o tempo da execução do teste.

Identificar áreas onde a abstração pode trazer benefícios

Abstração em uma TAA permite a independência de tecnologia em que o mesmo conjunto de teste pode ser usado em diferentes ambientes de teste e em diferentes tecnologias. A portabilidade dos artefatos de teste é aumentada. Além disso, a neutralidade do fornecedor é assegurada, o que evita efeitos de bloqueio para um TAS. A abstração também melhora a capacidade de manutenção e adaptabilidade a tecnologias SUT novas ou em desenvolvimento. Além disso, a abstração ajuda a tornar uma TAA (e suas instâncias por TAS) mais acessível a não-técnicos, uma vez que os conjuntos de teste podem ser documentados (incluindo meios gráficos) e explicados em um nível superior, o que melhora a legibilidade e a compreensão.

O TAE precisa discutir com os stakeholders em desenvolvimento de software, garantia de qualidade e testes que nível de abstração para usar em qual área do TAS. Por exemplo, que interfaces da camada de adaptação de teste ou de execução de teste precisam ser externalizadas, formalmente definidas e mantidas estáveis ao longo da vida útil da TAA? Ele também precisa ser discutido se uma

definição de teste abstrato está sendo usada ou se a TAA usa uma camada de execução de teste com scripts de teste apenas. Da mesma forma, precisa ser entendido se a geração de teste é abstraída pelo uso de modelos de teste e abordagens de teste baseadas em modelos. O TAE precisa estar ciente de que existem relações de compromisso entre implementações sofisticadas e diretas de uma TAA em relação à funcionalidade geral, manutenção e expansibilidade. Uma decisão sobre qual abstração usar em uma TAA precisa levar em conta estas relações de compromisso.

Quanto mais abstração for usada para um TAA, mais flexível será em relação à evolução futura ou à transição para novas abordagens ou tecnologias. Isso ocorre ao custo de investimentos iniciais maiores (p. ex., arquitetura e ferramentas de automação de teste mais complexas, requisitos de conjuntos de habilidades mais altos, curvas de aprendizado maiores), o que atrasa o equilíbrio inicial, mas pode compensar a longo prazo. Também pode levar a um menor desempenho do TAS.

Embora as considerações detalhadas do retorno sobre investimento (ROI) sejam da responsabilidade da TAM, o TAE precisa fornecer insumos para a análise do ROI, fornecendo avaliações técnicas e comparações de diferentes arquiteturas e abordagens de automação de teste com relação ao tempo, custos, esforços e benefícios.

Entenda as tecnologias SUT e como elas se interconectam com o TAS

O acesso às interfaces de teste do SUT é central para qualquer execução de teste automatizada. O acesso pode estar disponível nos seguintes níveis:

- Nível de software, por exemplo, SUT e software de teste estão ligados;
- Nível de API, por exemplo, o TAS invoca as funções, operações ou métodos fornecidos numa interface de programação de aplicações (remota);
- Nível de protocolo, por exemplo, o TAS interage com o SUT via HTTP, TCP etc.;
- Nível de serviço, por exemplo, o TAS interage com os serviços SUT via serviços da Web, serviços *RESTful*, entre outros.

Além disso, o TAE precisa decidir sobre o paradigma de interação da TAA a ser utilizada para a interação entre o TAS e o SUT, sempre que são separados por API, protocolos ou serviços. Estes paradigmas incluem o seguinte:

- *Paradigma impulsionado por eventos*, que controla a interação através de eventos sendo trocados em um barramento de eventos;
- *Paradigma cliente-servidor*, que impulsiona a interação através da invocação de serviço dos solicitadores de serviços ao provedor de serviços;
- *Paradigma ponto-a-ponto*, que impulsiona a interação via invocação de serviço.

Muitas vezes a escolha de paradigma depende da arquitetura SUT e pode ter implicações na arquitetura SUT. A interligação entre o SUT e a TAA precisa ser cuidadosamente analisada e projetada para selecionar uma arquitetura segura no futuro entre os dois sistemas.

Compreender o ambiente SUT

Um SUT pode ser um software, um software autônomo que funciona apenas em relação a outro software (p. ex., sistemas de sistemas), um hardware (p. ex., sistemas incorporados) ou componentes

ambientais (p. ex., sistemas cyber-físicos). Um TAS simula ou emula o ambiente SUT como parte de uma configuração de teste automatizada.

Exemplos de ambientes de teste e usos de exemplo incluem o seguinte:

- Um computador com o SUT e o TAS. Útil para testar um aplicativo de software;
- Computadores em rede individuais para um SUT e TAS, respetivamente. Útil para testar software de servidor;
- Dispositivos de teste adicionais para estimular e observar as interfaces técnicas de um SUT; Útil para testar o software, por exemplo, em um conversor de televisão;
- Dispositivos de teste em rede para emular o ambiente operacional do SUT. Útil para testar o software de um roteador de rede;
- Simuladores para simular o ambiente físico do SUT. Útil para testar o software de uma unidade de controle embutida.

Tempo e complexidade para uma determinada implementação de arquitetura de testware

Embora a estimativa de esforço para um projeto TAS seja responsabilidade de um TAM, um TAE precisa apoiar um TAM nisso, fornecendo boas estimativas para o tempo e a complexidade de um projeto TAA.

Métodos para estimativas e exemplos incluem o seguinte:

- Estimativa baseada em analogia, como pontos de funções, estimativa de três pontos, *Wide Band Delphi* e estimativa de testes;
- Estimativa pelo uso de estruturas de divisão de trabalho, como as encontradas em software de gerenciamento ou modelos de projeto;
- Estimativa paramétrica, como Modelo de Custo Construtivo (COCOMO);
- Estimativas baseadas em tamanho, tais como Análise de Ponto de Função, Análise de Ponto de História ou Análise de Caso de Uso;
- Estimativas de grupo, como o *Planning Poker.*

Facilidade de uso para uma determinada implementação de arquitetura de testware

Além da funcionalidade do TAS, da sua compatibilidade com o SUT, da sua estabilidade a longo prazo e evolução, dos seus requisitos de esforço e das considerações ROI, o TAE tem a responsabilidade específica de abordar questões de usabilidade para um TAS.

Isso inclui, mas não está limitado a:

- Modelagem orientada ao testador;
- Facilidade de utilização do TAS;
- Suporte do TAS para outras funções no desenvolvimento de software, garantia de qualidade e gerenciamento de projetos;
- Organização eficaz, navegação e busca dentro ou com o TAS;
- Documentação útil, manuais e texto de ajuda para o TAS;

- Relatórios práticos por e sobre o TAS;
- Desenhos iterativos para abordar o feedback do TAS e *insights* empíricos.

3.2.2 Abordagens para automatizar casos de teste

Os casos de teste precisam ser traduzidos em sequências de ações que são executadas contra um SUT. Essa sequência de ações pode ser documentada em um procedimento de teste e/ou um script de teste. Além das ações, os casos de teste automatizados também devem definir dados de teste para a interação com o SUT e incluir etapas de verificação para verificar que o resultado esperado foi alcançado pelo SUT.

Várias abordagens podem ser usadas para criar a sequência de ações:

1. O TAE implementa os casos de teste diretamente em scripts de teste automatizados. Esta opção é a menos recomendada, pois carece de abstração e aumenta a carga de manutenção.
2. O TAE projeta procedimentos de teste e os transforma em scripts de teste automatizados. Esta opção tem abstração, mas falta automatização para gerar os scripts de teste.
3. O TAE utiliza uma ferramenta para traduzir procedimentos de teste em scripts de teste automatizados. Esta opção combina a abstração e a geração automática de scripts.
4. O TAE utiliza uma ferramenta que gera procedimentos automatizados de teste ou traduz os scripts de teste diretamente dos modelos. Esta opção tem o grau mais alto de automação.

Observe que as opções dependem fortemente do contexto do projeto. Também pode ser eficiente iniciar a automação de teste aplicando uma das opções menos avançadas, pois estas são tipicamente mais fáceis de implementar. Isso pode fornecer valor agregado a curto prazo, embora isso resulte em uma solução menos sustentável.

Abordagens bem estabelecidas para automatizar casos de teste incluem:

- **Captura e reprodução**, que pode ser usada para a opção 1;
- **Script estruturado**, abordagem orientada por dados e abordagem orientada por palavras-chave, que pode ser usada para a opção 2 ou 3;
- **Testes baseados em modelos** (incluindo a abordagem orientada por processos), que podem ser utilizados para a opção 4.

Essas abordagens são explicadas posteriormente em termos de conceitos principais e prós e contras.

Abordagem de captura e reprodução

Conceito principal

Nas abordagens de captura e reprodução, as ferramentas são usadas para capturar interações com o SUT enquanto executa a sequência de ações conforme definido por um procedimento de teste. Entradas são capturadas, e as saídas também podem ser registradas para verificações posteriores. Durante a repetição de eventos, existem várias possibilidades de verificação de saída manual e automatizada:

- **Manual**: o testador tem de procurar as anomalias nas saídas do SUT;

- **Completo**: todas as saídas do sistema gravadas durante a captura devem ser reproduzidas;
- **Exato**: todas as saídas do sistema que foram gravadas durante a captura devem ser reproduzidas pelo SUT para gravação com nível de detalhe;
- **Pontos de verificação**: apenas determinadas saídas do sistema são verificadas em determinados pontos para os valores especificados.

Prós

A abordagem de captura e reprodução pode ser usada para SUT no nível GUI e/ou API. Inicialmente, é fácil de configurar e usar.

Contras

Os scripts de captura e reprodução são difíceis de manter e evoluir porque a captura da execução do SUT depende fortemente da versão do SUT a partir da qual a captura foi realizada. Por exemplo, ao gravar no nível GUI, as alterações no layout da GUI podem afetar o script de teste, mesmo que seja apenas uma alteração no posicionamento de um elemento GUI. Portanto, as abordagens de captura e reprodução permanecem vulneráveis a mudanças.

A implementação dos casos de teste (scripts) só pode ser iniciada quando o SUT estiver disponível.

Scripts lineares

Conceito Principal

Como com todas as técnicas de script, o script linear começa com alguns procedimentos de teste manual. Note que estes não podem ser documentos escritos - o conhecimento sobre quais testes executar e como executá-los podem ser "conhecidos" por um ou mais analistas de teste.

Cada teste é executado manualmente enquanto a ferramenta de teste registra a sequência de ações e, em alguns casos, captura a saída visível do SUT para a tela (*printscreen*). Isso geralmente resulta em um script (geralmente grande) para cada procedimento de teste. Os scripts gravados podem ser editados para melhorar a legibilidade (p. ex., adicionando comentários para explicar o que está acontecendo em pontos-chave) ou adicione mais verificações usando a linguagem de script da ferramenta.

Os scripts podem então ser reproduzidos pela ferramenta, fazendo com que a ferramenta repita as mesmas ações tomadas pelo testador quando o script foi gravado. Embora isso possa ser usado para automatizar testes GUI, não é uma boa técnica para usar onde grandes números de testes devem ser automatizados e são necessários para muitas versões do software. Isto é devido ao custo de manutenção elevado que normalmente é causado por alterações ao SUT (cada alteração no SUT pode exigir muitas alterações nos scripts gravados).

Prós

As vantagens dos scripts lineares se concentram no fato de que há pouco ou nenhum trabalho de preparação necessário antes que você possa começar a automatizar. Uma vez que você aprendeu a usar a ferramenta é simplesmente uma questão de gravar um teste manual e

reproduzi-lo (embora a parte de gravação deste pode exigir interação adicional com a ferramenta de teste para solicitar que as comparações do realizado com a saída esperada ocorre para verificar se o *software* está funcionando corretamente). Habilidades de programação não são necessários, mas geralmente são úteis.

Contras

As desvantagens dos scripts lineares são numerosas. A quantidade de esforço necessária para automatizar qualquer procedimento de teste será dependente, em grande parte, do tamanho (número de passos ou ações) necessários para executá-lo. Assim, o 1000º procedimento de teste a ser automatizado terá uma quantidade de esforço proporcionalmente semelhante à do procedimento de teste 100º. Em outras palavras, não há muito espaço para diminuir o custo de construção de novos testes automatizados.

Além disso, se houvesse um segundo script que realizasse um teste semelhante, embora com diferentes valores de entrada, esse script conteria a mesma sequência de instruções do primeiro script. Apenas as informações incluídas nas instruções (conhecidas como argumentos de instrução ou parâmetros) seriam diferentes. Se houvesse vários testes (e, portanto, scripts), todos eles conteriam a mesma sequência de instruções, tudo isso precisaria ser mantido sempre que o software mudasse de uma forma que afetasse os scripts.

Como os scripts estão em uma linguagem de programação, ao invés de uma linguagem natural, os não-programadores podem achá-los difíceis de entender. Algumas ferramentas de teste usam linguagens proprietárias (exclusivas para a ferramenta), portanto, leva tempo para aprender a linguagem e se tornar proficiente com ela.

Os scripts gravados contêm apenas declarações gerais nos comentários, se houver algum. Os scripts longos, em particular, são mais bem documentados com comentários para explicar o que está acontecendo em cada etapa do teste. Isso facilita a manutenção. Os scripts podem se tornar muito grandes (contendo muitas instruções) quando o teste inclui muitas etapas.

Os scripts não são modulares e difíceis de manter. Os scripts lineares não seguem paradigmas comuns de reutilização de software e modularidade e estão estreitamente associados à ferramenta utilizada.

Script estruturado

Conceito Principal

A principal diferença entre a técnica de script estruturado e a técnica de script linear é a introdução de uma biblioteca de scripts. Ela contém scripts reutilizáveis que executam sequências de instruções comumente necessárias em vários testes. São exemplos bons de tais scripts aqueles que interagem, por exemplo, com as operações de interfaces SUT.

Prós

Os benefícios desta abordagem incluem uma redução significativa nas mudanças de manutenção e o custo reduzido de automatizar novos testes (porque eles podem usar scripts que já existem em vez de ter que criá-los todos do zero).

As vantagens do script estruturado são em grande parte obtidas através da reutilização de scripts. Mais testes podem ser automatizados sem ter que criar o volume de scripts que uma abordagem de script linear exigiria. Isso tem um impacto direto nos custos de construção e manutenção. Os testes subsequentes não terão tanto esforço para automatizar porque alguns dos scripts criados para implementar o primeiro teste podem ser reutilizados.

Contras

O esforço inicial para criar os scripts compartilhados pode ser visto como uma desvantagem, mas esse investimento inicial deve pagar grandes dividendos se abordado adequadamente. Serão necessárias habilidades de programação para criar todos os scripts já que uma simples captura não será suficiente. A biblioteca de scripts deve ser bem gerenciada, ou seja, os scripts devem ser documentados e deve ser fácil para Analistas de Testes encontrar os scripts necessários (convenção de nomenclatura e organização).

Teste orientado por dados

Conceito Principal

A técnica de script baseada em dados baseia-se na técnica de scripts estruturados. A diferença mais significativa é como as entradas de teste são manipuladas. As entradas são extraídas dos scripts e colocadas em um ou mais arquivos separados (normalmente chamados arquivos de dados).

Isso significa que o script de teste principal pode ser reutilizado para implementar uma série de testes (em vez de apenas um único teste). Normalmente, o script de teste principal "reutilizável" é chamado de script de "controle". O script de controle contém a sequência de instruções necessárias para executar os testes, mas lê os dados de entrada de um arquivo de dados. Um teste de controle pode ser usado para muitos testes, mas geralmente é insuficiente para automatizar uma ampla gama de testes. Assim, um número de scripts de controle será necessário, mas que é apenas uma fração do número de testes que são automatizados.

Prós

O custo de adicionar novos testes automatizados pode ser significativamente reduzido por esta técnica de script. Esta técnica é usada para automatizar variações de um teste, dando maior profundidade aos testes em uma área específica podendo aumentar a cobertura do teste.

Ter os testes "descritos" pelos arquivos de dados significa que os analistas de teste podem especificar testes "automatizados" simplesmente preenchendo um ou mais arquivos de dados. Isso dá aos Analistas de Teste mais liberdade para especificar testes automatizados sem dependência tanto dos Analisadores de Teste Técnicos (que podem ser um recurso escasso).

Contras

A necessidade de gerenciar arquivos de dados e certificar-se de que eles são legíveis pelo TAS é uma desvantagem, mas pode ser abordado adequadamente.

Além disso, importantes casos de teste negativos podem ser perdidos. Os testes negativos são uma combinação de procedimentos de teste e dados de teste. Numa abordagem direcionada para os dados de teste, principalmente, os "procedimentos de teste negativos" podem ser omitidos.

Testes por palavras-chave

Conceito principal

A técnica de script baseada em palavras-chave baseia-se na técnica de script orientado por dados. Existem duas diferenças principais: (1) os arquivos de dados são agora chamados de 'arquivos de definição de teste' ou algo semelhante (p. ex., arquivos de palavra de ação); (2) existe apenas um script de controle.

Um arquivo de definição de teste contém uma descrição dos testes de uma forma que deve ser mais fácil para os analistas de teste entenderem (mais fácil do que o arquivo de dados equivalente). Geralmente conterá dados como os arquivos de dados, mas os arquivos de palavras-chave também conterão instruções de alto nível (as palavras-chave).

As palavras-chave devem ser escolhidas para serem significativas para o Analista de Testes, os testes que estão sendo descritos e a aplicação que está sendo testada. Esses são principalmente (mas não exclusivamente) usados para representar interações de negócios de alto nível com um sistema (p. ex., "classificação"). Cada palavra-chave representa um número de interações detalhadas com o sistema em teste. As sequências de palavras-chave (incluindo os dados de teste relevantes) são usadas para especificar os casos de teste. Palavras-chave especiais podem ser usadas para as etapas de verificação, ou podem conter as ações e as etapas de verificação.

O escopo de responsabilidade dos Analistas de Teste inclui criar e manter os arquivos de palavras-chave. Isso significa que, uma vez que os scripts de suporte são implementados, os analistas de teste podem adicionar testes "automatizados" simplesmente especificando-os em um arquivo de palavras-chave (como com scripts de dados).

Prós

Uma vez que o script de controle e de suporte para as palavras-chave foram escritos, o custo de adicionar novos testes automatizados será muito reduzido por esta técnica.

Ter os testes "descritos" pelos arquivos de palavras-chave significa que os Analistas de Teste podem especificar testes "automatizados" simplesmente usando as palavras-chave e os seus dados associados ao escrever seus testes. Isso dá aos Analistas de Teste mais liberdade para especificar testes automatizados sem depender tanto dos Analistas Técnicos em Teste (que podem ser um recurso escasso). O benefício desta abordagem sobre a baseada em dados neste respeito é o uso das palavras-chave. Cada palavra-chave deve representar uma sequência de ações detalhadas que produzem algum resultado significativo. Por exemplo, "criar conta", "ordem de lugar", "status de ordem de verificação" são todas as ações possíveis para um aplicativo de compras *on-line* que envolvem um número de etapas detalhadas. Quando um analista de teste descreve um teste de sistema para outro analista de teste, é provável que falem em termos dessas ações de alto nível, e não as etapas detalhadas. O

objetivo da abordagem orientada por palavras-chave é, então, implementar essas ações de alto nível e permitir que os testes sejam definidos em termos de ações de alto nível sem referência às etapas detalhadas.

Esses casos de teste são mais fáceis de manter, ler e escrever, pois a complexidade pode ser ocultada nas palavras-chave (ou nas bibliotecas, no caso de uma abordagem de script estruturada). As palavras-chave podem oferecer uma abstração das complexidades das interfaces do SUT.

Contras

Implementar as palavras-chave continua a ser uma grande tarefa para o Engenheiro de Automação de Teste, especialmente se estiver usando uma ferramenta que não oferece suporte para esta técnica de script. Para sistemas pequenos, pode ser demasiada sobrecarga para implementar e os custos superariam os benefícios.

Cuidado deve ser tomado para garantir que as palavras-chave corretas serão implementadas. Boas palavras-chave serão usadas muitas vezes em muitos testes diferentes enquanto palavras-chave pobres provavelmente serão usadas apenas uma vez ou poucas vezes.

Script direcionado a processo

Conceito principal

A abordagem baseada em processos baseia-se na técnica de script orientada por palavras-chave, com a diferença de que cenários - representando casos de uso do SUT e suas variantes - constituem os scripts que são parametrizados com dados de teste ou combinados em definições de teste de nível superior.

Tais definições de teste são mais fáceis de lidar, uma vez que a relação lógica entre ações, por exemplo, "status de ordem de verificação" após "ordem de lugar" no teste de caraterística ou "status de ordem de verificação" sem "ordem de lugar" anterior no teste de robustez pode ser determinada.

Prós

A utilização de uma definição de casos de teste baseada em processos e baseada em cenários, permite que os procedimentos de teste sejam definidos a partir de uma perspectiva de fluxo de trabalho. O objetivo da abordagem orientado por processos é implementar esses fluxos de trabalho de alto nível usando bibliotecas de teste que representam as etapas de teste detalhadas (consulte também a abordagem orientada por palavras-chave).

Contras

Processos de um SUT podem não ser fáceis de compreender por um Analista Técnico de Teste - e assim é a implementação dos scripts orientados a processos, particularmente nenhuma lógica do processo de negócios é suportada pela ferramenta.

Cuidado também deve ser tomado para garantir que os processos corretos, por meio de palavras-chave corretas, sejam implementados. Os processos bons serão referenciados por outros processos e resultarão em muitos testes relevantes, enquanto os processos

deficientes não serão rentáveis em termos de relevância, capacidade de detecção de erros etc.

Teste baseado em modelo

Conceito Principal

O teste baseado em modelo refere-se à geração automatizada de casos de teste (veja também o syllabus Teste Baseado em Modelo no BSTQB) - em oposição à execução automatizada de casos de teste - por meio de captura e reprodução, scripts lineares, scripts estruturados, scripts baseados em dados ou scripts baseados em processos. Testes baseados em modelos usam modelos (semi-) formais que abstraem das tecnologias de script do TAA. Diferentes métodos de geração de teste podem ser usados para derivar testes para qualquer um dos frameworks de script discutidos anteriormente.

Prós

O teste baseado em modelos permite que a abstração se concentre na essência do teste (em termos de lógica de negócios, dados, cenários, configurações etc. a serem testados). Ele também permite gerar testes para diferentes sistemas de destino e tecnologias específicas, de modo que os modelos utilizados para a geração de teste constituam uma representação de *testware* que pode ser reutilizada e mantida à medida que a tecnologia evolui.

Em caso de alterações nos requisitos, o modelo de teste só deve ser adaptado. Um conjunto completo de casos de teste é gerado automaticamente. As técnicas de modelagem de casos de teste são incorporadas nos geradores de casos de teste.

Contras

A perícia de modelagem é necessária para executar esta abordagem de forma eficaz. A tarefa de modelar abstraindo interfaces, dados ou comportamentos de um SUT pode ser difícil. Além disso, as ferramentas de modelagem e de teste baseadas em modelos ainda não são maduras. As abordagens de teste baseadas em modelos requerem ajustes nos processos de teste. Por exemplo, o papel do modelador de teste precisa ser estabelecido. Além disso, os modelos utilizados para a geração de teste constituem artefatos importantes para a garantia de qualidade de um SUT e precisam ser assegurados e mantidos também.

3.2.3 Considerações técnicas do SUT

Além disso, aspetos técnicos de um SUT devem ser considerados ao projetar um TAA. Alguns destes são discutidos abaixo, embora esta não é uma lista completa, mas deve servir como uma amostra de alguns aspetos importantes.

Interfaces do SUT

Um SUT possui interfaces internas (dentro do sistema) e interfaces externas (para o ambiente do sistema e seus usuários ou por componentes expostos). Uma TAA precisa ser capaz de controlar e/ou observar todas as interfaces do SUT que são potencialmente afetadas pelos procedimentos de teste (isto é, as interfaces precisam ser testáveis). Além disso, também pode haver a necessidade de

registrar as interações entre o SUT e o TAS com diferentes níveis de detalhe, incluindo tipicamente a impressão de tempo.

Durante a definição da arquitetura, é necessário o foco do teste no início do projeto (ou continuamente em ambientes ágeis) para verificar a disponibilidade das interfaces de teste ou instalações de teste necessárias para que o SUT seja testável (modelagem para testabilidade).

Dados SUT

Um SUT usa dados de configuração para controlar sua instanciação, configuração, administração etc. Além disso, ele usa dados de usuário que processa. Um SUT também pode usar dados externos de outros sistemas para completar suas tarefas. Dependendo dos procedimentos de teste, todos esses tipos de dados precisam ser definíveis, configuráveis e capazes de serem instanciados pelo TAA. A maneira específica de lidar com os dados é decidida no projeto TAA. Dependendo da abordagem, os dados podem ser tratados como parâmetros, folhas de dados de teste, bancos de dados de teste, dados reais etc.

Configurações SUT

Um SUT pode ser implementado em diferentes configurações, por exemplo, em diferentes sistemas operacionais, dispositivos de destino ou com configurações de idioma diferentes. Diferentes configurações de SUT podem ser tratadas pela TAA dependendo dos procedimentos de teste. Os procedimentos de teste podem exigir diferentes configurações de teste (em um laboratório) ou configurações de teste virtual (na nuvem) da TAA em combinação com uma determinada configuração do SUT. Também pode ser necessário adicionar simuladores e/ou emuladores de componentes selecionados para os aspetos selecionados do SUT.

Padrões SUT e configurações legais

Além dos aspetos técnicos de um SUT, o projeto do TAA pode precisar respeitar os requisitos legais e/ou de normas de modo a projetar a TAA de forma compatível. Exemplos incluem requisitos de privacidade para os dados de teste ou requisitos de confidencialidade que afetam as capacidades de log e relatório da TAA.

Ferramentas e ambientes de ferramentas usados para desenvolver o SUT

Junto com o desenvolvimento de um SUT, diferentes ferramentas podem ser usadas para a engenharia de requisitos, design e modelagem, codificação, integração e implantação do SUT. A TAA, juntamente com as suas próprias ferramentas, deve ter em conta a paisagem da ferramenta SUT, a fim de permitir a compatibilidade das ferramentas, a rastreabilidade e/ou a reutilização de artefatos.

Testar interfaces no produto de software

É altamente recomendável não remover todas as interfaces de teste antes da liberação do produto. Na maioria dos casos, essas interfaces podem ser deixadas no SUT sem causar problemas com o produto finalizado. Quando deixadas no local, as interfaces podem ser usadas por engenheiros de serviço e suporte para diagnóstico de problemas, bem como para lançamentos de testes de manutenção. É importante verificar se as interfaces não representam riscos de segurança. Se

necessário, os desenvolvedores normalmente podem desativar essas interfaces de teste de forma que não possam ser usadas fora do departamento de desenvolvimento.

3.2.4 Considerações para processos de desenvolvimento e qualidade

Os aspetos dos processos de desenvolvimento e de garantia de qualidade de um SUT devem ser considerados na concepção de um TAA. Alguns destes são discutidos abaixo, embora esta não é uma lista completa, mas deve servir como uma amostra dos aspetos importantes.

Requisitos de controle de execução de teste

Dependendo do nível de automação exigido pela TAA, a execução do teste interativo, a execução do teste em modo batch ou a execução de teste totalmente automatizada pode precisar ser suportada pelo TAA.

Requisitos de relatórios

Dependendo dos requisitos de relatórios, incluindo tipos e suas estruturas, a TAA precisa ser capaz de suportar relatórios de teste fixos, parametrizados ou definidos em diferentes formatos e layouts.

Papel e direitos de acesso

Dependendo dos requisitos de segurança, a TAA pode ser necessária para fornecer um papel e sistema de direitos de acesso.

Panorama de ferramenta estabelecida

O gerenciamento de projetos SUT, gerenciamento de teste, repositório de código e teste, rastreamento de defeitos, gerenciamento de incidentes, análise de risco etc., podem ser suportados por ferramentas que compõem a ferramenta estabelecida. A TAA também é suportada por uma ferramenta ou conjunto de ferramentas que precisa se integrar perfeitamente com as outras ferramentas no ambiente. Além disso, os scripts de teste devem ser armazenados e versionados como o código SUT para que as revisões sigam o mesmo processo para ambos.

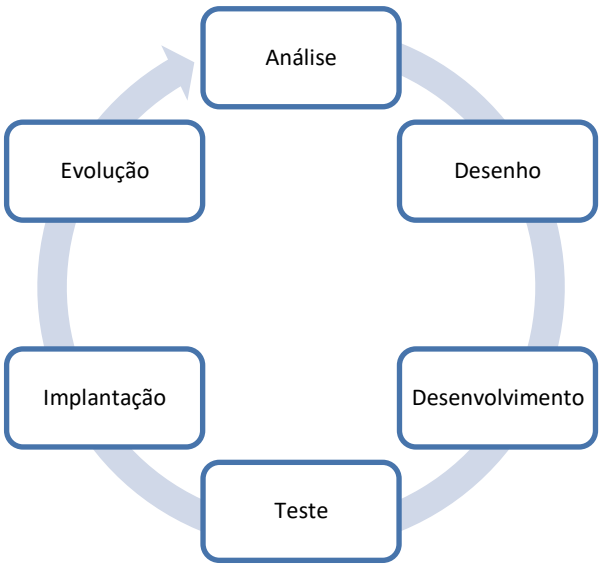
3.3 Desenvolvimento do TAS

3.3.1 Introdução ao desenvolvimento do TAS

O desenvolvimento de um TAS é comparável a outros projetos de desenvolvimento de software. Ele pode seguir os mesmos procedimentos e processos, incluindo revisões por pares por desenvolvedores e testadores. Específicos para um TAS são sua compatibilidade e sincronização com o SUT. Estas exigem consideração no desenho da TAA (ver capítulo 3.2) e no desenvolvimento do TAS. Além disso, o SUT é afetado pela estratégia de teste, por exemplo, tendo que disponibilizar interfaces de teste ao TAS.

Esta seção usa o ciclo de vida de desenvolvimento de software (SDLC) para explicar o processo de desenvolvimento do TAS e os aspetos relacionados ao processo de compatibilidade e sincronização com o SUT. Estes aspetos são igualmente importantes para qualquer outro processo de desenvolvimento que tenha sido escolhido ou esteja em vigor para o desenvolvimento de SUT e/ou TAS - eles precisam ser adaptados em conformidade.

O ciclo de vida de desenvolvimento de sistema básico para TAS é mostrado ao lado.



O conjunto de requisitos para um TAS precisa ser analisado e coletado. Os requisitos orientam a concepção do TAS como definido pela sua TAA (ver capítulo 3.2). O projeto é transformado em software por abordagens de engenharia de software. Observe que um TAS também pode usar hardware de dispositivo de teste dedicado, que está fora de consideração para este plano de estudos. Como qualquer outro software, um TAS precisa ser testado. Isso normalmente é feito por testes de capacidade básica que são seguidos por uma interação entre o TAS e SUT. Após a implantação e o uso de um TAS, muitas vezes é necessária uma evolução para adicionar mais capacidade de teste, alterar testes ou atualizar o TAS para coincidir com mudanças no SUT. A evolução do TAS requer uma nova rodada de desenvolvimento de acordo com o ciclo de vida de desenvolvimento de software.

Observe também que o ciclo de vida não mostra o backup, arquivamento e desmontagem de um TAS. Assim como o desenvolvimento do TAS, esses procedimentos devem seguir os métodos estabelecidos da organização.

3.3.2 Compatibilidade entre o TAS e o SUT

Compatibilidade de processo

O teste do SUT deve ser sincronizado com o seu desenvolvimento, assim a automação de teste deverá ser sincronizada com o desenvolvimento. Por conseguinte, é vantajoso coordenar os processos para desenvolvimento do SUT, do TAS e dos testes. Um grande ganho pode ser alcançado quando o desenvolvimento do SUT e do TAS são compatíveis em termos de estrutura de processo, gerenciamento de processos e suporte de ferramentas.

Compatibilidade de equipe

A compatibilidade de equipes é outro aspeto da entre o desenvolvimento do TAS e do SUT. Se uma mentalidade compatível é usada para abordar e gerenciar o TAS e o desenvolvimento do SUT, ambas as equipes se beneficiarão com a revisão dos requisitos, modelagens e/ou artefatos de

desenvolvimento, discutindo problemas e encontrando soluções compatíveis. A compatibilidade entre membros da equipe também ajuda na comunicação e interação uns com os outros.

Compatibilidade tecnológica

Além disso, a compatibilidade tecnológica entre o TAS e o SUT deve ser considerada. É benéfico para projetar e implementar uma interação perfeita desde o início. Mesmo que isso não seja possível (p. ex., porque as soluções técnicas não estão disponíveis), pode ser possível uma interação contínua através do uso de adaptadores, invólucros ou outros meios.

Compatibilidade de ferramentas

A compatibilidade de ferramentas, o desenvolvimento, e a garantia de qualidade precisam ser consideradas. Por exemplo, se forem utilizadas as mesmas ferramentas para gerenciamento de requisitos e/ou de problemas, será mais fácil o intercâmbio das informações e a coordenação do desenvolvimento do TAS e SUT.

3.3.3 Sincronização entre TAS e SUT

Sincronização de requisitos

Após a elicitação de requisitos, os requisitos de SUT e TAS devem ser desenvolvidos. Os requisitos de TAS podem ser agrupados em dois grupos principais:

(1) Requisitos que abordam o desenvolvimento como um sistema baseado em software, como requisitos para as caraterísticas para um projeto de teste, especificação de teste, análise de resultado de teste etc.
(2) Requisitos que abordam o teste do SUT por meio do TAS. Esses chamados requisitos de teste correspondem aos requisitos de SUT e refletem todos os recursos e propriedades do SUT que serão testados pelo TAS. Sempre que os requisitos forem atualizados, é importante verificar a consistência entre ambos e verificar se todos os requisitos de SUT que devem ser testados pelo TAS têm requisitos de teste definidos.

Sincronização de fases de desenvolvimento

A fim de ter o TAS pronto para testar o SUT, as fases de desenvolvimento precisam ser coordenadas. É mais eficiente quando os requisitos, modelagens, especificações e implementações são sincronizados.

Sincronização do rastreamento de defeitos

Os defeitos podem estar relacionados ao SUT, ao TAS aos requisitos, a modelagens e às especificações. Devido à relação entre os dois projetos, sempre que um defeito é corrigido dentro de um, a ação corretiva pode afetar o outro. O rastreamento de defeitos e os testes de confirmação têm de abordar tanto o TAS como o SUT.

Sincronização da evolução SUT e TAS

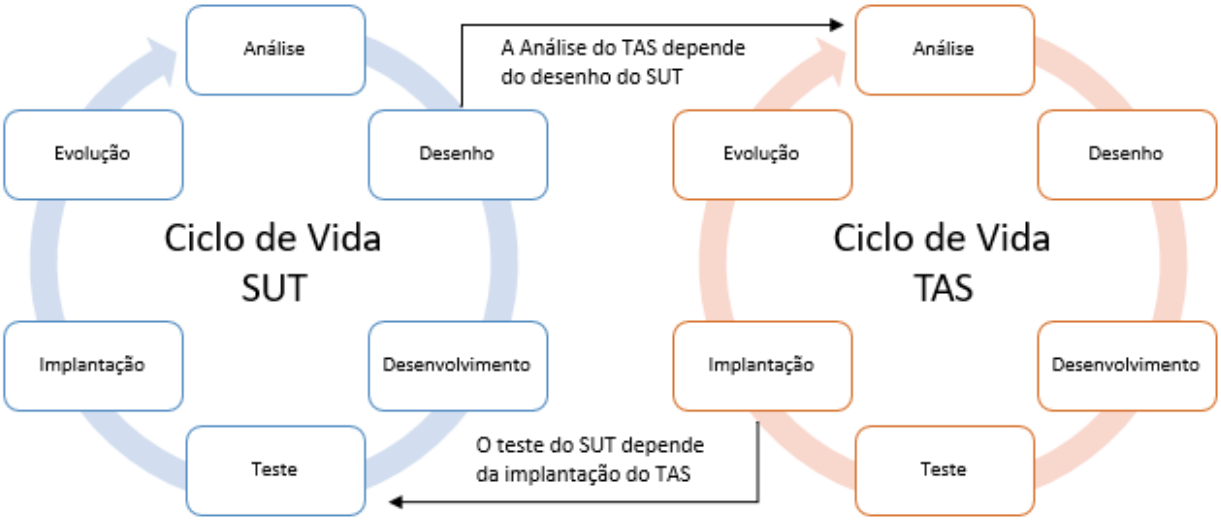
Tanto o SUT como o TAS podem evoluir para acomodar novos recursos ou desativar recursos, corrigir defeitos ou solucionar alterações no ambiente (incluindo alterações no SUT e no TAS, respetivamente, uma vez que um é um componente de ambiente para o outro).

Qualquer alteração aplicada a um SUT ou a um TAS pode afetar o outro de modo que o gerenciamento dessas mudanças deve abordar tanto o SUT quanto o TAS.

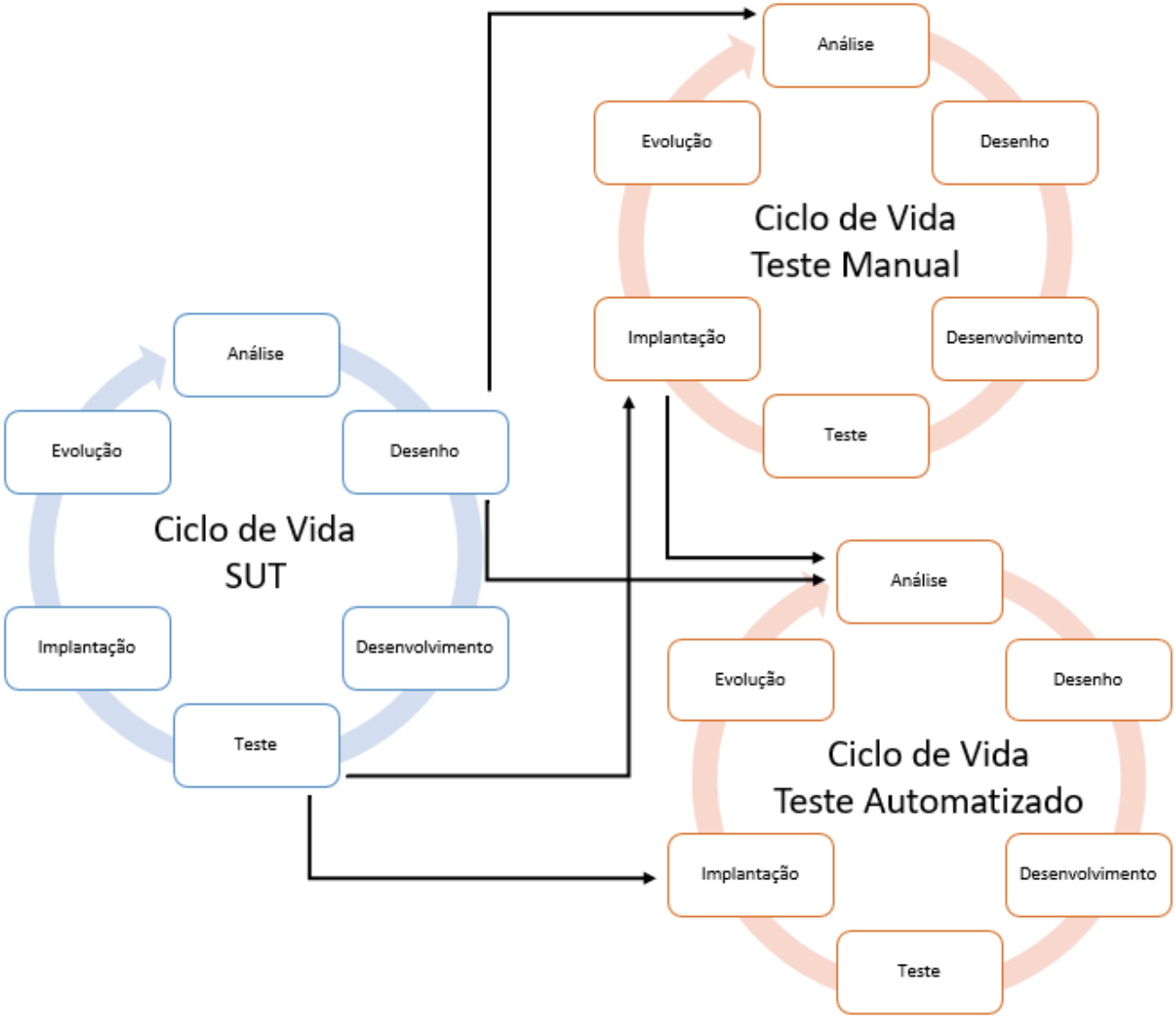
Duas abordagens de sincronização entre os processos de desenvolvimento SUT e TAS estão ilustradas nas próximas figuras.

A figura abaixo mostra uma abordagem em que os dois processos de ciclo de desenvolvimento de software para o SUT e o TAS são principalmente sincronizados em duas fases:

(1) A análise TAS é baseada no desenho do SUT;
(2) O teste do SUT faz uso do TAS implantado.



A figura seguinte mostra uma abordagem híbrida com testes manuais e automatizados. Sempre que os testes manuais são utilizados antes dos testes serem automatizados ou sempre que os testes manuais e automatizados são usados em conjunto, a análise TAS deve ser baseada tanto no design SUT quanto nos testes manuais. Desta forma, o TAS é sincronizado com ambos. O segundo grande ponto de sincronização para tal abordagem é como antes: o teste SUT requer testes implantados, que no caso de testes manuais poderiam ser apenas os procedimentos de teste manual a serem seguidos.



3.3.4 Reutilização na construção do TAS

A reutilização de um TAS se refere a reutilização de artefatos do TAS (de qualquer nível de sua arquitetura) em linhas de produtos, frameworks de produtos, domínios de produtos e/ou famílias de projetos. Os requisitos para reutilização resultam da relevância dos artefatos TAS para as outras variantes de produtos, produtos e/ou projetos.

Os artefatos TAS reutilizáveis podem incluir:

- (Partes de) modelos de teste de metas de teste, cenários de teste, componentes de teste ou dados de teste;
- (Partes de) casos de teste, dados de teste, procedimentos de teste ou bibliotecas de teste;
- O mecanismo de teste e/ou a estrutura do relatório de teste;
- Os adaptadores para os componentes e/ou interfaces SUT.

Embora os aspetos de reuso já estejam estabelecidos quando a TAA é definida, o TAS pode ajudar a aumentar a capacidade de reutilização por:

- Seguir a TAA ou revisá-la e atualizá-lo sempre que necessário;
- Documentar os artefatos TAS para que sejam facilmente compreendidos e possam ser incorporados em novos contextos;
- Assegurar a correção de qualquer artefato TAS para que o uso em novos contextos seja suportado por sua alta qualidade.

É importante notar que enquanto o design para reutilização é principalmente uma questão para o TAA, a manutenção e melhorias para reutilização são uma preocupação ao longo do ciclo de vida TAS. Exige uma consideração e um esforço contínuos para que a reutilização aconteça, para medir e demonstrar o valor agregado da reutilização, e para evangelizar outras pessoas para reutilizar os TAS existentes.

3.3.5 Apoio a uma variedade de sistemas-alvo

O suporte ao TAS para uma variedade de sistemas-alvo refere-se à capacidade do TAS para testar diferentes configurações de um produto de software.

As configurações referem-se a qualquer um dos seguintes:

- Número e interligação de componentes do SUT;
- Ambientes (software e hardware) nos quais os componentes do SUT são executados;
- Tecnologias, linguagens de programação ou sistemas operacionais utilizados para implementar os componentes do SUT;
- Bibliotecas e pacotes que os componentes do SUT estão usando;
- Ferramentas usadas para implementar os componentes do SUT.

Enquanto os quatro primeiros aspetos impactam o TAS em qualquer nível de teste, o último se aplica principalmente no nível de componente e de integração de testes.

A capacidade de um TAS para testar diferentes configurações de produto de software é determinada quando a TAA é definida. No entanto, o TAS tem de implementar a capacidade de lidar com variações técnicas e permitir que a gestão dos recursos do TAS e componentes necessários para diferentes configurações de um produto de software.

O tratamento da variedade TAS em relação à variedade do produto de software pode ser tratado de forma diferente:

- O gerenciamento de versão e configuração pode ser usado para fornecer as respetivas versões e configurações do TAS e SUT que se encaixam umas com as outras.

É importante notar que enquanto a modelagem para a variabilidade TAS é principalmente uma questão para o TAA, a manutenção e melhorias para esta variabilidade é uma preocupação de longo prazo no ciclo de vida do TAS. Requer consideração e esforço contínuo para revisar, adicionar e até remover opções e formas de variabilidade.

4 Riscos de implantação e contingência [150 min]

Palavras-chave

Risco, mitigação de risco, avaliação de risco, risco de produto

Objetivos de Aprendizagem

4.1 Seleção da abordagem de automação de teste e planejamento da implementação e implantação

ALTA-E-4.1.1 (K3): Aplicar diretrizes que suportem atividades piloto e de implantação de ferramentas de teste eficazes.

4.2 Avaliação do risco e estratégias de mitigação

ALTA-E-4.2.1 (K4): Analisar os riscos de implantação e identificar problemas técnicos que podem levar à falha do projeto de automação de teste e estratégias de mitigação do plano.

4.3 Manutenção da automação do teste

ALTA-E-4.3.1 (K2): Compreender quais fatores suportam e afetam a manutenção do TAS.

4.1 Seleção de abordagem de automação de teste e planejamento de implementação e implantação

Existem duas atividades principais envolvidas na implementação de um TAS: piloto e implantação. As etapas que compõem essas duas atividades variam dependendo do tipo de TAS e de situações específicas.

Para o piloto, pelo menos as seguintes etapas devem ser consideradas:

- Identificar um projeto adequado;
- Planejar o piloto;
- Conduzir o piloto;
- Avaliar o piloto.

Para implantação, pelo menos as seguintes etapas devem ser consideradas:

- Identificar o(s) projeto(s) inicial(is);
- Implantar o TAS nos projetos selecionados;
- Monitorar e avaliar o TAS nos projetos após um período pré-definido;
- Implementação dos demais projetos para o resto da organização.

4.1.1 Projeto Piloto

A implementação da ferramenta normalmente começa com um projeto piloto. O objetivo do projeto-piloto é garantir que o TAS possa ser utilizado para alcançar os benefícios planejados. Os objetivos do projeto piloto incluem:

- Conhecer mais detalhes sobre o TAS;
- Ver como o TAS se encaixa com os processos, procedimentos e ferramentas existentes; Identificar como eles podem mudar. (Geralmente, é preferível modificar o TAS para que ele se encaixe nos processos e procedimentos existentes, se estes precisam ser ajustados para "apoiar o TAS", isso deve ser, pelo menos, uma melhoria para os próprios processos);
- Projetar a interface de automação para atender às necessidades dos testadores;
- Decidir sobre formas padronizadas de utilização, gestão, armazenamento e manutenção do TAS e os ativos de teste, incluindo a integração com o gerenciamento de configuração e gerenciamento de mudança (p. ex., decidir sobre as convenções de nomenclatura para arquivos e testes, criação de bibliotecas e definir a modularidade dos conjuntos de testes);
- Identificar métricas e métodos de medição para monitorar a automação de teste em uso, incluindo usabilidade, manutenção e expansibilidade;
- Avaliar se os benefícios podem ser alcançados a um custo razoável. Esta será uma oportunidade para redefinir as expetativas uma vez que as TAS tenham sido utilizadas;
- Determinar quais habilidades são necessárias e quais estão disponíveis e quais estão faltando.

Identificar um projeto adequado

O projeto-piloto deve ser selecionado cuidadosamente usando as seguintes diretrizes:

- Não selecione um projeto crítico. Quando a implantação do TAS causa atraso, isso não deve ter grande impacto em projetos críticos. A implantação do TAS custará tempo no início. A equipe do projeto deve estar ciente disso;
- Não selecione um projeto trivial. Um projeto trivial não é um bom candidato, pois o sucesso da implantação não implica sucesso em projetos não triviais e, portanto, acrescenta menos às informações necessárias para a implantação;
- Envolver os stakeholders necessários (incluindo a gestão) no processo de seleção;
- O SUT do projeto-piloto deve ser uma boa referência para os outros projetos da organização, por exemplo, o SUT deve conter componentes GUI representativos que devem ser automatizados.

Planejar o piloto

O piloto deve ser tratado como um projeto de desenvolvimento comum: faça um plano, reserve orçamento e recursos, relate o progresso, defina marcos etc. Um ponto de atenção extra é para certificar-se de que as pessoas que trabalham na implantação do TAS podem gastar bastante esforço na implantação, mesmo quando outros projetos exigem recursos para suas atividades. É importante ter o compromisso da gerência, particularmente em todos os recursos compartilhados. Essas pessoas provavelmente não serão capazes de trabalhar em tempo integral na implantação.

Quando o TAS não foi adquirido por um fornecedor, mas desenvolvido internamente, os desenvolvedores correspondentes precisarão estar envolvidos nas atividades de implantação.

Conduzir o piloto

Execute o piloto da implantação e preste atenção aos seguintes pontos:

- O TAS fornece a funcionalidade conforme esperado (e prometido pelo fornecedor)? Caso contrário, isso precisa ser resolvido o mais rápido possível. Quando o TAS é desenvolvido internamente, os desenvolvedores correspondentes precisam auxiliar a implantação fornecendo qualquer funcionalidade ausente;
- O TAS e o processo existente se apoiam mutuamente? Se não, eles precisam ser alinhados.

Avaliar o piloto

Use todos os stakeholders para a avaliação.

4.1.2 Implantação

Uma vez que o piloto foi avaliado e considerado bem-sucedido, o TAS deve ser implantado para o restante do departamento ou organização. A implantação deve ser realizada de forma incremental e ser bem gerida. Os fatores de sucesso para implantação incluem:

- *Implantação incremental*: Execute a implantação para o resto da organização em etapas, em incrementos. Desta forma, o apoio aos novos usuários vem em "ondas" ao invés de tudo de uma vez. Isso permite usar o TAS para aumentar as etapas. Possíveis gargalos podem ser

identificados e resolvidos antes que se tornem problemas reais. As licenças podem ser adicionadas quando necessário;
- *Adaptação e aperfeiçoamento dos processos de acordo com o uso do TAS*: Quando diferentes usuários utilizam o TAS, diferentes processos entram em contato e precisam ser ajustados ao TAS, ou o TAS pode precisar de adaptações sutis ao seu processo;
- *Fornecimento de treinamento e coaching e mentoring para novos usuários*: Novos usuários precisam de treinamento e *coaching* no uso do novo TAS. Certifique-se de que este está no lugar. Treinamento e *workshops* devem ser fornecidos aos usuários antes que eles realmente usem o TAS;
- *Definição de diretrizes de uso*: É possível escrever diretrizes, listas de verificação e FAQs para o uso do TAS. Isso pode impedir perguntas de suporte extensas;
- *Implementando uma maneira de coletar informações sobre o uso real*: Deve haver uma maneira automatizada de coletar informações sobre o uso real do TAS. Não só o uso em si, mas também quais partes do TAS (certas funcionalidades) estão sendo utilizadas. Desta forma, seu uso pode ser facilmente monitorado;
- *Monitorando o uso, os benefícios e os custos do TAS*: Monitorar o uso do TAS durante um certo período indica se ele é realmente usado. Essas informações também podem ser usadas para recalcular o *business case* (p. ex., quanto tempo foi salvo, quantos problemas foram evitados);
- *Fornecimento de suporte para as equipes de teste e desenvolvimento para um determinado TAS*;
- *Recolhendo lições aprendidas de todas as equipes*: Realizar reuniões de avaliação e retrospectiva com as diferentes equipes que utilizam o TAS. Desta forma, as lições aprendidas podem ser identificadas. As equipes vão sentir que sua entrada é necessária e queriam melhorar o uso do TAS;
- *Identificação e implementação melhorias*: Com base no *feedback* da equipe e na monitorização do TAS, medidas de melhoria são identificadas e implementadas. Comunique também isso claramente aos stakeholders.

4.1.3 Implantação do TAS dentro do ciclo de vida do software

A implantação de um TAS depende muito da fase de desenvolvimento do projeto de software que será testado por ele.

Geralmente, um novo TAS ou uma nova versão dele, é implantado no início do projeto ou ao atingir um marco, como o congelamento de código ou o fim de um *sprint*. Isso ocorre porque as atividades de implantação, com todos os testes e modificações envolvidas, exigem tempo e esforço. Também esta é uma boa maneira de mitigar o risco do TAS não funcionar e causar interrupções no processo de automação de teste. No entanto, se houver problemas críticos que precisam ser corrigidos para o TAS ou se um componente do ambiente no qual ele é executado precisa ser substituído, em seguida, a implantação será feita independentemente da fase de desenvolvimento do SUT.

4.2 Avaliação do risco e estratégias de mitigação

Questões técnicas podem levar a riscos do produto ou do projeto. Questões técnicas típicas incluem:

- Demasiada abstração pode levar à dificuldade em compreender o que realmente acontece (p. ex., com palavras-chave);
- Dados: as tabelas de dados podem se tornar muito grandes, complexas ou pesadas;
- Dependência do TAS para usar determinadas bibliotecas do sistema operacional ou outros componentes que podem não estar disponíveis em todos os ambientes de destino do SUT.

Os riscos típicos do projeto de implantação incluem:

- Problemas de pessoal: conseguir as pessoas certas para manter a base de código pode ser difícil;
- Novos resultados do SUT podem fazer com que o TAS funcione incorretamente;
- Atrasos na introdução da automatização;
- Atrasos na atualização do TAS com base nas alterações feitas ao SUT;
- O TAS não consegue capturar os objetos (não padrão) que pretende.

Os possíveis pontos de falha do projeto TAS incluem:

- Migração para um ambiente diferente;
- Implantação para o ambiente de destino;
- Nova entrega do desenvolvimento.

Há uma série de estratégias de mitigação de risco que podem ser empregadas para lidar com essas áreas de risco. Estes são discutidos abaixo.

O TAS possui um ciclo de vida de software próprio, seja desenvolvido internamente ou uma solução adquirida. Uma coisa a lembrar é que o TAS, como qualquer outro software, precisa estar sob controle de versão e seus recursos documentados. Caso contrário, torna-se muito difícil implementar diferentes partes dele e fazê-los trabalhar juntos, ou trabalhar em determinados ambientes.

Além disso, um procedimento de implantação documentado, claro e fácil de seguir. Esse procedimento é dependente da versão. Portanto, ele deve ser incluído na versão de controle também.

Há dois casos distintos ao implementar um TAS:

1. Implementação inicial;
2. Implementação de manutenção - o TAS já existe e precisa ser mantido.

Antes de começar com a primeira implantação de um TAS, é importante ter certeza de que ele pode ser executado em seu próprio ambiente, isolado de alterações aleatórias e os casos de teste podem ser atualizados e gerenciados. Tanto o TAS como sua infraestrutura devem ser mantidos.

No caso de implantação pela primeira vez, são necessários os seguintes passos básicos:

- Definir a infraestrutura na qual o TAS será executado;
- Criar a infraestrutura para o TAS;
- Criar um procedimento para manter o TAS e sua infraestrutura;
- Criar um procedimento para manter o conjunto de testes que o TAS executará.

Os riscos relacionados à implantação pela primeira vez incluem:

- O tempo total de execução do conjunto de testes pode ser maior do que o tempo de execução planejado para o ciclo de teste. Neste caso, é importante certificar-se de que o conjunto de testes tenha tempo suficiente para ser executado inteiramente antes do início do próximo ciclo de teste agendado;
- Existem problemas de instalação e configuração com o ambiente de teste (p. ex., configuração do banco de dados e carga inicial, início e parada dos serviços). Em geral, o TAS precisa de uma maneira eficaz de configurar pré-requisitos necessários para os casos de teste automatizados dentro do ambiente de teste.

Para implantações de manutenção, há considerações adicionais. O TAS em si precisa evoluir, e as atualizações para ele precisam ser implantadas na produção. Antes de implantar uma versão atualizada do TAS em produção, ele precisa ser testado como qualquer outro software. Portanto, é necessário verificar a nova funcionalidade, para verificar se o conjunto de testes pode ser executado no TAS atualizado, que os relatórios podem ser enviados e que não há problemas de desempenho ou outras regressões funcionais. Em alguns casos, todo o conjunto de testes pode precisar ser alterado para se ajustar à nova versão do TAS.

Quando a implantação de manutenção ocorre, os seguintes passos são necessários:

- Fazer uma avaliação das mudanças na nova versão do TAS em relação ao antigo;
- Testar o TAS para novas funcionalidades e regressões;
- Verifique se o conjunto de testes precisa ser adaptado à nova versão do TAS.

Uma atualização também incorre os seguintes riscos e ações de mitigação correspondentes:

- O conjunto de testes precisa ser alterado para ser executado no TAS atualizado: faça as alterações necessárias ao conjunto de testes e teste-os antes de implantá-los no TAS;
- Os simuladores, controladores e interfaces usados nos testes precisam ser alterados para se ajustarem ao TAS atualizado: faça as alterações necessárias ao arnês de teste e teste-o antes de se implementar no TAS;
- A infraestrutura precisa mudar para acomodar o TAS atualizado: faça uma avaliação dos componentes de infraestrutura que precisam ser alterados, execute as alterações e teste-os com o TAS atualizado;
- O TAS atualizado apresenta defeitos ou problemas de desempenho adicionais: realize uma análise de riscos versus benefícios. Se os problemas descobertos tornam impossível atualizá-lo, talvez seja melhor não prosseguir ou esperar por uma próxima versão do TAS. Se os problemas são negligenciáveis em comparação com os benefícios, ele ainda pode ser atualizado. Certifique-se de criar uma nota de lançamento com problemas conhecidos para notificar o Engenheiro de Automação de Teste e outros stakeholders e tentar obter uma estimativa sobre quando os problemas serão corrigidos.

4.3 Manutenção da automação de teste

Desenvolver soluções de automação de teste não é trivial. Eles precisam ser modulares, escaláveis, compreensíveis, confiáveis e testáveis. Para adicionar ainda mais complexidade, soluções de automação de teste - como qualquer outro sistema de software, têm de evoluir. Seja devido a mudanças internas ou mudanças no ambiente em que operam, a manutenção é um aspeto

importante da arquitetura de um TAS. Manter o TAS adaptando-o a novos tipos de sistemas a serem testados, acomodando suporte para novos ambientes de software ou tornando-o compatível com novas leis e regulamentações, ajuda a garantir o funcionamento seguro e seguro do TAS. Também otimiza a vida útil e o desempenho do TAS.

4.3.1 Tipos de manutenção

A manutenção é feita em um TAS operacional através de modificações, migração ou aposentadoria do sistema. Este processo pode ser estruturado nas seguintes categorias:

- **Manutenção preventiva**: Mudanças são feitas para fazer o TAS suportar mais tipos de teste, testar em múltiplas interfaces, testar várias versões do SUT ou suportar a automação de teste para um novo SUT;
- **Manutenção corretiva**: São feitas alterações para corrigir falhas. A melhor maneira de manter um TAS em operação, reduzindo assim o risco de usá-lo, é através da execução de testes de manutenção regulares;
- **Manutenção perfeita**: O TAS é otimizado e os problemas não funcionais são fixos. Eles podem abordar o desempenho, sua usabilidade, robustez ou confiabilidade;
- **Manutenção adaptativa**: À medida que novos sistemas de software são lançados no mercado (sistemas operacionais, gerenciadores de banco de dados, navegadores da Web etc.), pode ser necessário que o TAS passe a suportá-los. Além disso, pode ser o caso em que o TAS precisa cumprir com novas leis, regulamentos ou requisitos específicos do setor. Neste caso, são feitas alterações ao TAS para que ele se adapte adequadamente. Nota: normalmente, a conformidade com leis e regulamentos cria manutenção obrigatória com regras específicas, requisitos e, por vezes, requisitos de auditoria. Além disso, à medida que as ferramentas de integração são atualizadas e novas versões criadas, os pontos de extremidade de integração de ferramentas precisam ser mantidos e mantidos funcionais.

4.3.2 Escopo e abordagem

A manutenção é um processo que pode afetar todas as camadas e componentes de um TAS. O escopo dela depende de:

- Tamanho e complexidade do TAS;
- Tamanho da mudança;
- Risco da mudança.

Dado o fato de que a manutenção se refere ao TAS em operação, uma análise de impacto é necessária para determinar como o sistema pode ser afetado pelas mudanças. Dependendo do impacto, as mudanças precisam ser introduzidas incrementalmente e os testes precisam ser realizados após cada etapa para garantir o funcionamento contínuo do TAS. Observação: manter o TAS pode ser difícil se suas especificações e documentação estiverem desatualizadas.

Como a eficiência do tempo é o principal fator que contribui para o sucesso da automação de testes, torna-se fundamental ter boas práticas para manter o TAS, incluindo:

- Procedimentos de implantação e o uso do TAS devem ser claros e documentados;

- Dependências de terceiros devem ser documentadas, juntamente com desvantagens e problemas conhecidos;
- Deve ser modular, de forma que partes dele possam ser facilmente substituídas;
- Deve ser executado em um ambiente substituível ou com componentes substituíveis;
- Deve separar scripts de teste do próprio TAF;
- Deve ser executado isoladamente do ambiente de desenvolvimento, de modo que as alterações ao TAS não afetarão adversamente o ambiente de teste;
- O TAS, o ambiente, o conjunto de testes e os artefatos do *testware*, devem estar sob gerenciamento de configuração.

Há também considerações para a manutenção dos componentes de terceiros e outras bibliotecas da seguinte maneira:

- Muitas vezes é o caso que o TAS irá utilizar componentes de terceiros para executar os testes. Também pode ser o caso de o TAS depender de bibliotecas de terceiros (p. ex., as bibliotecas de automação de UI). Todas as partes componentes de terceiros do TAS devem ser documentadas e sob gerenciamento de configuração;
- É necessário ter um plano no caso destes componentes externos precisarem de modificação ou correção. A pessoa responsável pela manutenção do TAS precisa saber com quem deve entrar em contato ou para onde enviar um problema;
- Deve haver documentação relativa à licença sob a qual os componentes de terceiros são usados, para que haja informações sobre se podem ser modificados, até que ponto e por quem;
- Para cada um dos componentes de terceiros, é necessário obter informações sobre atualizações e novas versões. Manter os componentes e bibliotecas de terceiros atualizados é uma ação preventiva que compensa o investimento a longo prazo.

Considerações sobre padrões de nomenclatura e outras convenções incluem:

- A ideia de nomear padrões e outras convenções têm uma razão simples: o conjunto de testes e o próprio TAS devem de fácil leitura, entendimento, mudança e guarda. Isso economiza tempo no processo de manutenção e minimiza o risco de introduzir regressões ou correções erradas que poderiam ser facilmente evitadas;
- É mais fácil introduzir novas pessoas no projeto de automação de teste quando são usadas convenções de nomenclatura padrão;
- Os padrões de nomeação podem se referir a variáveis e arquivos, cenários de teste, palavras-chave e parâmetros de palavras-chave. Outras convenções referem-se aos pré-requisitos e pós-ações para a execução do teste, o conteúdo dos dados de teste, o ambiente de teste, o status da execução do teste e os logs e relatórios de execução;
- Todas as normas e convenções devem ser acordadas e documentadas ao iniciar um projeto de automação de teste.

Considerações sobre documentação incluem:

- A necessidade de documentação boa e atual para os cenários de teste e o TAS é bastante clara, mas há duas questões relacionadas a isso: alguém tem que escrevê-lo e alguém tem que mantê-lo;

- Enquanto o código da ferramenta de teste pode ser documentado automaticamente ou semi automaticamente, todo o projeto, componentes, integrações com terceiros, dependências e procedimentos de implantação precisam ser documentados por alguém;
- É uma boa prática introduzir a redação de documentação como parte do processo de desenvolvimento. Uma tarefa não deve ser considerada como feita a menos que seja documentada ou a documentação seja atualizada.

As considerações sobre o material de treinamento incluem:

- Se a documentação para o TAS estiver bem escrita, ela pode ser usada como base para o material de treinamento;
- O material de treinamento é uma combinação de especificações funcionais, modelagem e arquitetura, implantação, manutenção, uso (manual do usuário), exemplos práticos, exercícios, dicas e truques do TAS;
- A manutenção do material de treinamento consiste em inicialmente escrevê-lo e depois revê-lo periodicamente. Isso é feito na prática pelos membros da equipe designados como treinadores no TAS e é mais provável que aconteça no final de uma iteração do ciclo de vida do SUT (no final dos *sprints*, por exemplo).

5 Relatório de automação de testes e métricas [165 min]

Palavras-chave

Densidade de defeitos de código de automação, cobertura, matriz de rastreabilidade, esforço de teste manual equivalente, métricas, log de teste, relatórios de teste

Objetivos de aprendizagem

5.1 Seleção de métricas TAS

ALTA-E-5.1.1 (K2): Classificar métricas que podem ser usadas para monitorar a estratégia e a eficácia da automação de teste.

5.2 Implementação da medição

ALTA-E-5.2.1 (K3): Implementar métodos de coleta de métricas para suportar requisitos técnicos e de gerenciamento. Explicar como a medição da automação de teste pode ser implementada.

5.3 Registro do TAS e do SUT

ALTA-E-5.3.1 (K4): Analisar o registro de teste dos dados do TAS e do SUT.

5.4 Relatórios de automação de testes

ALTA-E-5.4.1 (K2): Explicar como um relatório de execução de teste é construído e publicado.

5.1 Seleção de métricas TAS

Esta seção se concentra nas métricas que podem ser usadas para monitorar a estratégia de automação de teste, a eficácia e a eficiência do TAS. Estas são separadas das métricas relacionadas ao SUT usadas para monitorar o SUT e os seus testes (funcional e não funcional). Esses são selecionados pelo gerente de teste geral do projeto. As métricas de automação de teste permitem que o TAM e o TAE acompanhem o progresso em direção aos objetivos da automação e monitorem o impacto das alterações feitas na solução de automação de teste.

As métricas do TAS podem ser divididas em dois grupos: externo e interno. As métricas externas são aquelas usadas para medir o impacto do TAS em outras atividades (em especial as atividades de teste). As métricas internas são aquelas utilizadas para medir a eficácia e a eficiência do TAS no cumprimento dos seus objetivos.

As métricas do TAS incluem tipicamente:

- Métricas externas
  - Benefícios de automação;
  - Esforço para construir testes automatizados;
  - Esforço para analisar incidentes de teste automatizados;
  - Esforço para manter testes automatizados;
  - Relação entre falhas e defeitos;
  - Tempo de execução dos testes automatizados;
  - Número de casos de teste automatizados;
  - Número de resultados de aprovados e falha;
  - Número de falhas falsas e resultados de falso-positivo;
  - Cobertura de código.
- Métricas internas
  - Métricas de script de ferramenta;
  - Densidade de defeitos de código de automação;
  - Velocidade e eficiência dos componentes TAS.

Estes são descritos a seguir.

Benefícios de automação

É particularmente importante medir e relatar os benefícios de um TAS. Isso ocorre porque os custos (em termos do número de pessoas envolvidas ao longo de um determinado período) são fáceis de ver. As pessoas que trabalham fora dos testes serão capazes de formar uma impressão do custo total, mas podem não ver os benefícios alcançados.

Qualquer medida de benefício dependerá do objetivo do TAS. Normalmente, isso pode ser uma economia de tempo ou esforço, um aumento na quantidade de testes realizados (amplitude ou profundidade de cobertura, ou frequência de execução) ou alguma outra vantagem, como aumento da repetibilidade, maior uso de recursos ou menos erros manuais.

As medidas possíveis incluem:

- Número de horas de esforço de teste manual economizado;
- Redução do tempo para realizar testes de regressão;
- Número de ciclos adicionais de execução de teste alcançados;
- Número ou porcentagem de testes adicionais realizados;
- Percentagem de casos de teste automatizados relacionados com todo o conjunto de casos de teste (embora automatizado não possa ser facilmente comparado com casos de teste manual)
- Aumento da cobertura (requisitos, funcionalidade, estruturais);
- Número de defeitos encontrados anteriormente por causa do TAS (quando o benefício médio de defeitos encontrado anteriormente é conhecido, isso pode ser "calculado" para uma soma dos custos evitados);
- Número de defeitos encontrados por causa do TAS que não teria sido encontrado por testes manuais (p. ex., defeitos de fiabilidade).

Observe que a automação de teste geralmente economiza esforço de teste manual. Este esforço pode ser dedicado a outros tipos de teste manual como por exemplo, testes exploratórios. Os defeitos encontrados por esses testes adicionais também podem ser vistos como benefícios indiretos do TAS, pois a automação do teste permitiu que esses testes manuais fossem executados. Sem o TAS estes testes não teriam sido executados e, posteriormente, os defeitos adicionais não teriam sido encontrados.

Esforço para construir testes automatizados

O esforço para automatizar testes é um dos principais custos associados à automação de teste. Este é muitas vezes mais do que o custo de executar o mesmo teste manualmente e, portanto, pode ser um detrimento para expandir o uso de automação de teste. Embora o custo de implementar um teste automatizado específico dependa em grande parte do próprio teste, outros fatores, como a abordagem de script usada, a familiaridade com a ferramenta de teste, o ambiente e o nível de habilidade do Engenheiro de Automação de Teste também terão impacto.

Como os testes maiores ou mais complexos normalmente levam mais tempo para automatizar do que testes simples ou curtos, a computação do custo de construção para a automação de teste pode ser baseada em um tempo médio de compilação. Isto pode ser refinado adicionalmente considerando o custo médio para um conjunto específico de testes, tais como aqueles que visam a mesma função ou aqueles em um determinado nível de teste. Outra abordagem é expressar o custo de construção como um fator do esforço necessário para executar o teste manualmente (esforço de teste manual equivalente, EMTE). Por exemplo, pode ser que leva o dobro do esforço de teste manual para automatizar um caso de teste, ou duas vezes o EMTE.

Esforço para analisar falhas SUT

Analisar falhas descobertas no SUT através da execução de testes automatizados pode ser significativamente mais complexo do que para um teste executado manualmente porque os eventos que antecederam a falha de um teste manual são frequentemente conhecidos pelo testador executando o teste. Isso pode ser atenuado conforme descrito no nível de projeto no Capítulo 3.1.4 e no nível de relatório nos Capítulos 5.3 e 5.4. Esta medida pode ser expressa como uma média por

caso de teste falhado ou pode ser expressa como um fator de EMTE. Este último sendo particularmente adequado onde os testes automatizados variam significativamente em complexidade e comprimento de execução.

O registro disponível do SUT e do TAS desempenha um papel crucial na análise de falhas. O registro deve fornecer informações suficientes para realizar essa análise de forma eficiente.

Os recursos de registro importantes incluem:

- Registro de SUT e registro de TAS devem ser sincronizados;
- O TAS deve registrar o comportamento esperado e real;
- O TAS deve registrar as ações a serem executadas.

O SUT, por outro lado, deve registrar todas as ações que são executadas (independentemente de a ação ser resultado de testes manuais ou automatizados). Quaisquer erros internos devem ser registrados e qualquer despejo de memória e rastreamentos de pilha devem estar disponíveis.

Esforço para manter testes automatizados

O esforço de manutenção necessário para manter os testes automatizados em sincronia com o SUT pode ser significativo e, em última instância, pode superar os benefícios obtidos pelo TAS. Esta tem sido a causa da falha para muitos esforços de automação. O monitoramento do esforço de manutenção é, portanto, importante para destacar quando é necessário tomar medidas para reduzir o esforço de manutenção ou, pelo menos, evitar que ele cresça sem controle.

Medidas de esforço de manutenção podem ser expressas como um total para todos os testes automatizados que requerem manutenção para cada nova versão do SUT. Eles também podem ser expressos como uma média por teste automatizado atualizado ou como um fator de EMTE. Uma métrica relacionada é o número ou a porcentagem de testes que exigem manutenção.

Quando o esforço de manutenção para testes automatizados é conhecido (ou pode ser derivado), esta informação pode desempenhar um papel crucial na decisão de implementar ou não determinadas funcionalidades ou de corrigir um defeito específico. O esforço necessário para manter o caso de teste ao software alterado deve ser considerado com a mudança do SUT.

Relação entre falhas e defeitos

Um problema comum com testes automatizados é que muitos deles podem falhar pela mesma razão - um único defeito no software. Enquanto a finalidade dos testes é destacar defeitos no software, tendo mais de um teste destacar o mesmo defeito é um desperdício. Este é particularmente o caso para testes automatizados, pois o esforço necessário para analisar cada teste falhado pode ser significativo. Medir o número de testes automatizados que falham para um determinado defeito pode ajudar a indicar onde isso pode ser um problema. A solução reside na concepção dos testes automatizados e na sua seleção para execução.

Tempo para executar testes automatizados

Uma das métricas mais fáceis de determinar é o tempo que leva para executar os testes automatizados. No início do TAS isso pode não ser importante, mas como o número de casos de teste automatizado aumenta, essa métrica pode se tornar muito importante.

Número de casos de teste automatizados

Essa métrica pode ser usada para mostrar a progressão feita pelo projeto de automação de teste. Mas é preciso levar em conta que apenas o número de casos de teste automatizados não revela muita informação. Por exemplo, não indica que a cobertura do teste tenha aumentado.

Número de resultados de aprovação e falha

Esta é uma métrica comum e rastreia quantos testes automatizados foram passados e quantos não conseguiram alcançar o resultado esperado. As falhas têm que ser analisadas para determinar se a falha foi devido a um defeito no SUT ou foi devido a questões externas, como um problema com o ambiente ou com o próprio TAS.

Número de falhas falsas e resultados de falsos passes

Como foi visto em várias métricas anteriores, pode levar algum tempo para analisar falhas de teste. Isso é ainda mais frustrante quando se trata de um falso alarme. Isso acontece quando o problema está no TAS ou caso de teste, mas não no SUT. É importante que o número de alarmes falsos (e o esforço potencialmente desperdiçado) sejam mantidos baixos. Falsas falhas podem reduzir a confiança no TAS. Por outro lado, os resultados de falsos positivos podem ser mais perigosos. Quando um passo falso ocorre, houve uma falha no SUT, mas não foi identificado pela automação de teste para um resultado relatado. Neste caso, um defeito potencial pode escapar à detecção. Isso pode ocorrer porque a verificação do resultado não foi feita corretamente, um oráculo de teste inválido foi usado ou o caso de teste estava esperando o resultado errado.

Observe que os alarmes falsos podem ser causados por defeitos no código de teste (consulte a métrica "Densidade do defeito do código de automação"), mas também pode ser causado por um SUT instável que está se comportando de maneira imprevisível (p. ex., tempo limite). Os ganchos de teste também podem causar falsos alarmes devido ao nível de intrusão que estão causando.

Cobertura de código

Conhecer a cobertura de código SUT fornecida pelos diferentes casos de teste pode revelar informações úteis. Isto também pode ser medido a um nível elevado, por exemplo, a cobertura de código do conjunto de testes de regressão. Não há percentual absoluto que indique uma cobertura adequada e 100% de cobertura de código é inatingível em qualquer outra coisa que não seja a mais simples das aplicações de software. No entanto, é geralmente acordado que quanto mais cobertura melhor, uma vez que reduz o risco global de implantação de software. Essa métrica também pode indicar atividade no SUT. Por exemplo, se a cobertura de código cair, isso provavelmente significa que a funcionalidade foi adicionada ao SUT, mas nenhum caso de teste correspondente foi adicionado ao conjunto de testes automatizado.

Métricas de script de ferramenta

Existem muitas métricas que podem ser usadas para monitorar o desenvolvimento de scripts de automação. Em sua maioria são semelhantes às métricas do código fonte para o SUT. Linhas de código e complexidade ciclomática podem ser usadas para destacar scripts excessivamente grandes ou complexos (sugerindo que é necessária uma remodelagem).

A proporção de comentários para instruções executáveis pode ser usada para dar uma possível indicação da extensão da documentação de script. O número de não-conformidades com os padrões de script pode dar uma indicação do grau em que essas normas estão sendo seguidas.

Densidade de defeitos de código de automação

O código de automação não é diferente do código do SUT, pois é um software e conterá defeitos. O código de automação não deve ser considerado menos importante do que o código SUT. Boas práticas de codificação e padrões devem ser aplicadas e o resultado destes monitorados por métricas, como a densidade de defeitos de código. Estes serão mais fáceis de coletar com o suporte de um sistema de gerenciamento de configuração.

Velocidade e eficiência dos componentes TAS

Diferenças no tempo necessário para executar as mesmas etapas de teste no mesmo ambiente podem indicar um problema no SUT. Se o SUT não estiver executando a mesma funcionalidade no mesmo tempo decorrido, a investigação é necessária. Isso pode indicar uma variabilidade no sistema que não é aceitável e que poderia piorar com o aumento da carga O TAS precisa estar funcionando bem o suficiente para que não prejudique o desempenho de SUT. Se o desempenho é requisito crítico para o SUT, então o TAS precisa ser projetado de uma forma que leva isso em conta.

Métricas de tendência

Com muitas métricas, são as tendências (ou seja, a forma como as medidas mudam ao longo do tempo) que podem ser as mais valiosas em relatar o que o valor de uma medida em um momento específico representa. Por exemplo, sabendo que o custo médio de manutenção por teste automatizado que exige manutenção é mais do que era para os dois lançamentos anteriores do SUT, pode levar a ação em determinar a causa do aumento e empreender medidas para reverter a tendência.

O custo da medição deve ser o mais baixo possível e isso pode ser conseguido com a automatização da coleta e geração de relatórios.

5.2 Implementação da medição

Uma vez que uma estratégia de automação de teste tem o *testware* automatizado no seu núcleo, o *testware* automatizado pode ser aprimorado para registrar informações sobre seu uso. Onde a abstração é combinada com *testware* estruturado, quaisquer aprimoramentos feitos subjacentes a ele podem ser utilizados por todos os scripts de teste automatizados de nível mais alto. Por exemplo,

o aprimoramento do subjacente ao *testware* para registrar o início e o fim da execução de um teste pode muito bem se aplicar a todos os demais testes.

Recursos de automação que suportam medição e geração de relatórios

As linguagens de script de muitas ferramentas de teste suportam medições e relatórios através de recursos que podem ser usados para registrar informações antes, durante e após a execução de testes individuais, conjuntos de testes e a suíte de testes completa.

O relato de cada uma das séries de testes precisa ter um recurso de análise para levar em conta os resultados dos testes anteriores, para que ele possa destacar tendências (como mudanças na taxa de sucesso do teste).

A automatização dos testes normalmente requer a automação tanto da execução do teste como da verificação do teste, sendo esta última obtida pela comparação de elementos específicos do resultado do teste com um resultado esperado predefinido. Esta comparação é geralmente mais bem realizada por uma ferramenta de teste. O nível de informação que é relatado como resultado desta comparação deve ser considerado. É importante que o status do teste seja determinado corretamente (p. ex., passar, falhar). No caso de um estado com falha, serão necessárias maiores informações sobre a causa da falha (p. ex., capturas de tela).

Distinguir entre as diferenças esperadas no resultado real e esperado de um teste nem sempre é trivial, embora o suporte à ferramenta possa ajudar muito na definição de comparações que ignoram as diferenças esperadas (como datas e horas) ao mesmo tempo em que destacam quaisquer diferenças inesperadas.

Integração com outras ferramentas de terceiros (planilhas, XML, documentos, bancos de dados, ferramentas de relatório etc.)

Quando informações da execução de casos de teste automatizados são usadas em outras ferramentas (para rastreamento e geração de relatórios, por exemplo, atualização da matriz de rastreabilidade), é possível fornecer informações no formato adequado a essas ferramentas de terceiros. Isso geralmente é obtido através da funcionalidade da ferramenta de teste existente (formatos de exportação para relatórios) ou criando relatórios personalizados que são exibidos em um formato consistente com outros programas (".xls" para Excel, ".doc" para Word, ".html" para Web etc.).

Visualização de resultados (painéis de controle, gráficos etc.)

Os resultados dos testes devem ser visíveis em gráficos. Considere o uso de cores para indicar problemas na execução do teste, como luzes de trânsito, para indicar o progresso da execução do teste e automação, para que as decisões possam ser feitas com base em informações relatadas. A gerência está particularmente interessada em resumos visuais para ver o resultado do teste de uma só vez. No caso de ser necessário mais informações, eles ainda podem mergulhar nos detalhes.

5.3 Registro do TAS e do SUT

O registro é muito importante no TAS, incluindo o registro tanto para a automação de teste como para o SUT. Os logs de teste são uma fonte que são frequentemente usados para analisar problemas potenciais. Na seção a seguir estão exemplos de log de teste, categorizados por TAS ou SUT.

O registro TAS (se o TAF ou o próprio caso de teste registra as informações não é tão importante e depende do contexto) deve incluir o seguinte:

- Que caso de teste está atualmente em execução, incluindo hora de início e término;
- O status da execução do caso de teste porque, embora as falhas possam ser facilmente identificadas em arquivos de log, o próprio framework também deve ter essas informações e deve reportar através de um painel de controle. O status de execução do caso de teste pode ser erro de passagem, falha ou TAS. O resultado do erro do TAS é usado para situações em que o problema não está no SUT;
- Detalhes do log de teste de alto nível (registrando etapas 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 e falhas reais dessas medidas dinâmicas devem ser registrados com o caso de teste que estava sendo executado quando o incidente foi detectado;
- No caso de testes de confiabilidade ou stress (onde são realizados numerosos ciclos), um contador deve ser registrado, de modo que possa ser facilmente determinado quantas vezes os casos de teste foram executados;
- Quando os casos de teste tiverem partes aleatórias (p. ex., parâmetros aleatórios ou passos aleatórios em testes de máquina de estado), o número aleatório das escolhas deve ser registrado;
- Todas as ações executadas por um caso de teste devem ser registradas de tal forma que o arquivo de log (ou partes dele) possa ser reproduzido para reexecutar o teste com exatamente os mesmos passos e o mesmo tempo. Isso é útil para verificar a reprodutibilidade de uma falha identificada e para capturar informações adicionais. As informações de ação de caso de teste também podem ser registradas no próprio SUT para uso ao reproduzir problemas identificados pelo cliente (o cliente executa o cenário, a informação de log é capturada e pode ser reproduzida pela equipe de desenvolvimento ao solucionar o problema);
- Capturas de tela e outras capturas visuais podem ser salvas durante a execução do teste para uso posterior durante a análise de falha;
- Sempre que um caso de teste encontrar uma falha, o TAS deve certificar-se de que todas as informações necessárias para analisar o problema estão disponíveis e armazenadas, bem como quaisquer informações sobre a continuação dos testes, se for aplicável. Quaisquer despejos de colisão associados e rastreamentos de pilha devem ser salvos pelo TAS em um local seguro. Além disso, todos os arquivos de log que podem ser sobrescritos (os buffers cíclicos são frequentemente usados para arquivos de log no SUT) devem ser copiados para este local onde estarão disponíveis para análise posterior;

- O uso de cores pode ajudar a distinguir diferentes tipos de informações registradas (p. ex., erros em vermelho, informações de progresso em verde);

Registro SUT:

- Quando o SUT identifica um problema, todas as informações necessárias para analisar o problema devem ser registradas, incluindo carimbos de data e hora, local de origem do problema, mensagens de erro etc.
- O SUT pode registar toda a interação do utilizador (diretamente através da interface de utilizador disponível, mas também através de interfaces de rede etc.). Desta forma, os problemas identificados pelos clientes podem ser analisados adequadamente e o desenvolvimento pode tentar reproduzir o problema.
- Na inicialização do sistema, as informações de configuração devem ser registradas em um arquivo, consistindo nas diferentes versões de software, firmware, configuração do SUT, configuração do sistema operacional etc.

Todas as informações de registro diferentes devem ser facilmente pesquisáveis. Um problema identificado no arquivo de log pelo TAS deve ser facilmente identificado no arquivo de log do SUT, e vice-versa (com ou sem ferramentas adicionais). Sincronizar vários logs com um carimbo de data-hora facilita a correlação do que ocorreu quando um erro foi relatado.

5.4 Relatório de Automação de Testes

Os logs de teste fornecem informações detalhadas sobre as etapas de execução, ações e respostas de um caso de teste e/ou conjunto de testes. No entanto, apenas os logs não podem fornecer uma boa visão geral do resultado de execução geral. Para isso, é necessário ter funcionalidade de relatórios no local. Após cada execução do conjunto de testes, um relatório conciso deve ser criado e publicado. Um componente gerador de relatório reutilizável poderia ser usado para isso.

Conteúdo dos relatórios

O relatório de execução do teste deve conter um resumo que dê uma visão geral dos resultados da execução, do sistema que está sendo testado e do ambiente em que os testes foram executados, o que é apropriado para cada um dos stakeholders.

É necessário saber quais testes falharam e as razões para a falha. Para facilitar a resolução de problemas, é importante conhecer o histórico da execução do teste e quem é responsável por ele (geralmente a pessoa que criou ou atualizou o último). A pessoa responsável precisa investigar a causa da falha, relatar as questões relacionadas a ela, fazer o acompanhamento da correção dos problemas e verificar se foi corretamente implementada.

Os relatórios também são usados para diagnosticar falhas dos componentes do TAF (consulte o Capítulo 7).

Publicação dos relatórios

O relatório deve ser publicado para todos os interessados nos resultados da execução. Ele pode ser carregado em um site, enviado para uma lista de discussão ou carregado em outra ferramenta, como

uma ferramenta de gerenciamento de teste. De um lado prático, é mais provável que os interessados no resultado da execução irão olhar para ele e analisá-lo se lhes for dada a facilidade de subscrição e pode receber o relatório por e-mail.

Opção é identificar partes problemáticas do SUT, manter um histórico dos relatórios, de modo que as estatísticas sobre casos de teste ou *suites* de teste com regressões frequentes podem ser reunidos.

6 Transição de teste manual para um ambiente automatizado [120 min]

Palavras-chave

Teste de confirmação, teste de regressão

Objetivos de aprendizagem

6.1 Critério para automação

ALTA-E-6.1.1 (K3): Aplicar critérios para determinar a adequação dos testes de automação

ALTA-E-6.1.2 (K2): Compreender os fatores na transição de testes manuais para automação

6.2 Identificar etapas necessárias para implementar a automação em testes de regressão

ALTA-E-6.2.1 (K2): Explicar os fatores a serem considerados na implementação de testes de regressão automatizada

6.3 Fatores a serem considerados ao implementar a automação no teste de novos recursos

ALTA-E-6.3.1 (K2): Explicar os fatores a serem considerados na implementação da automação no teste de novos recursos

6.4 Fatores a considerar ao Implementar Automação de Testes de Confirmação

ALTA-E-6.4.1 (K2): Explicar os fatores a serem considerados na implementação do teste de confirmação automatizado

6.1 Critérios para automação

Tradicionalmente, as organizações desenvolvem casos de teste manual. Ao decidir migrar para um ambiente de teste automatizado, é preciso avaliar o estado atual dos testes manuais e determinar a abordagem mais eficaz para automatizar esses ativos de teste. A estrutura existente de um teste manual pode ou não ser adequada para automação, caso em que pode ser necessária uma reescrita completa do teste para suportar a automação. Alternativamente, componentes relevantes de testes manuais existentes (p. ex., valores de entrada, resultados esperados, caminho de navegação) podem ser extraídos de testes manuais existentes e reutilizados para automação. Uma estratégia de teste manual que leva em consideração a automação permitirá testes cuja estrutura facilita a migração para a automação.

Nem todos os testes podem ou devem ser automatizados, e às vezes a primeira iteração de um teste pode ser manual. Portanto, há dois aspetos da transição a considerar: a conversão inicial de testes manuais existentes para automação e a subsequente transição de novos testes manuais para automação.

Observe também que certos tipos de teste só podem ser executados (efetivamente) de forma automatizada, por exemplo, testes de confiabilidade, testes de estresse ou testes de desempenho.

Com a automação de teste é possível testar aplicações e sistemas sem uma interface de usuário. Neste caso, o teste pode ser feito no nível de integração através de interfaces no software. Embora esses tipos de casos de teste também possam ser executados manualmente (usando comandos inseridos manualmente para acionar as interfaces), isso pode não ser prático. Por exemplo, com automação pode ser possível inserir mensagens em um sistema de fila de mensagens. Desta forma, o teste pode começar mais cedo (e pode identificar defeitos anteriormente), quando o teste manual ainda não é possível.

Antes de iniciar um esforço de teste automatizado, é preciso considerar a aplicabilidade e a viabilidade de criar testes automatizados versus testes manuais. Os critérios de aptidão podem incluir, mas não se limitam a:

- Frequência de uso;
- Complexidade para automatizar;
- Compatibilidade e suporte de ferramentas;
- Maturidade do processo de teste;
- Adequação da automação para a fase do ciclo de vida do produto de software;
- Sustentabilidade do ambiente automatizado;
- Controle do SUT;
- Planejamento técnico em apoio à análise do ROI.

Cada um deles é explicado em mais detalhes abaixo.

Frequência de uso

Quantas vezes um teste precisa ser executado é uma consideração quanto à viabilidade de se automatizar ou não. Os testes que são executados mais regularmente, como parte de um ciclo de lançamento maior ou menor, são melhores candidatos para automação, pois serão

usados com mais frequência. Como regra geral, quanto maior o número de lançamentos de aplicativos e, portanto, os ciclos de teste correspondentes, maior será o benefício de testes automatizados.

À medida que os testes funcionais se tornam automatizados, eles podem ser usados em versões subsequentes como parte dos testes de regressão. Testes automatizados usados em testes de regressão fornecerão alto retorno sobre o investimento (ROI) e mitigação de risco para a base de código existente.

Se um script de teste for executado uma vez por ano e o SUT for alterado dentro do ano, pode não ser viável ou eficiente criar um teste automatizado. O tempo que pode levar para adaptar o teste numa base anual para se conformar ao SUT pode ser melhor feito manualmente.

Complexidade para automatizar

Nos casos em que um sistema complexo precisa ser testado, pode haver um tremendo benefício da automação para poupar o testador manual a tarefa difícil de ter que repetir etapas complexas que são tediosas, demoradas e propensas a executar erros.

No entanto, certos scripts de teste podem ser difíceis ou não rentáveis para automatizar. Existe uma série de fatores que podem afetar isso, incluindo: um SUT que não é compatível com as soluções de teste automatizadas disponíveis existentes; o requisito de produzir código de programa substancial e desenvolver chamadas para API para automatizar; a multiplicidade de sistemas que precisam ser abordados como parte de uma execução de teste; a interação com interfaces externas e/ou sistemas proprietários; alguns aspetos do teste de usabilidade; a quantidade de tempo necessária para testar os scripts de automação etc.

Compatibilidade e suporte de ferramentas

Existe uma ampla gama de plataformas de desenvolvimento usadas para criar aplicações. O desafio para o testador é saber quais as ferramentas de teste disponíveis (se houver) para suportar qualquer plataforma, e até que ponto a plataforma é suportada. As organizações usam uma variedade de ferramentas de teste, incluindo aquelas de fornecedores comerciais, código aberto e internamente desenvolvidas. Cada organização terá diferentes necessidades e recursos para suportar ferramentas de teste. Os vendedores comerciais geralmente oferecem suporte pago e, no caso dos líderes de mercado, geralmente têm um ecossistema de especialistas que podem ajudar com a implementação da ferramenta de teste. Ferramentas de código aberto podem oferecer suporte, como fóruns on-line a partir do qual os usuários podem obter informações e postar perguntas. Ferramentas de teste desenvolvidas internamente confiam no pessoal existente para fornecer suporte.

A questão da compatibilidade da ferramenta de teste não deve ser subestimada. Embarcar em um projeto de automação de teste sem entender totalmente o nível de compatibilidade entre as ferramentas de teste e o SUT pode ter resultados desastrosos. Mesmo se a maioria dos testes para o SUT pode ser automatizada, pode haver a situação em que os testes mais críticos não podem.

Maturidade do processo de teste

Para efetivamente implementar a automação dentro de um processo de teste, esse processo deve ser estruturado, disciplinado e repetível. A automação traz todo um processo de desenvolvimento para o processo de teste existente, que requer o gerenciamento do código de automação e dos componentes relacionados.

Adequação da automação para a fase do ciclo de vida do produto de software

Um SUT tem um ciclo de vida do produto que pode ir de anos a décadas. À medida que o desenvolvimento de um sistema começa, o sistema muda e se expande para endereçar defeitos e adicionar refinamentos para atender às necessidades do usuário final. Nos estágios iniciais do desenvolvimento de um sistema, a mudança pode ser muito rápida para implementar uma solução de teste automatizada. Como os layouts e controles de tela são otimizados e aprimorados, a criação de automação em um ambiente dinâmico pode exigir um trabalho contínuo, o que não é eficiente ou eficaz. Isso seria semelhante a tentar mudar um pneu em um carro em movimento. É melhor esperar que o carro para parar. Para sistemas grandes em um ambiente de desenvolvimento sequencial, quando um sistema se estabilizou e inclui um núcleo de funcionalidade, isso se torna o melhor momento para iniciar a implementação de testes automatizados.

Com o tempo, os sistemas chegam ao fim de seus ciclos de vida e são aposentados ou redesenhados para usar tecnologia mais recente e mais eficiente. A automação não é recomendada para um sistema próximo ao fim de seu ciclo de vida, pois haverá pouco valor na realização de uma iniciativa de tão curta duração. No entanto, para sistemas que estão sendo redesenhados usando uma arquitetura diferente, mas preservando a funcionalidade existente, um ambiente de teste automatizado que define elementos de dados será igualmente útil nos sistemas antigos e novos. Neste caso, a reutilização de dados de teste seria possível e a recodificação do ambiente automatizado para ser compatível com a nova arquitetura seria necessária.

Sustentabilidade do ambiente automatizado

Um ambiente de teste para automação precisa ser flexível e adaptável às mudanças que ocorrerão ao SUT ao longo do tempo. Isso inclui a capacidade de diagnosticar e corrigir rapidamente problemas de automação, a facilidade com que os componentes de automação podem ser mantidos e a facilidade com a qual novos recursos e suporte podem ser adicionados ao ambiente automatizado. Esses atributos são parte integrante do projeto geral e implementação do gTAA.

Controle do SUT (pré-condições, configuração e estabilidade)

O TAE deve identificar caraterísticas de controle e visibilidade no SUT que ajudarão na criação de testes automatizados efetivos. Caso contrário, a automação de teste depende apenas de interações interface do usuário, resultando em uma solução de automação de teste menos mantida. Consulte o Capítulo 2.3 sobre modelagem para testabilidade e automação para obter mais informações.

Planejamento técnico em apoio à análise do ROI

A automação de teste pode fornecer vários graus de benefício para uma equipe de teste. No entanto, um nível significativo de esforço e custo está associado à implementação de uma solução de teste automatizada e eficaz. Antes de incorrer o tempo e o esforço para desenvolver testes automatizados, uma avaliação deve ser realizada para verificar o que o potencial pretendido e os resultados da implementação de automação de teste podem ser. Uma vez determinado isso, as atividades necessárias para efetuar tal plano devem ser definidas e os custos associados devem ser usados para calcular o ROI.

Para se preparar adequadamente a transição para um ambiente automatizado, as seguintes áreas precisam ser tratadas:

- Disponibilidade de ferramentas no ambiente de teste para automação de teste;
- Correção de dados de teste e casos de teste;
- Escopo do esforço de automação de teste;
- Educação da equipe de teste para a mudança de paradigma;
- Papéis e responsabilidades;
- Cooperação entre Desenvolvedores e Engenheiro de Automação de Testes;
- Esforço paralelo;
- Relatório de automação de testes.

Disponibilidade de ferramentas no ambiente de teste para automação de teste

As ferramentas de teste selecionadas precisam ser instaladas e confirmadas para estarem funcionando no ambiente do laboratório de teste. Isso pode envolver o download de quaisquer atualizações de versão, selecionando a configuração de instalação apropriada, incluindo suplementos, necessária para suportar o SUT e garantindo que as funções do TAS funcionem corretamente no ambiente de laboratório de teste versus o ambiente de desenvolvimento de automação.

Correção de dados de teste e casos de teste

A correção e a integridade dos dados de testes manuais e dos casos de teste são necessárias para garantir que o uso da automação proporcionará resultados previsíveis. Os testes executados sob automação precisam de dados explícitos para entrada, navegação, sincronização e validação.

Escopo do esforço de automação de teste

A fim de mostrar o sucesso inicial na automação e obter *feedback* sobre questões técnicas que podem afetar o progresso, começar com escopo limitado facilitará futuras tarefas de automação. Um projeto piloto pode segmentar uma área da funcionalidade de um sistema que seja representativo da interoperabilidade geral do sistema. As lições aprendidas com o piloto ajudarão a ajustar estimativas de tempo futuras, horários e identificar as áreas que requerem recursos técnicos especializados. Um projeto piloto fornece uma maneira rápida

de mostrar o sucesso precoce da automação, o que reforça o suporte de gerenciamento adicional.

Para ajudar nisto, os casos de teste a serem automatizados devem ser escolhidos com sabedoria. Escolha os casos que exigem pouco esforço para automatizar, mas fornecem um alto valor agregado. Regressão automática ou *smoke-teste* podem ser implementados e adicionar valor considerável como estes testes normalmente são executados com muita frequência, mesmo diariamente. Outro bom candidato para começar é o teste de confiabilidade. Esses testes são muitas vezes compostos de etapas e são executados uma e outra vez, revelando problemas que são difíceis de detectar manualmente. Esses testes de confiabilidade levam pouco esforço para implementar, mas podem mostrar valor agregado muito em breve.

Estes projetos-piloto colocam a automatização no centro das atenções (esforço manual de teste guardado ou problemas graves identificados) e abrem caminho a novas extensões (esforço e dinheiro).

Além disso, priorização deve ser dada aos testes que são críticos para a organização como estes irão mostrar o maior valor inicialmente. No entanto, neste contexto, é importante que, como parte de um esforço piloto, os testes mais tecnicamente desafiadores para automatizar são evitados. Caso contrário, muito esforço será gasto tentando desenvolver uma automação com pouquíssimo resultado para mostrar. Como regra geral, identificar testes que compartilham caraterísticas com uma grande parte da aplicação proporcionará o impulso necessário para manter o esforço de automação vivo.

Educação da equipe de teste para mudança de paradigma

Os testadores possuem muitas características: alguns são peritos do domínio que vêm da comunidade do usuário final ou o envolvimento como um analista de negócios, enquanto outros têm fortes habilidades técnicas que lhes permitam compreender melhor a arquitetura do sistema subjacente. Para que os testes sejam eficazes, é preferível uma ampla mistura. À medida que a equipe de teste muda para a automação, os papéis se tornarão mais especializados. Alterar a composição da equipe de teste é essencial para que a automação seja bem-sucedida e educar a equipe cedo sobre a mudança pretendida ajudará a reduzir a ansiedade sobre os papéis ou o possível pensamento de ser redundante. Quando tratada corretamente, a mudança para a automação deve fazer com que todos na equipe de teste estejam muito animados e prontos para participar da mudança organizacional e técnica.

Papéis e responsabilidades

A automação de teste deve ser uma atividade na qual todos podem participar. No entanto, isso não significa que todos tenham o mesmo papel. Projetar, implementar e manter um ambiente de teste automatizado é de natureza técnica e, como tal, deve ser reservado para indivíduos com fortes habilidades de programação e conhecimentos técnicos. Os resultados de um esforço automatizado de desenvolvimento de testes devem ser um ambiente que seja utilizável por indivíduos técnicos e não técnicos. A fim de maximizar o valor de um ambiente de teste automatizado, há uma necessidade de indivíduos com perícia de domínio e

habilidades de teste, pois será necessário desenvolver os scripts de teste apropriados (incluindo dados de teste correspondentes). Estes serão utilizados para conduzir o ambiente automatizado e fornecer a cobertura de teste alvo. Os especialistas de domínio revisam os relatórios para confirmar a funcionalidade do aplicativo, enquanto os especialistas técnicos garantem que o ambiente automatizado está operando correta e eficientemente. Esses especialistas técnicos também podem ser desenvolvedores com interesse em testes. Experiência em desenvolvimento de software é essencial para projetar software que é sustentável, e isso é de extrema importância na automação de teste. Os desenvolvedores podem se concentrar no framework de automação de teste ou em bibliotecas de teste. A implementação de casos de teste deve permanecer com os testadores.

Cooperação entre Desenvolvedores e o Engenheiro de Automação de Testes

Uma automação de teste bem-sucedida também requer o envolvimento da equipe de desenvolvimento de software, bem como de testadores. Os desenvolvedores e testadores precisarão trabalhar muito mais de perto para a automação de testes, para que os desenvolvedores possam fornecer pessoal de suporte e informações técnicas sobre seus métodos e ferramentas de desenvolvimento. Os Engenheiro de Automação de Teste podem levantar preocupações sobre a testabilidade dos projetos do sistema e do código do desenvolvedor. Isto será especialmente o caso se os padrões não forem seguidos, ou se os desenvolvedores usam bibliotecas e objetos estranhos, caseiros ou até mesmo muito novos. Por exemplo, os desenvolvedores podem escolher um controle GUI de terceiros que pode não ser compatível com a ferramenta de automação selecionada. Finalmente, a equipe de gerenciamento de projetos de uma organização deve ter um entendimento claro sobre os tipos de funções e responsabilidades necessárias para um esforço de automação bem-sucedido.

Esforço paralelo

Como parte das atividades de transição, muitas organizações criam uma equipe paralela para iniciar o processo de automatização de scripts de teste manual existentes. Os novos scripts automatizados são então incorporados ao esforço de teste, substituindo os scripts manuais. No entanto, antes de fazê-lo, geralmente é recomendado para comparar e validar que o script automatizado está executando o mesmo teste e validação como o script manual está substituindo.

Em muitos casos, uma avaliação dos scripts manuais será feita antes da conversão para automação. Como resultado dessa avaliação, pode ser determinado que há uma necessidade de reestruturar scripts de teste manual existentes para uma abordagem mais eficiente e eficaz sob automação.

Relatórios de automação de teste

Existem vários relatórios que podem ser gerados automaticamente por um TAS. Estes incluem critérios de aprovação/reprovação de scripts individuais ou etapas dentro de um script, estatísticas de execução de teste geral e desempenho geral do TAS. É igualmente importante

ter visibilidade sobre o funcionamento correto do TAS, de modo que qualquer resultado específico de aplicação que seja relatado possa ser considerado exato e completo (ver Capítulo 7, verificando o TAS).

6.2 Identificar as etapas necessárias para implementar a automação em testes de regressão

O teste de regressão fornece uma grande oportunidade para usar a automação. Uma suíte de teste de regressão cresce à medida que os testes funcionais de hoje, se tornam testes de regressão de amanhã. É apenas uma questão de tempo antes que o número de testes de regressão se torne maior do que o tempo e recursos disponíveis para uma equipe de teste manual tradicional. No desenvolvimento de etapas para se preparar para automatizar testes de regressão. Uma série de perguntas devem ser feitas:

- Com que frequência os testes devem ser executados?
- Qual é o tempo de execução para cada teste para o conjunto de regressão?
- Há sobreposição funcional entre os testes?
- Os testes compartilham dados?
- Os testes são dependentes um do outro?
- Que pré-condições são necessárias antes da execução do teste?
- Qual a percentagem de cobertura do SUT que os testes representam?
- Os testes são executados atualmente sem falha?
- O que deve acontecer quando os testes de regressão demorarem muito?

Cada um deles é explicado em mais detalhes abaixo.

Frequência de execução do teste

Os testes que são executados frequentemente como parte dos testes de regressão são os melhores candidatos para automação. Estes testes já foram desenvolvidos, testando-se a funcionalidade conhecida do SUT, e seu tempo de execução será reduzido tremendamente através do uso da automação.

Tempo de execução do teste

O tempo que leva para executar qualquer teste ou um conjunto de testes é um parâmetro importante na avaliação do valor da implementação de testes automatizados em testes de regressão. Uma opção é começar implementando a automação em testes mais demorados. Isso permitirá que cada teste seja executado de forma mais rápida e eficiente, além de adicionar ciclos adicionais de execução de teste de regressão automatizada. O benefício é um retorno adicional e mais frequente sobre a qualidade do SUT e redução no risco de implantação.

Sobreposição funcional

Ao automatizar os testes de regressão existentes, é uma boa prática para identificar qualquer sobreposição funcional que existe entre os casos de teste e, sempre que possível, reduzir essa

equivalência de sobreposição no teste automatizado. Isso trará mais eficiências no tempo de execução do teste automatizado, o que será significativo à medida que mais e mais casos de teste automatizados forem executados. Muitas vezes, testes desenvolvidos usando automação assumirão uma nova estrutura, uma vez que dependem de componentes reutilizáveis e repositórios de dados compartilhados. Não é incomum para decompor os testes manuais existentes em vários testes automatizados menores. Da mesma forma, a consolidação de vários testes manuais em um teste automatizado maior pode ser a solução apropriada. Os testes manuais precisam ser avaliados individualmente, e como um grupo, para que uma estratégia de conversão efetiva possa ser desenvolvida.

Compartilhamento de dados

Os testes geralmente compartilham dados. Isso pode ocorrer quando os testes usam o mesmo registro de dados para executar diferentes funcionalidades do SUT. Um exemplo disto pode ser o caso de teste "A" que verifica o tempo de férias disponível de um funcionário, enquanto o caso de teste "B" pode verificar quais cursos o funcionário tomou como parte de suas metas de desenvolvimento de carreira. Cada caso de teste usa o mesmo empregado, mas verifica parâmetros diferentes. Em um ambiente de teste manual, os dados do empregado normalmente seriam duplicados várias vezes em cada caso de teste manual que verificou os dados dos funcionários usando esse empregado. No entanto, em um teste automatizado, os dados compartilhados devem ser armazenados e acessados, sempre que possível, de uma única fonte para evitar duplicação ou introdução de erros.

Testar a interdependência

Ao executar cenários complexos de teste de regressão, um teste pode ter uma dependência de um ou mais testes. Essa ocorrência pode ser bastante comum e pode acontecer, por exemplo, como resultado de um novo "identificador de ordem" que é criado como resultado de uma etapa de teste. Testes subsequentes podem querer verificar que: a) a nova ordem é exibida corretamente no sistema, b) as alterações na ordem são possíveis, ou c) a exclusão da ordem é bem-sucedida. Em cada caso, o valor " identificador de ordem " que é criado dinamicamente no primeiro teste deve ser capturado para reutilização por testes posteriores. Dependendo do projeto do TAS, isso pode ser resolvido.

Pré-requisitos de teste

Muitas vezes, um teste não pode ser executado antes de definir as condições iniciais. Essas condições podem incluir selecionar o banco de dados correto ou o conjunto de dados de teste a partir do qual testar, ou definir valores iniciais ou parâmetros. Muitas dessas etapas de inicialização que são necessárias para estabelecer a pré-condição de um teste podem ser automatizadas. Isso permite uma solução mais confiável e confiável quando essas etapas não podem ser perdidas antes de executar os testes. Como os testes de regressão são convertidos em automação, essas pré-condições precisam fazer parte do processo de automação.

Cobertura SUT

Cada vez que os testes são executados, parte da funcionalidade de um SUT é exercida. A fim de determinar a qualidade global do SUT, os testes precisam ser projetados para ter a cobertura mais ampla e mais profunda. Além disso, as ferramentas de cobertura de código podem ser usadas para monitorar a execução de testes automatizados para ajudar a quantificar a eficácia dos testes. Através de testes de regressão automatizados, ao longo do tempo podemos esperar que testes adicionais proporcionem cobertura adicional. Medir isso fornece um meio eficaz de quantificar o valor dos próprios testes.

Testes executáveis

Antes de converter um teste de regressão manual em um teste automatizado, é importante verificar se o teste manual funciona corretamente. Isso fornece o ponto de partida correto para garantir uma conversão bem-sucedida para um teste de regressão automatizado. Se o teste manual não for executado corretamente, ou porque ele foi mal escrito, ou usa dados inválidos, ou está desatualizado ou fora de sincronia com o SUT atual, ou como resultado de um defeito SUT, convertê-lo em automação antes de entender ou resolver a causa raiz da falha irá criar um teste automatizado não-funcional que será um desperdício e improdutivo.

Conjuntos de teste de regressão grande

O conjunto de testes de regressão para um SUT pode se tornar bastante grande, tão grande que o conjunto de teste não pode ser executado completamente durante a noite, ou durante o fim de semana. Nesse caso, a execução simultânea de casos de teste é uma possibilidade se vários SUT estiverem disponíveis (para aplicações de PC isso provavelmente não representa um problema, mas quando o SUT consiste em um avião ou um foguete espacial esta é uma história diferente). Os SUT podem ser escassos e/ou caros, tornando a concorrência uma opção irrealista. Neste caso, uma possibilidade pode ser executar apenas partes do teste de regressão. Ao longo do tempo (semanas) o conjunto completo eventualmente será executado. A escolha de qual parte do conjunto de testes de regressão para executar também pode ser baseada em uma análise de risco (quais partes do SUT foram alteradas ultimamente?).

6.3 Fatores a serem considerados ao implementar a automação no teste de novos recursos

Em geral, é mais fácil automatizar casos de teste para novas funcionalidades, já que a implementação ainda não está concluída (ou melhor: ainda não iniciada). O Engenheiro de Automação de Teste pode usar seu conhecimento para explicar aos desenvolvedores e arquitetos o que exatamente é necessário na nova funcionalidade de tal forma que ela possa ser testada de forma eficaz e eficiente pela solução de automação de teste.

À medida que novos recursos são introduzidos em um SUT, os testadores são obrigados a desenvolver novos testes contra esses novos recursos e requisitos correspondentes. O TAE deve solicitar *feedback* de modeladores de teste experientes no domínio e determinar se o TAS atual

atenderá às necessidades dos novos recursos. Esta análise inclui, mas não se limita à abordagem existente utilizada, ferramentas de desenvolvimento de terceiros, ferramentas de teste utilizadas etc.

As alterações ao TAS devem ser avaliadas em relação aos componentes de *testware* automatizados existentes de modo que as alterações ou adições sejam totalmente documentadas e não afetem o comportamento (ou o desempenho) da funcionalidade TAS existente.

Se um novo recurso for implementado com, como exemplo, uma classe de objeto diferente, pode ser necessário fazer atualizações ou adições aos componentes do *testware*. Além disso, a compatibilidade com as ferramentas de teste existentes deve ser avaliada e, quando necessário, identificadas soluções alternativas. Por exemplo, se estiver usando uma abordagem baseada em palavras-chave, pode ser necessário desenvolver palavras-chave adicionais ou modificar/expandir palavras-chave existentes para acomodar a nova funcionalidade.

Pode haver um requisito para avaliar ferramentas de teste adicionais para suportar o novo ambiente sob o qual a nova funcionalidade existe. Por exemplo, uma nova ferramenta de teste pode ser necessária se a ferramenta de teste existente apenas suportar HTML.

Novos requisitos de teste podem afetar os testes e componentes *testware* existentes. Portanto, antes de fazer quaisquer alterações, os testes automatizados existentes devem ser executados contra o SUT novo ou atualizado para verificar e registrar quaisquer alterações na operação adequada dos testes automatizados existentes. Isso deve incluir interdependências de mapeamento para outros testes. Quaisquer novas mudanças na tecnologia exigirão a avaliação dos componentes *testware* atuais (incluindo ferramentas de teste, bibliotecas de funções, API etc.) e compatibilidade com o TAS existente.

Quando os requisitos existentes mudam, o esforço para atualizar os casos de teste que verificam esses requisitos deve fazer parte do cronograma do projeto (estrutura de divisão do trabalho). A rastreabilidade dos requisitos para os casos de teste indicará quais os casos de teste precisam ser atualizados. Essas atualizações devem fazer parte do plano geral.

Finalmente, é preciso determinar se o TAS existente continuará a atender às necessidades atuais do SUT. As técnicas de implementação ainda são válidas, ou é necessária uma nova arquitetura, e isso pode ser feito estendendo a capacidade atual?

Quando uma nova funcionalidade está sendo introduzida, esta é uma oportunidade para que o Engenheiro de Automação de Teste certifique-se de que a funcionalidade recém-definida será testável. Durante a fase de projeto, o teste deve ser levado em conta pelo planejamento para fornecer interfaces de teste que podem ser usados por linguagens de script ou a ferramenta de automação de teste para verificar a nova funcionalidade. Consulte o capítulo 2.3, modelagem para testabilidade e automação, para obter mais informações.

6.4 Fatores a considerar testando ao implementar automação de confirmação

O teste de confirmação é executado após uma correção de código que corrige um defeito relatado. Um testador normalmente segue as etapas necessárias para replicar o defeito para verificar se o defeito não existe mais.

Os defeitos têm uma maneira de reintroduzir-se em releases subsequentes (isso pode indicar um problema de gerenciamento de configuração) e, portanto, os testes de confirmação são os principais candidatos à automação. O uso da automação ajudará a reduzir o tempo de execução dos testes de confirmação. O teste de confirmação pode ser adicionado e complementar ao banco de teste de regressão automatizado existente.

O teste de confirmação automatizado normalmente tem um escopo estreito de funcionalidade. A implementação pode ocorrer em qualquer momento, uma vez que um defeito é relatado e os passos necessários para replicá-lo são entendidos. Testes de confirmação automatizados podem ser incorporados em um conjunto de regressão automatizada padrão ou, quando prático, subsumidos em testes automatizados existentes. Com ambas as abordagens, o valor da automatização do teste de confirmação de defeitos ainda é válido.

O rastreamento de testes de confirmação automatizados permite relatórios adicionais de tempo e número de ciclos gastos na resolução de defeitos.

Além do teste de confirmação, os testes de regressão são necessários para garantir que os novos defeitos não tenham sido introduzidos como um efeito colateral da correção de defeitos. A análise de impacto pode ser necessária para determinar o escopo apropriado dos testes de regressão.

7 Verificação do TAS [120 min]

Palavras-chave

verificação

Objetivos de aprendizagem

7.1 Verificando componentes automatizados do ambiente de teste

ALTA-E-7.1.1 (K3): Verificar a correção de um ambiente de teste automatizado, incluindo a configuração da ferramenta de teste.

7.2 Verificando o conjunto de testes automatizado

ALTA-E-7.2.1 (K3): Verifique o comportamento correto de um determinado script de teste automatizado e/ou conjunto de testes.

7.1 Verificando componentes automatizados do ambiente de teste

A equipe de automação de teste precisa verificar se o ambiente de teste automatizado está funcionando como esperado. Essas verificações são feitas, por exemplo, antes de iniciar os testes automatizados. Há uma série de etapas que podem ser tomadas para verificar os componentes do ambiente de teste automatizado. Cada uma delas é explicada em mais detalhes abaixo:

Instalação, configuração, adequação e personalização da ferramenta de teste

O TAS é composto de muitos componentes. Cada um deles precisa ser contabilizado para garantir desempenho confiável e repetitivo. No núcleo de um TAS estão os componentes executáveis, as bibliotecas funcionais correspondentes e os dados de suporte e os arquivos de configuração. O processo de configuração de um TAS pode variar do uso de scripts de instalação automatizados para colocar manualmente arquivos em pastas correspondentes. As ferramentas de teste, assim como os sistemas operacionais e outras aplicações, têm regularmente atualizações ou podem ter complementos opcionais ou necessários para garantir a compatibilidade com qualquer ambiente SUT.

A instalação automatizada (ou cópia) a partir de um repositório central tem vantagens. Pode ser garantido que os testes em diferentes SUT tenham sido realizados com a mesma versão do TAS, e a mesma configuração do TAS, onde isso é apropriado. As atualizações para o TAS podem ser feitas através do repositório. O uso do repositório e o processo para atualizar para uma nova versão do TAS devem ser os mesmos que para as ferramentas de desenvolvimento padrão.

Testar scripts com aprovações e reprovações conhecidas

Quando casos de teste que sabemos que serão aprovados falham, é imediatamente claro que algo está fundamentalmente errado e deve ser corrigido o mais rapidamente possível. Por outro lado, quando os casos de teste passam, mesmo que eles deveriam ter falhado, precisamos identificar o componente que não funcionou corretamente. É importante verificar a geração correta de arquivos de log e métricas de desempenho, bem como a configuração e desmontagem automatizada do caso ou script de teste. Também é útil executar alguns testes dos diferentes tipos e níveis de teste (testes funcionais, testes de desempenho, testes de componentes etc.). Isso também deve ser realizado no nível da estrutura.

Repetibilidade na configuração ou desmontagem do ambiente de teste

Um TAS será implementado em uma variedade de sistemas e servidores. Para garantir que o TAS funcione adequadamente em cada ambiente, é necessário ter uma abordagem sistemática para carregar e descarregar o TAS de qualquer ambiente. Isto é conseguido com sucesso quando a construção e reconstrução do TAS não fornece nenhuma diferença discernível na forma como opera dentro de e entre múltiplos ambientes. O gerenciamento de configuração dos componentes do TAS garante que uma determinada configuração pode ser criada de forma confiável.

Configuração do ambiente de teste e componentes

Entender e documentar os vários componentes que compõem o TAS fornece o conhecimento necessário para quais aspetos do TAS podem ser afetados ou exigir mudanças quando o ambiente do SUT muda.

Conectividade contra sistemas ou interfaces internas e externos

Uma vez que um TAS é instalado em um determinado ambiente SUT e antes do uso real contra um SUT, um conjunto de verificações ou pré-condições deve ser administrado para garantir que a conectividade com sistemas internos e externos, interfaces etc., está disponível. Estabelecer pré-condições para a automação é essencial para garantir que o TAS foi instalado e configurado corretamente.

Nível de intrusão de ferramentas de teste automatizadas

O TAS muitas vezes será fortemente acoplado com o SUT. Isso ocorre por modelagem para que haja um alto nível de compatibilidade, especialmente no que se refere a interações GUI nível. No entanto, esta estreita integração também pode ter efeitos negativos. Estes podem incluir: um SUT se comporta de forma diferente quando o TAS reside dentro do ambiente do SUT; o SUT tem comportamento diferente do que quando usado manualmente; o desempenho do SUT é afetado pelo TAS no ambiente ou ao executar o TAS contra o SUT. O nível de intrusão difere com a abordagem de teste automatizada escolhida. Por exemplo:

- Ao fazer a interface com o SUT a partir de interfaces externas, o nível de intrusão será muito baixo. As interfaces externas podem ser sinais eletrônicos (para *switches* físicos), sinais USB para dispositivos USB (como teclados). Com esta abordagem o usuário final é simulado da melhor maneira. Nesta abordagem, o software do SUT não é alterado para fins de teste. O comportamento e o tempo do SUT não são influenciados pela abordagem de teste. Interagir com o SUT desta maneira pode ser muito complexo. Hardware dedicado pode ser necessário, linguagens de descrição de hardware são necessários para a interface com o SUT etc. Para sistemas de software apenas isso não é uma abordagem típica, mas para produtos com software incorporado esta abordagem é mais comum.
- Ao fazer a interface com o SUT no nível GUI, o ambiente do SUT é adaptado para injetar comandos de interface do usuário e extrair as informações necessárias para os casos de teste. O comportamento do SUT não é alterado diretamente, mas o tempo é afetado que pode resultar em um impacto sobre o comportamento. O nível de intrusão é maior do que no ponto anterior, mas a interface com o SUT desta forma é menos complexa. Muitas vezes, as ferramentas comerciais de prateleira podem ser usadas para este tipo de automação.
- A interface com o SUT pode ser feita através de interfaces de teste no software ou usando interfaces já fornecidas pelo software. A disponibilidade dessas interfaces (API) é uma parte importante do projeto para a testabilidade. O nível de intrusão pode ser elevado neste caso. Testes automatizados usam interfaces que podem não ser usadas por usuários finais do sistema em todas (interfaces de teste) ou interfaces podem ser usadas em um contexto diferente do que no mundo real. Por outro lado, é muito fácil e barato realizar testes

automatizados através de interfaces (API). Testar o SUT através de interfaces de teste pode ser uma abordagem sólida, desde que o risco potencial seja compreendido.

Um alto nível de intrusão pode mostrar falhas durante o teste que não são evidentes nas condições de uso do mundo real. Se isso causar falhas com os testes automatizados, a confiança na solução de automação de teste pode cair dramaticamente. Os desenvolvedores podem exigir que as falhas identificadas por testes automatizados devem ser reproduzidas manualmente, se possível, a fim de auxiliar na análise.

Teste de componente de estrutura

Assim como qualquer projeto de desenvolvimento de software, os componentes de estrutura automatizada precisam ser testados e verificados individualmente. Isso pode incluir testes funcionais e não funcionais (desempenho, utilização de recursos, usabilidade etc.). Por exemplo, os componentes que fornecem verificação de objeto em sistemas GUI precisam ser testados para uma ampla gama de classes de objeto, a fim de estabelecer que a verificação de objeto funciona corretamente. Da mesma forma, logs de erro e relatórios devem produzir informações precisas sobre o status de automação e comportamento SUT.

Exemplos de testes não funcionais podem incluir a compreensão da degradação do desempenho da estrutura, a utilização de recursos do sistema que podem indicar problemas como vazamentos de memória. Interoperabilidade dos componentes dentro ou fora do quadro.

7.2 Verificando o conjunto de testes automatizado

As suítes de testes automatizadas precisam ser testadas quanto à integridade, consistência e comportamento correto. Diferentes tipos de verificações podem ser aplicados para garantir que o conjunto de testes automatizado esteja funcionando a qualquer momento ou para determinar se ele é adequado para uso. Há uma série de etapas que podem ser tomadas para verificar o conjunto de testes automatizados que incluem:

- Executar scripts de teste com aprovações e reprovações conhecidas
- Verificar a suíte de testes
- Verificar novos testes que se concentram em novos recursos do framework
- Considerar a repetibilidade dos testes
- Avaliar se há pontos de verificação suficientes no conjunto de testes automatizado ou casos de teste.

Cada um deles é explicado em mais detalhes abaixo.

Executar scripts de teste com aprovações e reprovações conhecidas

Quando conhecidos casos de teste aprovados falham, é imediatamente claro que algo está fundamentalmente errado e deve ser corrigido o mais rapidamente possível. Por outro lado, quando um conjunto de testes passa, mesmo quando deveria ter falhado, é necessário identificar este caso de teste que não funcionou corretamente. É importante verificar a geração correta de arquivos de log, dados de performance, configuração e desmontagem do caso de teste ou script de teste.

Também é útil executar alguns testes de diferentes tipos e níveis de teste (testes funcionais, testes de desempenho, testes de componentes etc.).

Verificar a suíte de testes

Verifique se o conjunto de testes está completo (os casos de teste têm resultados esperados, os dados de teste estão presentes) e a versão correta com o framework e o SUT.

Verificar novos testes que se concentram em novos recursos do framework

A primeira vez que um novo recurso do TAS é realmente usado em casos de teste, ele deve ser verificado e monitorado de perto para garantir que o recurso está funcionando corretamente.

Considerar a repetibilidade dos testes

Ao repetir testes, o resultado ou veredito do teste deve ser sempre o mesmo. Ter casos de teste no conjunto de teste que não dão um resultado confiável (p. ex., condições de corrida) poderia ser movido a partir do conjunto de testes automatizado ativo e analisado separadamente para encontrar a causa raiz. Caso contrário, o tempo será gasto repetidamente nestas execuções de teste para analisar o problema.

Falhas intermitentes precisam ser analisadas. O problema pode estar no próprio caso de teste ou na estrutura (ou pode até ser um problema no SUT). Análise de arquivo de log (do caso de teste, framework e SUT) pode identificar a causa raiz do problema. A depuração também pode ser necessária. Suporte do analista de teste, desenvolvedor de software e especialista de domínio pode ser necessário para encontrar a causa raiz.

Avaliar se há pontos de verificação suficientes no conjunto de testes automatizado ou casos de teste

Deve ser possível verificar se o conjunto de testes automatizado foi executado e atingiu os resultados esperados. Devem ser fornecidas evidências para garantir que o conjunto de testes ou casos de teste tenham funcionado como esperado. Esta evidência pode incluir registro no início e no final de cada caso de teste, registrando o status de execução de teste para cada caso de teste concluído, verificação de que as condições de pós foram alcançadas etc.

8 Melhoria contínua [150 min]

Palavras-chave

manutenção

Objetivos de aprendizagem

8.1 Opções para melhorar a automação de teste

ALTA-E-8.1.1 (K4): Analisar os aspetos técnicos de uma solução de automação de teste implementada fornecendo recomendações para melhoria.

8.2 Adaptação da automação de teste ao ambiente e mudanças SUT

ALTA-E-8.2.1 (K4): Analisar o *testware* automatizado, incluindo componentes de ambiente de teste, ferramentas e bibliotecas de função de suporte, para entender onde a consolidação e atualizações devem ser feitas seguindo um conjunto de ambiente de teste ou alterações SUT.

8.1 Opções para melhorar a automação de teste

Além das tarefas de manutenção em andamento necessárias para manter o TAS sincronizado com o SUT, geralmente há muitas oportunidades para melhorar o TAS. As melhorias para o TAS podem ser realizadas para alcançar uma série de benefícios, incluindo uma maior eficiência (reduzindo ainda mais a intervenção manual), facilitar sua utilização, implementar capacidades adicionais e um melhor suporte para as atividades de teste. A decisão sobre aplicar melhorias no TAS será influenciada pelos benefícios que irão agregar mais valor a um projeto.

As áreas específicas de um TAS que podem ser consideradas para melhoria incluem scripts, verificação, arquitetura, pré e pós processamento, documentação e suporte a ferramentas. Estas são descritas em mais detalhe abaixo.

Scripts

As abordagens de scripts variam da abordagem simples estruturada às abordagens baseadas em dados e às mais sofisticadas orientadas por palavras-chave, conforme descrito no capítulo 3.2.2. Pode ser apropriado atualizar a abordagem de script TAS atual para todos os novos testes automatizados. A abordagem também pode ser adaptada a todos os testes automatizados existentes ou, pelo menos, aqueles que envolvem a maior quantidade de esforço de manutenção.

Em vez de mudar completamente a abordagem de script, as melhorias de TAS podem se concentrar na implementação de scripts. Por exemplo:

- Avaliar o caso de teste, etapa ou procedimento sobrepostos em um esforço para consolidar testes automatizados. Casos de teste contendo sequências de ações semelhantes não devem implementar essas etapas várias vezes. Essas etapas devem ser feitas 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 *testware*. Quando os passos de teste não são idênticos, mas semelhantes, a parametrização pode ser necessária. Observação: esta é uma abordagem típica de testes orientados por palavras-chave.
- Estabelecer um processo de recuperação de erros para o TAS e o SUT. Quando um erro ocorre durante a execução de casos de teste, o TAS deve ser capaz de recuperar a partir desta condição de erro, a fim de ser capaz de continuar com o próximo caso de teste. Quando ocorre um erro no SUT, o TAS precisa ser capaz de executar ações de recuperação necessárias no SUT (p. ex., uma reinicialização do SUT completo).
- Avaliar mecanismos de espera para garantir que o melhor tipo está sendo usado. Há três mecanismos de espera comuns:
  1. Espera em um *hard-coded* (aguarde um certo número de milissegundos) pode ser uma causa raiz para muitos problemas de automação de teste.
  2. Espera dinâmica por sondagem, por exemplo, verificando se uma certa mudança de estado ou ação ocorreu, é muito mais flexível e eficiente:
     - Espera somente o tempo necessário e nenhum tempo de teste é desperdiçado

    - Quando, por algum motivo, o processo demorar mais tempo, a pesquisa apenas aguardará até que a condição seja verdadeira. Lembre-se de incluir um mecanismo de tempo limite, caso contrário, o teste pode esperar para sempre em caso de problema.
  3. Uma maneira ainda melhor é assinar o mecanismo de evento do SUT. Isso é muito mais confiável do que as outras duas opções, mas a linguagem de script de teste precisa suportar a assinatura de eventos e o SUT precisa oferecer esses eventos para o aplicativo de teste. Lembre-se de incluir um mecanismo de tempo limite, caso contrário, o teste pode esperar para sempre em caso de um problema
- Tratar o software de teste como um software. Desenvolvimento e manutenção de *testware* é apenas uma forma de desenvolvimento de software. Como tais boas práticas de codificação (p. ex., usando diretrizes de codificação, análise estática, revisões de código) devem ser aplicadas. Pode até ser uma boa ideia usar desenvolvedores de software (em vez de Engenheiros de Teste) para desenvolver certas partes do *testware* (p. ex., bibliotecas).
- Avaliar scripts existentes para revisão ou eliminação. Vários scripts podem ser problemáticos (p. ex., falhando ou com custos de manutenção elevados) e pode ser aconselhável redesenhar esses scripts. Outros scripts de teste podem ser removidos do conjunto porque eles não estão mais adicionando nenhum valor.

Execução de Teste

Quando uma suíte de teste de regressão automatizada não é terminada durante a noite, isso não deve vir como uma surpresa. Quando o teste demora muito tempo, pode ser necessário testar simultaneamente em sistemas diferentes, mas isso nem sempre é possível. Quando sistemas caros (alvos) são usados para testes, pode ser uma restrição que todos os testes devem ser feitos em um único alvo. Pode ser necessário dividir o conjunto de testes de regressão em várias partes, cada uma executando em um período definido (p. ex., em uma única noite). Uma análise mais aprofundada da cobertura de teste automatizada pode revelar duplicação. Remover a duplicação pode reduzir o tempo de execução e pode gerar mais eficiências. Uma análise posterior da cobertura de teste automatizada pode revelar duplicação. Remover a duplicação pode reduzir o tempo de execução e pode gerar mais eficiências.

Verificação

Antes de criar 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 são idênticos, mas semelhantes, o uso da parametrização ajudará a permitir que uma função seja usada em vários tipos de objetos.

Arquitetura

Pode ser necessário alterar a arquitetura para apoiar melhorias da testabilidade do SUT. Estas mudanças podem ser feitas na arquitetura do SUT ou da automação. Isso pode proporcionar uma grande melhoria na automação de teste, mas pode exigir mudanças significativas e investimento no SUT ou TAS. Por exemplo, se o SUT vai ser alterado para fornecer API para testes, em seguida, o TAS também deve ser refatorado em conformidade. Adicionar esses tipos de recursos em um estágio posterior pode ser bastante caro; é muito melhor pensar nisso no início da automação (e nos

estágios iniciais do desenvolvimento do SUT - ver capítulo 2.3, modelagem para testabilidade e automação).

Pré e pós processamento

Fornecer tarefas padrão de configuração e desmontagem. Estes também são conhecidos como pré-processamento (configuração) e pós-processamento (desmontagem). Isso economiza as tarefas sendo implementadas repetidamente para cada teste automatizado, não só reduzindo os custos de manutenção, mas também reduzindo o esforço necessário para implementar novos testes automatizados.

Documentação

Isso abrange todas as formas de documentação a partir da documentação de script (o que os scripts fazem, como devem ser usados etc.), a documentação do usuário para o TAS e os relatórios e logs produzidos pelo TAS.

Recursos TAS

Adicione recursos e funções adicionais do TAS, como relatórios detalhados, logs, integração a outros sistemas etc. Somente adicione novos recursos quando estes forem usados. A adição de recursos não utilizados aumenta a complexidade e diminui a confiabilidade e a manutenção.

Atualização e aprimoramento do TAS

A atualização ou aprimoramento de novas versões do TAS, disponibilizam novas funções que serão usadas pelos casos de teste (ou falhas que foram corrigidas). O risco é que a atualização do *framework* (atualizando as ferramentas de teste existentes ou introduzindo novas) possa ter um impacto negativo nos casos de teste existentes. Testar a nova versão da ferramenta de teste executando testes por amostragem antes de lançar a nova versão. Os testes por amostragem devem ser representativos dos testes automatizados em diferentes aplicações, diferentes tipos de teste e, se for caso disso, ambientes diferentes.

8.2 Planejando a implementação da melhoria da automação de teste

As mudanças em um TAS existente requerem planejamento e investigação cuidadosa. Muito esforço tem sido gasto na criação de um TAS robusto que consiste em um TAF e bibliotecas de componentes. Qualquer mudança, não importa o quão trivial, pode ter grande impacto sobre a confiabilidade e o desempenho do TAS.

Identificar alterações nos componentes do ambiente de teste

Avaliar quais mudanças e melhorias precisam ser feitas. Será que estas exigem alterações ao software de teste, bibliotecas de função personalizada, sistema operacional? Cada um destes tem um impacto sobre como o TAS executa. O objetivo geral é garantir que os testes automatizados

continuem a ser executados de forma eficiente. As alterações devem ser feitas de forma incremental para que o impacto no TAS possa ser medido através de uma série limitada de scripts de teste. Uma vez que se verifica que nenhum efeito prejudicial existe, as mudanças podem ser totalmente implementadas. Uma regressão completa é o passo final para validar que a mudança não afetou negativamente os scripts automatizados. Durante a execução desses scripts de regressão, os erros podem ser encontrados. Identificar a causa raiz destes erros (através de relatórios, logs, análise de dados etc.) proporcionará um meio para garantir que eles não são resultantes da atividade de melhoria de automação.

Aumente a eficiência e a eficácia das bibliotecas de funções principais do TAS

Como um TAS amadurece, novas maneiras são descobertas para executar tarefas de forma mais eficiente. Essas novas técnicas (que incluem a otimização de código em funções, usando bibliotecas de sistemas operacionais mais recentes etc.) precisam ser incorporadas às bibliotecas de função de núcleo que são usadas pelo projeto atual e todos os projetos.

Funções de múltiplos alvos que agem sobre o mesmo tipo de controle para consolidação

Uma grande parte do que ocorre durante uma execução de teste automatizada é a interrogação de controles na GUI. Esta interrogação serve para fornecer informações sobre esse controle (p. ex., visível ou não visível, ativado ou não ativado, tamanho e dimensões, dados etc.). Com essas informações, um teste automatizado pode selecionar um item de uma lista suspensa, inserir dados em um campo, ler um valor de um campo etc. Existem várias funções que podem atuar sobre 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 funciona somente em listas suspensas. Alternativamente, pode haver uma função (ou uma pode ser criada e usada dentro do TAS) que funciona com várias funções especificando uma função 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 o requisito de manutenção.

Refatorar a TAA para acomodar mudanças no SUT

Através da vida de um TAS, mudanças precisam ser feitas para acomodar mudanças no SUT. Como o SUT evolui e amadurece, a TAA subjacente terá que evoluir também para garantir que a capacidade está lá para apoiar o SUT. Deve-se ter cuidado ao estender os recursos para que eles não sejam implementados de forma aferrolhada, mas sim analisados e alterados ao nível arquitetural da solução automatizada. Isso garantirá que, uma vez que a nova funcionalidade SUT requer scripts adicionais, componentes compatíveis serão instalados para acomodar esses novos testes automatizados.

Convenções de nomenclatura e padronização

À medida que as alterações são introduzidas, as convenções de nomenclatura para novas bibliotecas de códigos e funções de automação precisam ser consistentes com padrões previamente definidos (consulte capítulo 4.3.2, escopo e abordagem).

Avaliação de scripts existentes para revisão ou eliminação de SUT

O processo de mudança e melhoria também inclui uma avaliação de scripts existentes, seu uso e valor continuado. Por exemplo, se certos testes são complexos e demorados para serem executados, decompô-los em vários testes menores pode ser mais viável e eficiente. Testes de segmentação que são executados com pouca ou nenhuma frequência para a eliminação reduzirão a complexidade do TAS e trarão maior clareza ao que precisa ser mantido.

Referências

Padrões

Os padrões para automação de teste incluem, mas não estão limitados a:

- O *Testing and Test Control Notation* (TTCN-3) do ETSI (*European Telecommunication Standards Institute*) e ITU (*International Telecommunication Union*) consistindo de:
  - 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
- O *Automatic Test Markup Language* (ATML) do IEEE (*Institute of Eletrical and Eletronics Engineers*) consistindo:
  - 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*
- A ISO/IEC/IEEE 29119-3:
  - ISO/IEC/IEEE 29119-3
- O *UML Testing Profile* (UTP) de OMG (*Objet Management Group*):
  - Arquitetura de teste, dados de teste, comportamento do teste, log de teste e gerenciamento de teste

Documentos ISTQB

[ISTQB-CTAL-TM] ISTQB Certified Tester, Advanced Level Syllabus, Test Manager, Version 2012, disponível em [ISTQB-Web, BSTQB-Web]

[ISTQB-CTAL-TTA] ISTQB Certified Tester, Advanced Level Syllabus, Technical Test Analyst, Version 2012, disponível em [ISTQB-Web, BSTQB-Web]

[ISTQB-CTEL-CEP] ISTQB Advanced Level Certification Extension, disponível em [ISTQB-Web, BSTQB-Web]

[ISTQB-CTEL-Modules] ISTQB Advanced Level Modules Overview, Version 1.2, August 23, 2013, disponível em [ISTQB-Web, BSTQB-Web]

[ISTQB-CTEL-TM] ISTQB Advanced Level – Test Management syllabus, Version 2011, disponível em [ISTQB-Web, BSTQB-Web]

[ISTQB-CTFL] ISTQB Foundation Level Syllabus, Version 2011, disponível em [ISTQB-Web, BSTQB-Web]

[ISTQB-Glossary] ISTQB Glossary of terms, Version 2.4, July 4, 2014, disponível em [ISTQB-Web, BSTQB-Web]

Marca registrada

As seguintes marcas registradas e marcas de serviço são usadas neste documento:

ISTQB® é uma marca registrada do International Software Testing Qualifications Board

BSTQB® é uma marca registrada do Brasillian Software Testing Qualifications Board

Livros

[Baker08] Paul Baker, Zhen Ru Dai, Jens Grabowski and Ina Schieferdecker, “Model-Driven Testing: Using the UML Testing Profile”, Springer 2008 edition, ISBN-10: 3540725628, ISBN-13: 978-3540725626

[Dustin09] 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

[Dustin99] Efriede Dustin, Jeff Rashka, John Paul, “Automated Software Testing: introdution, management, and performance”, Addison-Wesley, 1999, ISBN-10: 0201432870, ISBN-13: 9780201432879

[Fewster&Graham12] Mark Fewster, Dorothy Graham, “Experiences of Test Automation: Case Studies of Software Test Automation”, Addison-Wesley, 2012

[Fewster&Graham99] Mark Fewster, Dorothy Graham, “Software Test Automation: Effetive use of test execution tools”, ACM Press Books, 1999, ISBN-10:0201331403, ISBN-13: 9780201331400

[McCaffrey06] 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

[Mosley02] Daniel J. Mosley, Bruce A. Posey, "Just Enough Software Test Automation", Prentice Hall, 2002, ISBN-10: 0130084689, ISBN-13:9780130084682

[Willcock11] Colin Willcock, Thomas Deiß, Stephan Tobies and Stefan Keil, "Na Introdution to TTCN-3" Wiley, 2nd edition 2011, ISBN10: 0470663065, ISBN-13: 978-0470663066

Referências da Web

[ISTQB-Web] www.istqb.org - Website do International Software Testing Qualifications Board.

[BSTQB-Web] www.bstqb.org.br - Website do Brazilian Software Testing Qualification Board, membro brasileiro do ISTQB.

Aviso aos Provedores de Treinamento

Treinamentos

Cada capítulo do syllabus tem um tempo atribuído em minutos. A finalidade deste é tanto para dar orientação sobre a proporção relativa de tempo a ser atribuído a cada seção de um curso credenciado e para dar um tempo mínimo aproximado para o ensino de cada seção.

Os provedores de treinamento podem gastar mais tempo do que o indicado e os candidatos podem passar mais tempo novamente em leitura e pesquisa. Um currículo do curso não tem que seguir a mesma ordem que o syllabus. Não é necessário para conduzir o curso em um bloco contínuo de tempo.

A tabela abaixo fornece uma orientação para os tempos de ensino e exercício para cada capítulo (todos os horários são mostrados em minutos).

1. Introdução (0 min)
2. Introdução e objetivos para automação de teste (30 min)
3. Preparação para automação de teste (165 min)
4. A arquitetura de automação de teste genérica (270 min)
5. Implantação de riscos e contingências (150 min)
6. Relatórios e métricas de automação de teste (165 min)
7. Transição do teste manual para um ambiente automatizado (120 min)
8. Verificação do TAS (120 min)
9. Melhoria contínua (150 min)

Total: 1170 min

O tempo total do curso em dias, baseado em uma média de sete horas por dia de trabalho, é: 2 dias, 5 horas, 30 minutos.

Exercícios práticos no local de trabalho

Não existem exercícios definidos que possam ser realizados no local de trabalho.

Regras para e-Learning

Todas as partes deste currículo são consideradas apropriadas para implementação como *e-learning*.