Fundamentos para uma Automação de Testes Bem-sucedida

📑 Índice

📌 Fundamentos para uma Automação de Testes Bem-sucedida

🟩 Resumo por tópicos:

- Comece com **estratégia**, não com POCs soltas
- Crie uma **cultura favorável** à automação
- Envolva os **desenvolvedores** no processo
- Escolha **ferramentas adequadas** ao seu contexto
- Pense na **manutenção e escalabilidade** desde o início
- Meça e comunique o **valor da automação (ROI)**

🟦 Tabela Resumo:

Tópico	Foco Principal
Estratégia	Clareza nos objetivos e metas
Cultura	Alinhamento e colaboração entre times
Ferramentas	Escolha com base nas necessidades reais
ROI	Mostrar o impacto e o valor para o negócio

📘 Capítulo 1 – Definindo uma Estratégia de Automação de Testes

🟩 Resumo por tópicos:

🎯 O que você quer alcançar com a automação?

- Reduzir o tempo de execução dos testes de regressão manuais
- Automatizar testes de novas funcionalidades para reduzir dívida técnica
- Habilitar testes contínuos no pipeline de CI
- **Clareza no objetivo** é essencial — ele deve guiar todas as decisões posteriores
- Comunicar o objetivo a toda a equipe para alinhamento e engajamento

👥 Quem participará da automação e como?

- Escrever, monitorar e manter os testes exige tempo e dedicação
- **Desenvolvedores**: já sabem programar, mas estão ocupados com novas features
- **Testadores manuais**: entendem de testes, mas sobrecarregados com testes funcionais
- **Caminhos possíveis**:
  - Alocar temporariamente alguém para backlog de automação (não é sustentável)
  - Dedicar alguém em tempo integral para automação (ideal, mas exige aprendizado)
  - Contratar um **engenheiro de automação** (melhor opção, se viável)
- Tratar automação como um projeto de software real — não como tarefa paralela

🛠️ Como será executada a estratégia?

- **Quais testes automatizar?**
  - Nem todos os testes precisam ser automatizados — avalie risco e valor
- **Definir cenários claros**
  - Use testes de aceitação, Gherkin ou scripts definidos
- **Ferramentas e padrões**
  - Escolha tecnologias adequadas e defina convenções de escrita
- **Execução dos testes**
  - Execução local com reporte manual
  - CI dedicado com execuções regulares (feedback rápido)
  - Integração total ao CI da dev (confiança total exigida)
- **Sugestão de fases evolutivas:**
  a. Testes locais
  b. CI separado e visível
  c. CI com validação de commits

🔄 Triagem e manutenção:

- Testes falharão — planeje tempo e responsáveis para triagem contínua
- Desenvolvedores podem analisar falhas relacionadas ao que desenvolveram
- Testadores manuais podem contribuir revisando falhas e códigos de teste
- Um **responsável fixo** deve monitorar os testes — testes não confiáveis levam ao abandono do projeto

🟦 Tabela Resumo – Capítulo 1:

Aspecto	Pontos-chave
Objetivo da automação	Clareza sobre o que resolver: tempo, dívida técnica, CI, etc.
Participantes	Devs/testadores dedicados ou engenheiro de automação
Testes escolhidos	Basear em valor e risco — nem tudo precisa ser automatizado
Execução	Gradual: local → CI separado → CI dev
Triagem e manutenção	Indispensáveis para manter confiança na automação

Cultura de time	Alinhamento, clareza de papéis e comunicação constante

📘 Capítulo 2 – Criando uma Cultura para o Sucesso da Automação de Testes

🟩 Resumo por tópicos:

🔄 Cultura é tão importante quanto estratégia:

- Mesmo com uma boa estratégia, a automação falha sem cultura organizacional de apoio
- Sucesso vem da **colaboração entre todas as áreas**: PO, BA, Devs, Testers e Eng. de Automação

👤 Product Owner / Business Analyst:

- Devem entender que **nem todo teste precisa ser automatizado**
- Ajudam a definir **quais testes são mais valiosos** para o negócio e devem rodar com frequência
- POs bem envolvidos **priorizam manutenção da automação no backlog**
- Devem reconhecer que testes automatizados **viabilizam entregas mais rápidas e seguras**

👨‍💻 Desenvolvedores:

- São **grandes beneficiários da automação** (feedback rápido sobre seu código)
- Devem contribuir com:
  - **Testes unitários**
  - **Atualização de scripts falhos** após mudanças nas funcionalidades
- Precisam **entender seu papel na automatização**, mesmo que não liderem a iniciativa

🧪 Testadores (QAs):

- Testes manuais **continuam essenciais** — automação **não substitui QA**
- Podem ajudar identificando:
  - **Testes ideais para automatizar**
  - **Áreas críticas da aplicação**
- Também podem atuar na **triagem e manutenção dos testes automatizados**
- Devem ver a automação como **aliada**, não como ameaça ao trabalho

🟦 Tabela Resumo – Capítulo 2:

Papel na Equipe	Contribuições para o Sucesso da Automação
Product Owner / BA	Definem prioridades de testes, incluem manutenção no backlog
Desenvolvedores	Criam testes unitários, adaptam scripts, promovem automatizabilidade
Testadores (QAs)	Continuam com testes exploratórios, ajudam a identificar e manter scripts
Cultura Organizacional	Colaboração, clareza de papéis e valorização equilibrada entre QA e Dev

📘 Capítulo 3 – Desenvolvendo para Automatizabilidade de Testes

🟩 Resumo por tópicos:

🧱 Pirâmide de Automação de Testes (Mike Cohn):

- Três camadas: **Unit**, **Service/API**, **UI**
- Quanto mais **próximo do código**, mais rápido e confiável o teste
- **Prioridade** deve ser:
  a. **Testes unitários** (maior volume, rápidos, específicos)
  b. **Testes de serviços** (integrações sem UI)
  c. **Testes de UI** (menos estáveis, mais lentos)

⚖️ Evite exagerar nos testes de UI:

- Testar tudo pela interface é **caro e instável**
- Use testes de UI **somente onde necessário**
- Combine testes em **diferentes camadas** para cobrir o cenário com menor custo

🔍 Exemplo prático: Adicionar item ao carrinho:

- Em vez de:
  - Pesquisar → clicar produto → adicionar ao carrinho → abrir carrinho → validar
- Use um **atalho (seam)**: acessar diretamente o produto e adicionar via chamada de serviço

✂️ Seams (atalhos de automação):

- **URLs diretas**, **serviços expostos** e **funções de negócio reutilizáveis**
- Ajudam a **eliminar passos desnecessários** na interface
- Facilitam setup e teardown dos testes
- **Dev pode ajudar criando endpoints ou funções expostas para testes**

🔎 Elementos de UI: Boas práticas de localização:

- Testes de UI precisam de **elementos bem identificados**
- Elementos sem `id`, `name` ou `data-test` geram testes **flaky**
- Boas práticas:
  - Acordo com Devs para adicionar atributos para testes
  - **Revisão de código** e **linters** para reforçar regra
  - Planejar com o time quais elementos devem ser identificáveis

🟦 Tabela Resumo – Capítulo 3:

Tema	Prática Recomendável
Pirâmide de Testes	Priorizar testes unitários e de serviço; UI apenas quando necessário
Redução da dependência da UI	Usar seams e caminhos diretos para reduzir tempo e instabilidade
Seams (atalhos)	URLs diretas, chamadas de serviço, funções de negócio acessíveis
Localizadores de UI	Criar IDs ou atributos customizados para facilitar automação

Papel dos Desenvolvedores	Fornecer acessos técnicos (URLs, APIs) e colaborar na automatizabilidade da app

📘 Capítulo 4 – Ferramentas para Automação de Testes

🟩 Resumo por tópicos:

🔧 Antes de escolher ferramentas, pense em quem vai usá-las

- Se **testadores não programam**, construir um framework do zero pode ser inviável
  - Requer **muito investimento em capacitação**
- **Ferramentas codeless** (sem código) são opção, mas ainda exigem manutenção e triagem
- Se **devs ou engenheiros de automação** vão trabalhar nos testes, usar ferramentas com **código é ideal** (mais controle e flexibilidade)

🛠️ Ferramentas mínimas para o projeto:

- Dois componentes são essenciais:
  a. **Ferramenta de interação** (executa ações)
  b. **Ferramenta de validação** (verifica o resultado)

🔄 Interação:

- **Testes unitários e de serviços** → use a **mesma linguagem do código de produção**
- **APIs (serviços)** → escolha uma linguagem com suporte a **requisições HTTP** e boas bibliotecas
- **UI (interface)** → é necessário um **navegador headless ou biblioteca de navegação**
  - Priorize linguagens e ferramentas com **grande suporte na comunidade**
  - Considere **facilidade de contratação** de profissionais e **compatibilidade com dispositivos/browsers**

✅ Validação:

- Use **bibliotecas de asserção** para transformar scripts em testes que **passam ou falham**
- Pode incluir validações:
  - **Funcionais**
  - **Visuais**
  - **Acessibilidade**
  - **Segurança**

🚀 Ferramentas complementares (para projetos mais sofisticados):

- **Relatórios com print/vídeo**
- Integração com **Gherkin (BDD)** se a equipe usar esse formato
- Suporte à **integração contínua (CI)** é importante para execução automatizada

🟦 Tabela Resumo – Capítulo 4:

Necessidade	Ferramentas ou Requisitos
Interação (execução)	Linguagem compatível, bibliotecas HTTP, navegação na UI
Validação	Bibliotecas de asserção (pass/fail), incluindo testes visuais e funcionais

Público-alvo	Avaliar conhecimento técnico do time (testadores vs. devs)
Codeless x Código	Codeless para agilidade; código para flexibilidade e controle
Recursos extras	Relatórios, integração com CI, Gherkin para BDD
Considerações práticas	Facilidade de contratação, suporte comunitário, compatibilidade com browsers

📘 Capítulo 5 – Preparando sua Automação de Testes para o Futuro

🟩 Resumo por tópicos:

🚨 Evite armadilhas de curto prazo:

- Automação focada só no presente **leva a problemas** conforme o projeto cresce
- Design mal feito **não parece um problema com poucos testes**, mas vira um gargalo com centenas
- **Refatoração futura** custa caro e consome tempo — antecipe boas práticas

⚙️ Execução paralela:

- Testes devem ser **independentes** para rodar em paralelo
- Cada teste precisa ter seu próprio **setup e teardown**
- Evite que múltiplos testes **compartilhem ou modifiquem os mesmos dados**
- Variáveis e objetos compartilhados devem ser **thread-safe** (não globais/estáticos)

💡 Código limpo também vale para automação:

- Evite:
  - Duplicação de código
  - Métodos longos
  - “Espera” ineficiente nos testes
- Adote práticas de **reusabilidade, clareza e manutenção**

🧱 Padrões de projeto úteis para testes:

- Conheça padrões aplicáveis a automação:
  - **Page Object Model (POM)**
  - **Screenplay**
  - **Fluent**
  - **Builder**
  - **Singleton**
  - **Factory**
  - **Facade**
- **Nem todos precisam ser usados**, mas conhecer os padrões ajuda a aplicar o melhor para cada contexto

🟦 Tabela Resumo – Capítulo 5:

Tema	Boas práticas para sustentação da automação

Escalabilidade	Projetar testes para rodarem em paralelo, sem dependência entre si
Isolamento de dados	Testes devem usar dados únicos para evitar interferência
Código limpo	Evitar duplicações, métodos longos e estruturas confusas
Design Patterns	Aplicar POM, Screenplay, Builder, etc. conforme a arquitetura do projeto
Mentalidade preventiva	Pensar no crescimento futuro evita retrabalho e torna a automação confiável

📘 Capítulo 6 – Escalando sua Automação de Testes

🟩 Resumo por tópicos:

🌐 Ambientes múltiplos:

- Ao escalar, você precisará executar testes em diferentes **ambientes (dev, staging, prod...)**
- Cada ambiente pode ter:
  - URLs diferentes
  - Credenciais distintas
  - Configurações de banco de dados
- **Solução recomendada:**
  - Centralizar configurações em **arquivos de propriedades**
  - Evita múltiplos "ifs" no código e **facilita manutenção**
- Use ferramentas como **containers** para uniformizar ambientes de execução

🌍 Testes cross-browser:

- Automação reduz esforço repetitivo em diferentes navegadores
- Mas exige **gerenciamento das execuções em múltiplos browsers e versões**
- Avaliar:
  - Testar apenas navegadores críticos?
  - Usar **serviços em nuvem** com variedade de browsers (ex: BrowserStack, Sauce Labs)

📱 Testes em múltiplos dispositivos:

- Usuários acessam apps via **desktop, tablet, mobile**, e isso deve ser testado
- Questões comuns:
  - Interface responsiva muda elementos → **testes devem se adaptar**
  - Apps nativas em **iOS e Android** exigem abordagem própria
- Soluções:
  - Frameworks como **Appium** permitem escrever testes para ambos os sistemas
  - **Escolha de linguagem** impacta contribuição dos devs (iOS ≠ Android)
  - Apps com **fluxos e recursos diferentes** precisam de lógica específica nos testes
- ⚠️ Decisões sobre dispositivos e plataformas devem ser tomadas **desde o início** para evitar retrabalho

🟦 Tabela Resumo – Capítulo 6:

Desafio	Solução / Consideração
Ambientes múltiplos	Usar arquivos de propriedades e configurar via container
Navegadores diferentes	Foco nos principais; considerar serviços em nuvem para testes cross-browser
Dispositivos variados	Validar responsividade e apps nativas com frameworks multiplataforma
Testes adaptáveis	Elementos e fluxos podem mudar — código de teste precisa ser flexível
Estratégia de longo prazo	Definir suporte a múltiplos ambientes/dispositivos **ainda no início**

📘 Capítulo 7 – Medindo o Valor da sua Automação de Testes

🟩 Resumo por tópicos:

🎯 Evite expectativas irreais:

- Muitos projetos de automação falham por **esperarem resultados imediatos**
- Defina **expectativas realistas desde o início** e comunique à equipe
- Os **benefícios vêm com o tempo** e esforço contínuo

⏱️ Redução no tempo de regressão

- Meça quanto tempo a regressão **levava manualmente**
- Ao longo do tempo, a automação deve **encurtar drasticamente esse tempo**
- Defina um **SLA (tempo máximo de execução)** para incentivar testes mais rápidos

🔁 Feedback rápido e frequente:

- Com automação, o time recebe retorno **a cada alteração no código**
- Inicialmente pode ser manual, mas deve evoluir para **execução contínua no CI**
- **Quanto mais rápido o feedback, menor o custo do erro**

🚀 Aumento da velocidade no desenvolvimento:

- Testes automatizados confiáveis geram **confiança na equipe**
- Devs se sentem seguros para entregar mais rápido
- A qualidade dos testes também **reduz tempo com falhas "falsas"**

📈 Capacidade de escalar:

- Testes automatizados devem **funcionar em múltiplos ambientes, browsers e dispositivos**
- Escalar é parte do ROI e deve ser **monitorado ao longo do tempo**

✅ Resumo geral do curso:

- Automação é um **projeto de software**, não tarefa lateral
- Sucesso depende de:
  - Meta clara e estratégia
  - Cultura organizacional

- Aplicação automatizável
- Ferramentas corretas
- Visão de longo prazo
- Capacidade de escalar
- **Mensuração de resultados reais**

🟦 Tabela Resumo – Capítulo 7:

Métrica de Valor (ROI)	Como Medir
Tempo de regressão	Comparar execução manual vs automatizada; definir SLA de execução
Frequência de feedback	Ver se os testes são disparados automaticamente com push/check-in
Confiança nos testes	Avaliar em retrospectivas se a equipe confia nas falhas reportadas
Aumento da produtividade	Medir se devs entregam mais rápido com cobertura automatizada
Escalabilidade	Verificar cobertura em múltiplos ambientes, navegadores e dispositivos