

AGILE TESTING CONDENSED

EM PORTUGUÊS DO BRASIL



A Brief Introduction by

Janet Gregory & Lisa Crispin

Traduzido por
Felipe Knorr Kuhn & Paulo Gonçalves

Agile Testing Condensed - Brazilian Portuguese Edition

Uma Breve Introdução

Janet Gregory, Lisa Crispin, Felipe Knorr Kuhn
and Paulo Gonçalves

Esse livro está à venda em

<http://leanpub.com/agiletesting-condensed-brazilian-portuguese-edition>

Essa versão foi publicada em 2020-12-22



Esse é um livro [Leanpub](#). A Leanpub dá poderes aos autores e editores a partir do processo de Publicação Lean. [Publicação Lean](#) é a ação de publicar um ebook em desenvolvimento com ferramentas leves e muitas interações para conseguir feedbacks dos leitores, pivotar até que você tenha o livro ideal e então conseguir tração.

© 2020 Janet Gregory, Lisa Crispin, Felipe Knorr Kuhn and Paulo Gonçalves

Tweet Sobre Esse Livro!

Por favor ajude Janet Gregory, Lisa Crispin, Felipe Knorr Kuhn and Paulo Gonçalves a divulgar esse livro no [Twitter](#)!

O tweet sugerido para esse livro é:

Estou lendo o livro "Agile Testing Condensed: Em Português", de @janetgregoryca e @lisacrispin e traduzido por @knorrium e @paulorgoncalves.

A hashtag sugerida para esse livro é [#agiletestingbr](#).

Descubra o que as outras pessoas estão falando sobre esse livro clicando nesse link para buscar a hashtag no Twitter:

[#agiletestingbr](#)

Conteúdo

SEÇÃO 1: Fundamentos	1
Capítulo 1: O Que Entendemos por Teste Ágil?	2
Modelos de teste contínuo	2
Dez princípios para testes ágeis	4
Manifesto do teste	5
Definição de teste ágil	6
Capítulo 2: Abordagem do Time Inteiro e Mentalidade do Teste Ágil	7
Foco na qualidade	8
Como os times lidam com os defeitos	9
Perspectivas múltiplas	10
Capítulo 3: Planejamento de Teste em Contextos Ágeis	11
O time	11
O produto	12
Planejamento em níveis de detalhe	13
Planejando o teste de regressão	16
SEÇÃO 2: Abordagens de Teste	17
Capítulo 4: Guiando o Desenvolvimento com Exemplos	18
Métodos baseados em exemplos	18
Por que os exemplos ajudam	20
Esta é a sua base	22
Capítulo 5: Habilitando a Colaboração	23

CONTEÚDO

Colabore com os clientes	23
Mapeamento de impacto	24
Faça perguntas	25
Mapeamento de exemplo	26
Crie confiança usando visibilidade	27
Capítulo 6: Explore Continuamente	29
Personas, tarefas e papéis	30
Fluxos de trabalho e passeios	31
Riscos e valor para o cliente	32
Explore em pares ou grupos	32
Estatutos (charters)	33
Executando, aprendendo, guiando	35
Técnicas adicionais	35
Aproveite as ferramentas para uma exploração eficaz	35
Capítulo 7: Testando Atributos de Qualidade	37
Definindo atributos de qualidade	37
Mitigar riscos colaborando desde o início	38
Planejando o teste de pré-lançamento	40
Planejando para aprendizado posterior	40
Conformidade regulatória	41
Capítulo 8: Teste em DevOps	42
Entrega e implantação contínuas	43
Testando em produção	45
Monitoramento e observabilidade	45
A nova tecnologia nos traz novas capacidades	46
SEÇÃO 3: Modelos Úteis	48
Capítulo 9: Os Quadrantes de Testes Ágeis	49
Quais testes em qual ordem?	51
Usando os quadrantes	52
Definindo “pronto”	54
Encontre os modelos que se encaixam no seu contexto	55
Capítulo 10: Visualizando uma Estratégia de Automação de Teste	56

CONTEÚDO

Usando modelos visuais	56
A pirâmide clássica de automação de teste	57
Testes automatizados como documentação viva	59
Estendendo o modelo	60
Responsabilidade compartilhada	61
SECÃO 4: Teste Ágil Hoje	62
Capítulo 11: O Novo Papel de um Testador	63
Testadores são a cola de qualidade do time	63
Jornada profissional de um testador ágil	65
O caminho fascinante da evolução como testadores	66
Seja tudo o que você pode ser	68
Comece com uma conversa	69
O mundo não precisa de mais validadores	70
Pensamentos de Lisa e Janet	73
Capítulo 12: Ingredientes Para o Sucesso	75
Fatores de sucesso	75
Práticas de criação de confiança	80
Caminhos para o sucesso	83
Glossário	86
Recursos Para Aprender Mais	88
Geral	88
Desenvolvendo um Entendimento Compartilhado - Colaboração	88
Testes Exploratórios	89
DevOps, Monitoramento, Observabilidade	89
Automação de Testes	90
Sobre as Autoras	91
Sobre os Tradutores	92

Agradecimentos

Nossos livros sempre envolvem um esforço comunitário. Muito obrigado aos nossos colegas que contribuíram com suas visões de como a função de testador está evoluindo no Capítulo 11. Estamos escrevendo um livro pequeno aqui, portanto, não podemos agradecer individualmente a todas as pessoas com quem continuamos aprendendo novas ideias e conceitos para aplicar em nosso próprio trabalho e transferência para mais pessoas, mas saiba que somos gratos.

Mais gratidão às pessoas que revisaram nosso rascunho e nos deram um feedback tão útil, incluindo Mike Talks, Nikola Sporczyk, Lena Pejgan Wiberg, Pascal Stiefel, Barbara Zaleska, Bertold Kolics, Carol Vaage e nossa editora Erica Hunter. Muito obrigado a Constance Hermit por nos deixar usar seus maravilhosos desenhos no Capítulo 12 e Jenn Sinclair por seus desenhos que nós usamos tanto.

Um agradecimento especial a José Diaz e Johanna Rothman por seus belos prefácios.

E, por último, mas não menos importante, somos muito agradecidas aos nossos maridos, Jack Gregory e Bob Downing, por estarem sempre presentes com uma taça de vinho quando precisamos.



Prefácio por José Díaz

CEO, Trendig.com

A vida leva você ao seu objetivo. Quem teria pensado em 2009, quando comecei o primeiro Agile Testing Days (ATD), que eu escreveria o prefácio de um livro das Rainhas do Teste Ágil, embora Janet e Lisa tenham feito parte do ATD desde então. Estou muito feliz e honrado por elas me pedirem para fazer isso.

Agile Testing Condensed está de acordo com os dois livros anteriores de Janet e Lisa. Desta vez, elas iluminaram os cantos e exploraram os limites do teste e da qualidade em projetos ágeis - curto, conciso, certeiro e direto do coração de duas mulheres que dedicaram suas carreiras e vida para compartilhar seus conhecimentos e promover a atividade de testar em uma profissão ágil. Contém exemplos perspicazes, anedotas e é divertido de ler. Ambas as autoras compartilham suas experiências práticas e convidam os leitores a refazer sua jornada através de vários projetos. Elas levam você pela mão para um passeio por este mundo desafiador e belo dos testes ágeis e explicam isso a você. É um baú do tesouro, para você e seu time.

Eu recomendo não apenas este livro inspirador, mas também o treinamento delas, “Agile Testing for the Whole Team”, que é realista para a vida profissional.

O livro coroa o trabalho delas dos últimos 20 anos, nos quais elas têm lidado intensamente com o mundo ágil e sua comunidade. Seus inúmeros projetos, treinamentos, workshops, palestras, *webinars*, *lean coffee*, *coaching*, espaços abertos, sessões de mentoria e conversas estão condensados neste livro e tornam seu conteúdo muito prático e valioso. Este livro é uma leitura obrigatória para qualquer pessoa que esteja considerando uma jornada no mundo do ágil (incluindo gerentes).

É feito para você e para mim. Com muito amor! Vamos aproveitar!

Prefácio por Johanna Rothman

Autora de *Create Your Successful Agile Project*

Você pode ter lido os outros livros da Janet e Lisa sobre testes ágeis. Aqueles livros são muito mais do que apenas testes ágeis. Eu recomendo fortemente que você os leia.

Este livro, *Agile Testing Condensed*, se concentra em como criar um ambiente em que o time - e em especial os testadores - podem ter sucesso. O livro oferece muitas fontes em livros, artigos e posts de blogs que ajudam a reforçar a ideia e oferecem a você, leitor, maneiras de pensar sobre o problema.

Cada capítulo contém joias - essa é a parte condensada - que podem ajudar a orientar seu pensamento sobre teste e qualidade.

Por exemplo, muitos times pensam que o teste é como um testador testa uma funcionalidade específica. Em vez disso, há uma seção que ajuda todos a pensar sobre o produto e como planejar o esforço de teste através dos vários níveis do produto.

Outro exemplo é como elas nos lembram da colaboração. Desde o trabalho em pares para explorar, até o trabalho em trios para definir histórias, os testadores colaboraram.

Eu amei a seção “Os testadores são uma cola da qualidade para um time”. É verdade, e muitos poucos testadores e times tiram vantagem dessa cola.

Se você está se perguntando como ser um testador ágil, ou se não tem certeza se precisa de testadores em seu time ágil, leia este livro - uma leitura curta e rápida que pagará dividendos por um longo tempo.

Por que este livro?

Nosso objetivo com este livro foi criar um livro pequeno, conciso e fácil de ler, que qualquer um pudesse conseguir um conhecimento básico de como ter sucesso com testes e construir uma cultura de qualidade em um contexto ágil. Nossos primeiros dois livros (muito maiores) vão mais a fundo e contém histórias da vida real de profissionais da área. Nós referenciamos esses livros como:

- *Agile Testing*, que é o *Agile Testing: A Practical Guide for Testers and Agile Teams*, 2009
- *More Agile Testing*, que é o *More Agile Testing: Learning Journeys for the Whole Team*, 2014

Este livro não é um livro de introdução ao teste. Há muitos recursos ótimos disponíveis para aprender o básico sobre teste, automação de testes, DevOps e outros tópicos. Você pode encontrar bons links na nossa bibliografia. Da mesma forma, esta não é uma introdução ao desenvolvimento ágil. Este livro é para leitores que estão em times adotando métodos ágeis ou aqueles que querem saber como o teste e a qualidade se encaixam no desenvolvimento ágil e estão procurando orientação, como gerentes ou executivos.

Como ler esse livro

Você é bem-vindo para começar em qualquer seção ou ler capítulos individuais baseados no quê você quer aprender. Cada capítulo é auto-suficiente, embora nós possamos referenciar outros capítulos.

Use ele como um guia de bolso para o teste ágil para ter por perto enquanto você trabalha. Quando você estiver preso e precisar de inspiração, ou quando o seu time estiver pensando sobre um desafio de teste, procure nesse livro por ideias. Quando você quiser se aprofundar, confira os nossos outros livros.

Ao longo do livro, você irá encontrar dicas que irão lhe dar ideias sobre como abordar um problema específico.

SEÇÃO 1: Fundamentos

Nesta primeira seção nós compartilhamos a nossa definição de “teste ágil”. O coração do teste ágil envolve o time inteiro no teste e construção da qualidade no nosso produto. Há uma grande mudança na mentalidade enquanto o time aprende a prevenir defeitos ao invés de depender dos testadores como uma rede de segurança para apanhá-los.

Novos times ágeis aprendem a cortar grandes funcionalidades em pequenos pedaços incrementais e entregar mudanças pequenas frequentemente. Ao mesmo tempo, eles devem manter em mente a visão geral para manter a felicidade dos clientes.

- Capítulo 1: O Que Entendemos por Teste Ágil?
- Capítulo 2: Abordagem do Time Inteiro e Mentalidade do Teste Ágil
- Capítulo 3: Planejamento de Teste em Contextos Ágeis

Capítulo 1: O Que Entendemos por Teste Ágil?

Ao longo dos anos fomos questionadas muitas vezes sobre como definir “teste ágil.” Incluímos nossa “definição” de teste ágil na última parte deste capítulo, mas há muitos fatores que entram nessa definição. Algumas das práticas que ajudam os times em sua jornada em direção ao sucesso são:

- Construindo qualidade em: foco dos times na prevenção de mal-entendidos sobre o comportamento da funcionalidade, bem como na prevenção de defeitos no código
- Orientando o desenvolvimento com exemplos concretos: usando práticas como desenvolvimento orientado a testes de aceitação (ATDD), desenvolvimento guiado por comportamento (BDD) ou especificação por exemplo (SBE)
- Incluindo atividades de teste, como ter conversas para construir um entendimento compartilhado; fazer perguntas para testar ideias e premissas; automatizar testes; realizar testes exploratórios; teste de atributos de qualidade como desempenho, confiabilidade e segurança; e aprender com o uso em produção
- Usando retrospectivas do time inteiro e pequenos experimentos para continuamente melhorar os testes e a qualidade e descobrir o que funciona em seu contexto

Detalharemos as práticas acima ao longo deste livro. Nós não consideramos como “melhores práticas” porque sabemos que elas estão sempre evoluindo.

Modelos de teste contínuo

O teste é parte integrante do desenvolvimento de software junto com a codificação, operações, compreensão das necessidades do cliente e muito

mais. Gostamos dos modelos que representam uma abordagem de testes contínua ou holística. Um dos nossos favoritos é a abordagem que Ellen Gottesdiener e Mary Gorman usam em seu livro *Discover to Deliver*, 2012. Mostrado na Figura 1.1., o processo de desenvolvimento representa um loop infinito, confirmado continuamente - que é como realmente desenvolvemos software. Aprendemos sobre uma funcionalidade que nossos clientes desejam, nós construímos e entregamos, e então aprendemos como os clientes realmente a usam. Usamos esse feedback para decidir o que construir (ou remover) a seguir.

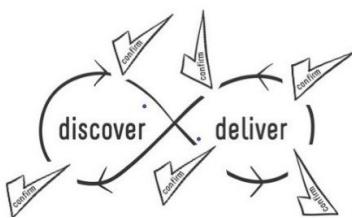


Figura 1.1: O loop da Descoberta e Entrega © 2015 por EBG Consulting

Dan Ashby adotou uma abordagem semelhante usando o diagrama abaixo (Figura 1.2) em uma postagem em seu blog enquanto falava sobre [testando em DevOps](#)¹. Este modelo mostra que o teste é parte integrante do DevOps. Nós entramos em mais detalhes sobre este assunto no Capítulo 8.

¹<https://danashby.co.uk/2016/10/19/continuous-testing-in-devops/>

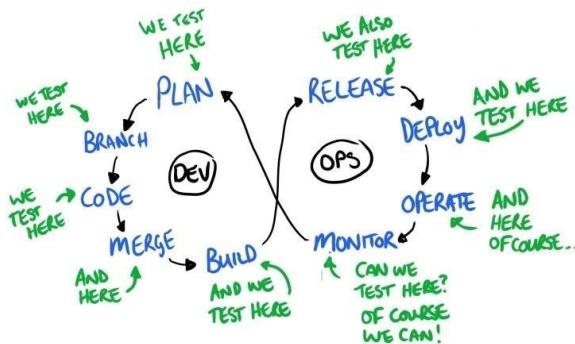


Figura 1.2: Teste contínuo no loop de DevOps

Gostamos de termos como teste contínuo ou teste holístico e reconhecemos que cada time precisa adaptá-los para seu próprio contexto único.

Dez princípios para testes ágeis

No livro *Agile Testing* apresentamos a ideia dos 10 Princípios para Testadores Ágeis. Percebemos agora que esses princípios não são necessariamente apenas para testadores, mas para qualquer pessoa do time. Escrevemos esses princípios em um momento em que a maioria dos testadores ainda fazia parte de um time de testes com um projeto em fases, e o teste ficava para o final - depois que toda a codificação estava “finalizada”.

Esses princípios ainda se aplicam hoje para qualquer pessoa em um time ágil que deseja entregar o produto com a mais alta qualidade possível.

- Fornecer feedback contínuo.
- Entregar valor ao cliente.
- Habilitar a comunicação face a face.
- Ter coragem.
- Manter a simplicidade.
- Praticar a melhoria contínua.
- Responder à mudança.

- Auto-organizar.
- Concentrar-se nas pessoas.
- Aproveitar!

Manifesto do teste

Vamos terminar este capítulo com este “manifesto do teste” criado por Karen Greaves e Samantha Laing. Seu manifesto reflete a mudança de mentalidade necessária para uma abordagem de teste ágil bem-sucedida. Nós testamos em todo o processo de desenvolvimento, focamos na prevenção de bugs, testamos muito mais do que a funcionalidade, e toda o time assume a responsabilidade pela qualidade. Você encontrará esses princípios refletidos em todos os nossos livros. Sempre que seu time estiver preso em um problema de qualidade ou teste, reflita sobre esses princípios para encontrar uma maneira para seguir adiante. (Figura 1.3).



Figura 1.3: O manifesto do teste

²<http://www.growingagile.co.nz/2015/04/the-testing-manifesto/>

Definição de teste ágil

A definição mais simples que encontramos para o que queremos dizer por teste ágil é a seguinte:

Práticas de teste colaborativas que ocorrem continuamente, desde o início até a entrega e além, apoiando a entrega frequente de valor para nossos clientes. As atividades de teste se concentram na construção de qualidade no produto, usando ciclos de feedbacks rápidos para validar nosso entendimento. As práticas fortalecem e apoiam a ideia de responsabilidade de todo o time pela qualidade.

Demora um pouco para digerir e você pode dar uma olhada em nosso [blog³](#) para mais detalhes.

³<https://agiletester.ca/ever-evolving-never-set-stone-definition-agile-testing/>

Capítulo 2: Abordagem do Time Inteiro e Mentalidade do Teste Ágil

Muitos times de software ainda utilizam uma abordagem linear em fases para entrega de software. Pessoas em uma determinada função são isoladas em times específicos, e elas passam o trabalho de um time para outro. O time de teste ou QA é visto como responsável por garantir a qualidade, geralmente bem no final do processo e logo antes da entrega em produção, quando é tarde demais para fazer o suficiente para melhorar a qualidade.

No desenvolvimento ágil quebramos os silos e transformamos o desenvolvimento em um processo contínuo e iterativo. Todo o time de entrega trabalha em conjunto para construir qualidade em todo o processo. Por “time inteiro”, geralmente queremos dizer time de entrega - as pessoas responsáveis por entender o que construir, desenvolverem e entregar o produto para o cliente.

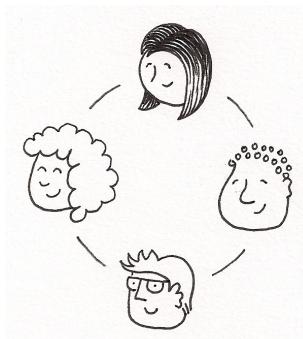


Figura 2.1: Um único time

Em organizações maiores, mesmo aquelas que adotaram princípios ágeis e práticas, pode haver mais de um time trabalhando em um produto, como

um time independente de banco de dados, um time de experiência do usuário ou outro time de produto. Nesses casos, a definição de time inteiro se estende para significar quem quer que você precise para entregar o produto. O movimento DevOps fez a inclusão das operações na entrega mais visível. Janet se refere às pessoas fora do time de entrega como uma família estendida.

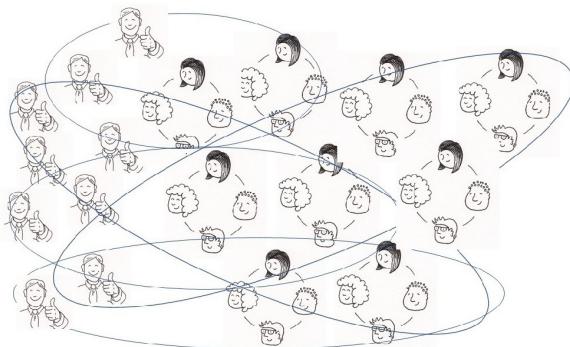


Figura 2.2: Vários times

Foco na qualidade

A abordagem de time inteiro significa que todos os membros do time são responsáveis pela qualidade de seu produto. Parte desta responsabilidade é garantir que as tarefas de teste sejam concluídas juntamente com o resto das tarefas de desenvolvimento. Quando o objetivo é entregar a mais alta qualidade possível, ao invés de entregar mais rápido, o time constrói uma base sólida de práticas. Para atingir esse nível de qualidade, os times gerenciam sua carga de trabalho para ter tempo para aprender as práticas essenciais, como o desenvolvimento guiado por testes (TDD - Test Driven Development) e testes exploratórios. Eles também dedicam tempo para aprender o domínio do negócio e construir relacionamentos com especialistas de negócios para identificar funcionalidades com o maior valor de negócio e, em seguida, implementá-las da maneira mais simples que for possível. Com o tempo, ao focar na qualidade, os times começam a ser capazes de trabalhar mais rápido.

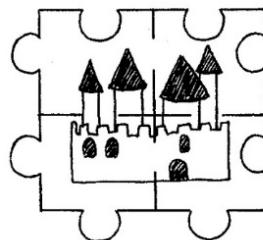


Figura 2.3: O seu valor de negócio entregue conforme o esperado

Existem várias áreas que exigem uma mudança na forma como os membros do time abordam o desenvolvimento. Quando todo o time é responsável pela qualidade do produto, bem como a qualidade do processo, cada membro do time precisa ser proativo em resolver problemas. Por exemplo, todos no time podem ajudar a descobrir o que tem mais valor para os clientes. Eles trabalham para entregar apenas o suficiente desse valor em pequenos incrementos para aprender como o cliente usa aquela funcionalidade. Ao criar esses ciclos rápidos de feedback, o time pode concentrar seus testes nas funcionalidades que possuem mais valor para o cliente.

Cada time precisa discutir e concordar sobre uma “Definição de Pronto” (DoD - Definition of Done) que tenha valor. Isso deve incluir como o time planeja lidar com os defeitos encontrados no código. A DoD deve incluir testes e a pergunta que precisa ser perguntada é: “Que tipos de teste você quer dizer?” No Capítulo 9 cobrimos os quadrantes do teste ágil e respondemos a essa pergunta. A DoD precisa ser entendida da mesma forma por todos os membros do time.

Como os times lidam com os defeitos

Uma grande mudança de mentalidade para os testadores é adotar uma abordagem de time para corrigir defeitos. No livro *More Agile Testing* falamos sobre como focar na prevenção de defeitos em vez de encontrá-los após a conclusão da codificação. Quando toda o time se concentra em construir qualidade no produto utilizando práticas como desenvolvimento

guiado por testes de aceitação e pensando sobre restrições em torno dos atributos de qualidade, isso pode ajudar a reduzir o número de defeitos encontrados no código.

Muitos times praticam tolerância zero a defeitos. Isso significa zero defeitos conhecidos escapam de uma iteração ou conclusão da história. Para fazer isso funcionar, os times devem ter feedback rápido das atividades de teste para que todos os defeitos encontrados possam ser corrigidos imediatamente. Uma vez encontrados, os programadores escrevem um ou mais testes executáveis, corrigem o código para que os testes passem e fazem teste exploratório, se necessário. O time pode então esquecer deles, sabendo que eles corrigiram o problema.

Muitas vezes, essa mudança filosófica na abordagem ajuda a mudar um ambiente adverso para um ambiente cooperativo.

Perspectivas múltiplas

Os membros do time têm diferentes pontos de vista, conjuntos de habilidades e perspectivas. Nós descobrimos que, usando todas as perspectivas, temos uma melhor compreensão dos riscos envolvidos na entrega de uma funcionalidade. Por exemplo, projetar para a testabilidade ajuda a transformar exemplos de comportamentos desejados e indesejados em testes executáveis. Os membros do time se tornam especialistas generalizados - isto é, eles podem ser especialistas em uma ou duas áreas, mas são capazes de contribuir para os objetivos comuns do time de várias maneiras.

Alguns exemplos:

- Os testadores podem ser especialistas em testar o produto, mas podem contribuir para compreender as funcionalidades e histórias, fazendo perguntas para descobrir suposições ocultas.
- Os programadores podem realizar testes exploratórios em seu próprio código antes de entregar o seu código.
- Os Product Owners executam testes de aceitação em cada história.

No Capítulo 11 falaremos um pouco mais sobre a função de um testador e como ela pode mudar para times ágeis.

Capítulo 3: Planejamento de Teste em Contextos Ágeis

Um dos nossos sete principais fatores de sucesso do *Teste Ágil* é “Não esqueça o panorama geral”. Os times muitas vezes se envolvem na construção, teste, e entrega de pequenos incrementos - o que encorajamos - e esquecem como essa pequena fatia se encaixa no sistema ou como funciona para resolver o problema do negócio.

Para planejar as atividades de teste de forma eficaz, um time precisa considerar seu contexto. Para entender seu contexto, pense em três aspectos dele: o time, o produto e os níveis de detalhe do seu sistema.

O time

Nem todos os times são criados iguais. Se você faz parte de um time pequeno co-localizado, você tem uma situação ideal para uma comunicação fácil. Você tem uma boa chance de aprender os valores uns dos outros e compartilhá-los. É um ponto ideal para entregar um ótimo produto e o planejamento é muito mais fácil. Os times podem entender facilmente a próxima funcionalidade e se aprofundar no planejamento a nível de história e tarefa.

No entanto, muitas pessoas trabalham em organizações grandes e globalmente distribuídas. Isso traz diferentes desafios. Organizações maiores têm vários projetos e muitos times. Quando eles adotam o ágil, muitas vezes substituem os silos com base na função, como desenvolvedores e QA, com Scrum ou silos de time de funcionalidades.

Quando muitos times grandes trabalham na mesma base de código a integração pode se tornar um grande desafio. Os times podem precisar de

especialistas, como testadores de desempenho, segurança e confiabilidade, mas pode não haver um número suficiente deles para atender todos os times multifuncionais. É ainda difícil tornar visíveis todos esses problemas e desafios. O planejamento no nível da versão é particularmente desafiador neste ambiente, mas é fundamental para fornecer novas funcionalidades aos clientes.

Não importa o contexto, o time de entrega deve assumir a responsabilidade pelo planejamento e conclusão de todas as atividades de teste, mesmo que isso signifique trazer especialistas. Se eles têm dependências, eles precisam trabalhar com outros times para gerenciar ou eliminar essas dependências, de preferência antes do início da codificação. Dito isso, os ajustes precisam ser feitos para se adequar a cada contexto único.

O produto

O nível de qualidade que as partes interessadas desejam depende do seu produto, bem como do tipo e quantidade de testes que podem ser necessários. Por exemplo, um sistema de gerenciamento de conteúdo usado apenas por usuários internos tem prioridades diferentes do software de dispositivo médico. Cada um tem um ambiente diferente no qual habitam e envolve riscos diferentes.

Considere o tamanho do seu produto, quantas pessoas o usam ou se ele está integrado com aplicações externas. Pense em como o produto é entregue e o risco associado ao mecanismo de entrega. Por exemplo, se a organização está hospedando sua própria aplicação da web, ela tem muito mais controle sobre quando e com que frequência o produto é atualizado. Ou se o produto precisa funcionar em muitos dispositivos, como telefones, como as atualizações acontecem sem interromper o uso regular?

Um dos principais objetivos dos testes é identificar e mitigar riscos - para o usuário e para a empresa. Obviamente, isso desempenha um grande papel em como você planeja seus testes. Esta é uma razão pela qual os times de entrega precisam aprender o domínio do negócio e trabalhar em estreita colaboração com especialistas em negócios. O domínio da expertise ajuda na hora de planejar o que testar. Seu time tem realmente uma boa ideia de como o produto é usado? Todos os membros do time tem conhecimento

de domínio? Colaborar com especialistas em produtos e negócios ajuda o time de entrega a encontrar maneiras ideais de construir funcionalidades que seus clientes valorizam.

Há muitas coisas a serem consideradas em torno do domínio do produto. Não é apenas o software que você está testando; é o produto que seus usuários finais dependem.

Planejamento em níveis de detalhe

O teste em vários níveis (Figura 3.1) requer planejamento extra. Os ciclos de liberação geralmente começam determinando o que pode ser entregue na primeira “release de aprendizado”. Talvez apenas parte de uma funcionalidade seja entregue. As funcionalidades são divididas em histórias e priorizadas para que o time saiba qual entregar primeiro. É importante que o time entenda o panorama geral antes de incluir as histórias em uma iteração. Quando os desenvolvedores trabalham em uma história, eles estão mais focados em garantir que as tarefas individuais sejam concluídas. Os testadores às vezes caem na armadilha de pensar apenas na história que estão testando, portanto, os lembretes do panorama geral são importantes.

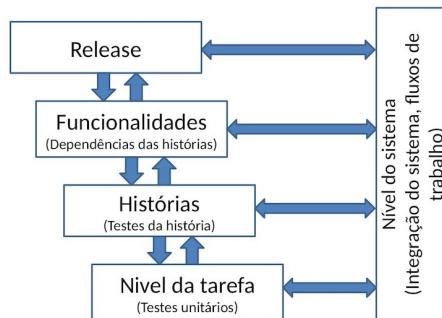


Figura 3.1: Níveis de detalhe para planejamento

Nível de release/funcionalidade

Os times precisam entender como cada história entregue pode afetar o panorama geral, especialmente em organizações maiores ou globais. Cada

release pode ser composta por muitas funcionalidades, que por sua vez podem ser compostas de muitas histórias e tarefas que têm impacto no sistema como um todo.

Em grandes organizações com vários times, todos trabalhando no mesmo produto, um dos problemas que frequentemente vemos é que os times tendem a se tornar “Silos.” Eles se esquecem de falar com outros times para resolverem possíveis dependências.

A Figura 3.2 mostra a importância de uma abordagem de teste que inclui todos os times trabalhando para o lançamento de um único produto. Para dar um panorama geral de cobertura de teste, considere reunir pessoas de diferentes times para criar um mapa mental de teste ou uma matriz de teste de funcionalidades (detalhes em *More Agile Testing*) que abrange o produto.

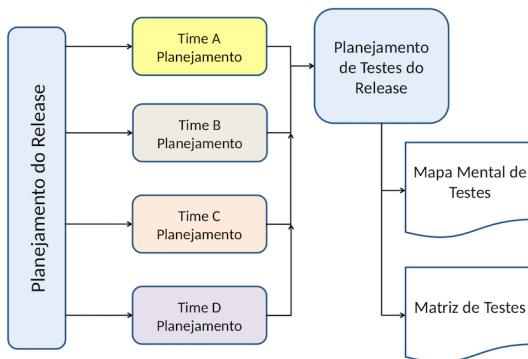


Figura 3.2: Planejamento para múltiplos times

Lembre-se, um tamanho não serve para todos, então faça concessões para o tamanho e número dos seus times, onde estão localizados, como será o trabalho coordenado entre os times, e se todas as habilidades necessárias para o teste estão disponíveis para cada time.

Idealmente, as atividades são coordenadas com outros times à medida que o desenvolvimento de funcionalidades progride. No entanto, é importante notar que um produto mais finalizado pode ser necessário para coisas como adicionar capturas de tela finais para o usuário ou documentação de treinamento. Ou, porque geralmente não é um processo rápido implantar uma correção para um aplicativo móvel, testes adicionais podem ser

necessários onde o time inteiro explora a versão mais recente mais uma vez.

Dica: Não cometa o erro de ter testadores e talvez time de operações encerrando o jogo final de pré-implantação enquanto os desenvolvedores começam novas histórias. Assim como o desenvolvimento, a preparação para a entrega deve ser um esforço do time inteiro.

Nível de história

Nesse nível, não importa se os times estão controlando suas iterações ou trabalhando em um método baseado em fluxo, como Kanban. Começar com testes de aceitação de alto nível (consulte o Capítulo 4: Guiando o Desenvolvimento com Exemplos para detalhes). Obtenha exemplos para aumentar o entendimento compartilhado da história e transformar esses exemplos em testes. Se os testes forem escritos antes que a codificação aconteça, eles ajudam a orientar o desenvolvimento e prevenir defeitos.

Dica: Considere quais estatutos de teste exploratório podem ser necessários (consulte o Capítulo 6). Pense sobre as restrições do produto e o que significa para testar atributos de qualidade (Capítulo 7).

Conforme os times planejam os testes e discutem a implementação de cada história, os detalhes sobre o teste emergem. Crie novos exemplos e testes para refletir o que foi aprendido sobre a história.

Nível de tarefa

Os programadores usam o Desenvolvimento Guiado por Testes (TDD) para escrever em baixo nível (unidade) testes antes de cada pequeno trecho de código. Alguns programadores chamam de Desenvolvimento Guiado por Design, uma vez que os ajuda a projetar seu código para ser testável. Esses testes são relativamente fáceis de escrever, executam rapidamente e dão feedback rápido para o time. Eles formam grande parte da base do modelo da pirâmide de automação de teste que discutimos no Capítulo 10: Visualizando uma Estratégia de Automação de Teste.

Planejando o teste de regressão

O teste de regressão trata de garantir que o sistema faça o que fez ontem. Práticas contemporâneas para entregar pequenas mudanças para produção frequentemente não deixam tempo para checagem de regressão manual completa, portanto, a automação de teste é criada à medida que o produto é desenvolvido. Automação de testes deve fazer parte de cada história, especialmente na camada de serviço (veja o Capítulo 10). Se uma história mudar de funcionalidade, certifique-se de incluir tarefas para alterar os testes existentes.

Os testes de regressão automatizados nos permitem ter confiança em nosso produto com feedback rápido. Muitos (senão todos) os testes são executados como parte de integração contínua (CI). Alguns times agendam testes mais lentos para serem executados menos frequentemente. Por exemplo, executá-los várias vezes ao dia em vez de cada build. Usar feature flags nas releases para “ocultar” as alterações dos usuários de produção permite que os times façam algumas atividades de teste de forma assíncrona à medida que são implantados continuamente en produção. A funcionalidade é “Ligada” em produção quando todos os testes forem concluídos (consulte o Capítulo 8: Teste em DevOps).

Você encontrará mais informações sobre isso no Capítulo 23: Teste e DevOps em *More Agile Testing*.

SEÇÃO 2: Abordagens de Teste

Nesta seção mergulhamos nas principais práticas para o teste ágil. Utilizar exemplos concretos para guiar o desenvolvimento é uma das maneiras mais eficazes para ter confiança em cada nova mudança. Compartilhamos maneiras de ajudar os membros do time em diferentes funções a aprender a colaborar na construção da qualidade no produto. O teste exploratório é outra prática principal para criação de confiança na qual todo o time deve se envolver.

Os times ágeis frequentemente caem na armadilha de apenas testar a funcionalidade - como cada funcionalidade deve se comportar. Existem muitos outros atributos de qualidade importantes que precisamos testar, que também cobrimos nesta seção.

DevOps e entrega contínua são temas em voga hoje em dia. Nós observamos como teste e testadores se encaixam e auxiliam seus times a obterem sucesso com tais abordagens.

- Capítulo 4: Guiando o Desenvolvimento com Exemplos
- Capítulo 5: Habilitando a Colaboração
- Capítulo 6: Explore Continuamente
- Capítulo 7: Testando Atributos de Qualidade
- Capítulo 8: Teste em DevOps

Capítulo 4: Guiando o Desenvolvimento com Exemplos

A ideia de utilizar exemplos para guiar o desenvolvimento de funcionalidades e histórias tem sido praticada por muitos times há anos. Nós vemos isso como uma abordagem valiosa, testada e aprovada. Os principais profissionais continuam a encontrar novas maneiras de ajudarem os times a terem sucesso com essas técnicas: por exemplo, mapeamento de exemplos de Matt Wynne (veja mais no Capítulo 5).

Exemplos concretos do comportamento desejado e indesejado do sistema ajudam os times a construir um entendimento compartilhado de cada funcionalidade e história. Isso lhes permite construir a coisa certa com menos rejeições de história e ciclos menores entre o início do desenvolvimento e a implantação em produção. Os testadores contribuem pedindo por esses exemplos concretos e os utilizando para criar testes executáveis que guiam o desenvolvimento. Eles podem ser a voz da experiência na liderança dessas conversas.

Métodos baseados em exemplos

Existem algumas variações para o desenvolvimento de funcionalidades e histórias com base em exemplos. O desenvolvimento guiado por comportamento (BDD - Behavior Driven Development) é o método que Janet ouve a maioria dos times afirmarem que usam. BDD, apresentado pela primeira vez por Daniel Terhorst-North, captura cenários de exemplo em uma linguagem natural específica de domínio. A sintaxe “Dado / Quando / Então” descreve pré-condições, alguma ação de gatilho e pós-condição de resultado. Conforme os desenvolvedores escrevem o código de produção,

eles provavelmente automatizam alguns ou todos os cenários para ajudar a saber quando eles entregaram o que o cliente deseja.

Escrever esses cenários pode parecer fácil, mas é preciso prática para simplificar os testes para que realmente haja apenas uma coisa sendo testada. Veja a Figura 4.1 para um exemplo.

O desenvolvimento guiado por testes de aceitação (ATDD - Acceptance Test-Driven Development) é semelhante. É uma forma mais genérica de guiar o desenvolvimento com exemplos sem linguagem ou regras estritas. A maioria das pessoas usam ATDD para testes funcionais, embora os requisitos para outros atributos de qualidade, como segurança ou acessibilidade, possam ser incluídos. Uma abordagem ATDD que utilizamos é capturar pelo menos um exemplo em alto nível do comportamento desejado ou “caminho feliz” e pelo menos um exemplo para cada tipo de comportamento indesejado conforme o time planeja a história. Testadores e outros membros do time definem exemplos mais detalhados à medida que ao desenvolvimento progride na história. Pelo menos alguns dos exemplos são transformados em testes executáveis que ajudam o time a decidir quando eles estão prontos.

A Figura 4.1 mostra uma progressão começando com o corte da funcionalidade em histórias. As bolhas azuis são feitas como parte de workshops de preparação de histórias, refinamento do backlog ou discussões dos “três amigos”.

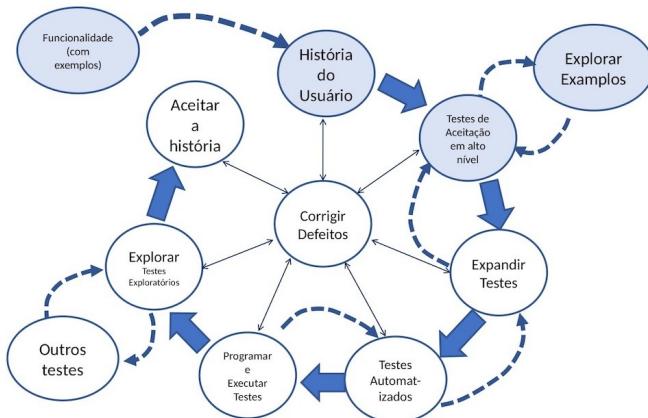


Figura 4.1: Desenvolvimento guiado por testes de aceitação (ATDD)

Os times que praticam a Especificação por Exemplo (SBE) começam identificando objetivos em torno da história utilizando uma abordagem como o mapeamento de impacto (veja mais no Capítulo 5). O time, então, definem exemplos chave, que se tornam as especificações, durante um workshop de especificação. À medida que o desenvolvimento prossegue, os exemplos são refinados e transformados em especificações executáveis para validar o produto com frequência. Estes exemplos executáveis, assim como no BDD e ATDD, se tornam a documentação viva do sistema. Quando Gojko Adzic cunhou o termo Especificação Por Exemplo, ele deliberadamente não utilizou a palavra “teste” para descrever qualquer uma das atividades.

Por que os exemplos ajudam

Olhar para cada nova funcionalidade de uma variedade de perspectivas ajuda ao time terá maior probabilidade de identificar o valor que os clientes obtêm de cada nova funcionalidade. Essa diversidade ajuda cada membro do time a superar os preconceitos inconscientes e “pensar fora da caixa”. Quando cada parte interessada é solicitada a informar exemplos concretos do comportamento do sistema, os times olham para esses exemplos, e é fácil ver as discrepâncias. Também é mais fácil chegar até o mínimo valor específico do que os clientes precisam e permite que os times entreguem “apenas o suficiente” da coisa certa.

Mostraremos o que queremos dizer com um cenário de exemplo.

História: Como um comprador canadense, quero dar dinheiro ao caixa e espero o troco correto para que eu pague apenas a quantia certa pelas minhas compras. A caixa registradora relata a quantidade correta de troco para dar.

Cenário: O caminho feliz onde o comprador dá mais do que o valor das compras e recebe o troco correto.

Dado eu sou um comprador canadense e fiz compras no valor de \$4,97,

Quando eu der ao caixa \$5,00,

Então espero receber \$0,05 de troco.

Observação: Utilizando este exemplo, uma regra de negócio que o time pode não ter considerado é que, no Canadá, não são utilizadas moedas de um centavo, então o número é arredondado para cima ou para baixo e o troco é dado o conforme for mais próximo de cinco centavos.

Uma das maneiras favoritas da Janet de mostrar exemplos é em formato tabular. Isto mostra rapidamente o que você está perdendo e o que as pessoas estão pensando enquanto estão discutindo. Cada linha pode se tornar um teste. A Figura 4.2 mostra esse formato para o cenário que utilizamos acima.

Total das compras	Dinheiro dado	Troco devolvido	Teste
4,97	5,00	0,05	Caminho feliz
4,97	4,00	0,00	Dinheiro insuficiente
4,97	10,00	5,05	Caminho feliz

Figura 4.2: Exemplo de um formato tabular de uso de exemplos concretos

Existem tantas maneiras boas de estruturar conversas onde os times podem definir exemplos. O [mapeamento de histórias](#)⁴ do Jeff Patton ajuda todo o time a acompanhar a jornada do usuário através do produto. Nós gostamos de capturar regras de negócio específicas, bem como exemplos

⁴<https://www.jpattonassociates.com/user-story-mapping/>

que ilustram elas através do mapeamento de exemplo. Conversas estruturadas utilizando as [Sete Dimensões do Produto](#)⁵ de Ellen Gottesdiener e Mary Gorman garantem que os times tenham exemplos de muitos aspectos diferentes de valor. Qualquer técnica que promove a colaboração face a face em pequenos grupos multifuncionais vale a pena tentar. Veja o Capítulo 7 em *More Agile Testing* para mais detalhes e histórias.

Esta é a sua base

Reunir membros do time de desenvolvimento e negócios para definir exemplos é a chave para entregar valor aos clientes em um ritmo frequente e sustentável. Isso permite que os times construam o entendimento compartilhado importante antes de começarem a codificar. Isso ajuda com que todos permaneçam firmes na realidade. Os exemplos concretos tornam-se testes executáveis que garantem que construímos a coisa certa e que essa coisa continuará funcionando corretamente no futuro até que os clientes decidam mudá-la.

Dica: Quando você se encontrar em uma discussão agitada sobre requisitos para uma funcionalidade ou um argumento sobre como uma determinada funcionalidade deve se comportar, PARE. Peça um exemplo. Ou ainda melhor, reúna o grupo para começar a desenhar exemplos no quadro branco (real ou virtual). Você economizará muito tempo e ficará mais perto de construir a coisa certa.

⁵<https://discovertodeliver.com/image/data/Resources/visuals/DtoD-7-Product-Dimensions.pdf>

Capítulo 5: Habilitando a Colaboração

A colaboração dentro de um time e entre os times é um dos pilares que tornam os times ágeis bem-sucedidos. No entanto, descobrimos que muitos times não tem ideia de como começar a criar esses relacionamentos. Nesse capítulo falaremos sobre algumas práticas muito simples que podem ajudar você e seu time a ganhar tração.

Colabore com os clientes

Vamos começar colaborando com o cliente - geralmente representado por um product owner. Se os times não entenderem qual é o problema que o cliente está tentando resolver, eles podem resolver o errado. É essencial que os times trabalhem com seus clientes para entenderem suas verdadeiras necessidades.

Primeiro, sugerimos fortemente que todos no time entendam o domínio. Isso pode ser feito trabalhando em estreita colaboração com os usuários finais, pedindo exemplos ou cenários, ou mesmo desenhando imagens em quadros brancos para compreender as diferenças e esclarecer os significados.

Perguntas como essas farão o cliente considerar o uso e o risco associado.

“Como você vai usar isso?”

“Qual é o pior que pode acontecer?”

Os testadores podem facilitar a comunicação desenvolvedor-cliente, mas é importante não atrapalhar. Chamamos a prática de reunir um testador, um programador e um especialista em negócios (product owner, gerente de produto ou analista de negócios) juntos para falar sobre uma história

de usuário, o “Poder dos três”. George Dinwiddie se refere a ele como os **Três Amigos**⁶. É uma maneira poderosa de construir um entendimento compartilhado sobre histórias, funcionalidades e como eles se encaixam no produto.

Dica: Reúna o testador, o programador e o especialista em negócios e talvez um ou dois outros papéis sempre que surgir uma dúvida. Por exemplo, um par de desenvolvedores está trabalhando em uma nova história e um dos testes voltados para os negócios falha. Eles vão falar para um testador que acham que o teste está esperando o comportamento errado. Essa é a hora de pegar um product owner e ter uma discussão tripla. Essa rápida conversa economiza muito tempo tentando corrigir um defeito que apareceu no código.

Dependendo do produto e do tipo de funcionalidade que estão sendo desenvolvidos, mais perspectivas podem ser necessárias, como de um designer de UX, um especialista em dados ou um especialista em operações.



Figura 5.1: Convide as pessoas certas

Mapeamento de impacto

Estruturas como **mapeamento de impacto**⁷ são úteis para decidir quais funcionalidades devemos construir e talvez até mesmo determinar qual deve ser a prioridade. Comece com o objetivo de uma funcionalidade (o “porquê”). Em seguida, identifique quem pode nos ajudar a atingir esse objetivo e quem pode ficar no nosso caminho. Para cada “quem”, pergunte

⁶<https://www.agileconnection.com/article/three-amigos-strategy-developing-user-stories>

⁷<https://www.impactmapping.org/>

como eles podem ajudar ou atrapalhar em alcançar a meta (esses são os impactos). Por último, pense sobre quais entregas podem resultar de cada impacto (o “o quê”). Este exercício ajuda o time a entender o quadro geral e as razões por trás do que estão desenvolvendo.

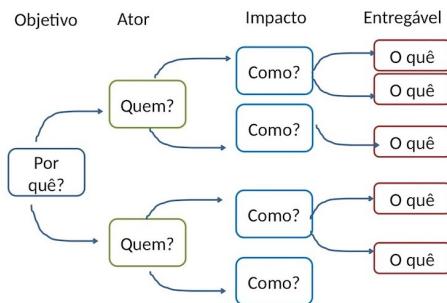


Figura 5.2: Mapeamento de impacto

Isto responde à pergunta “Como saberemos se esta funcionalidade atinge o objetivo após o lançamento?”

Faça perguntas

É comum que uma reunião de planejamento de funcionalidades comece com uma discussão sobre como implementar a funcionalidade. Às vezes, o product owner tem suas próprias ideias: “Pegue o mesmo código que usamos para códigos de desconto e torne-os valores negativos para que possamos adicionar sobretaxas.” (Sim, Lisa teve exatamente essa experiência.) É importante não permitir que isso aconteça - comece explicando o porquê.

Quando você se reúne com uma parte interessada do negócio, como um product owner para falar sobre as próximas funcionalidades, a primeira pergunta a fazer é: “Por que estamos fazendo essa funcionalidade?” Outras boas perguntas: “Que problema isso resolverá para a empresa, o cliente ou o usuário final?”

QA significa “Question Asker” (questionador) - uma ideia que tivemos de Pete Walen. Os testadores costumam fazer perguntas que ninguém mais

pensa em fazer, então se você não tem um testador no time, tente designar uma função de questionador.

Times experientes costumam criar critérios de qualidade na maneira como trabalham. Por exemplo, se quiserem evitar problemas de segurança, como Cross-Site Scripting (XSS) e injeção de SQL, provavelmente está integrado à arquitetura do sistema.

No entanto, em nossa experiência, as partes interessadas do negócio muitas vezes assumem incorretamente que o time técnico já sabe quais atributos de qualidade são importantes - atributos como quantos usuários simultâneos usarão o produto, quais dispositivos precisam ser suportados ou quanto rápido o tempo de resposta percebido de um aplicativo precisa ser rápido. Consulte o Capítulo 7 para mais informações sobre atributos de qualidade.

Fazer perguntas específicas e também abertas ajuda a expor suposições ocultas.

- “É possível que possamos implementar essa funcionalidade e não resolver o problema?”
- “O que os usuários farão antes de usar essa funcionalidade?”
- “O que eles farão depois?”

Mapeamento de exemplo

Matt Wynne nos apresentou a ideia de [mapeamento de exemplo](#)⁸, e descobrimos que é uma ótima maneira de explorar uma nova funcionalidade e o valor que ela deve entregar. Trabalhe com o product owner ou outras partes interessadas sobre as regras de negócios em uma discussão do tipo “Poder dos Três”. As regras de negócio são ótimos lugares para começar a explorar uma funcionalidade, pois podem nos ajudar a dividir uma funcionalidade em histórias, bem como obter um entendimento compartilhado de como a funcionalidade deve se comportar.

A técnica de mapeamento de exemplo de Matt Wynne é uma base altamente eficaz para esse tipo de conversa porque exemplos concretos são

⁸<https://cucumber.io/blog/example-mapping-introduction/>

usados para ajudar a esclarecer nossa compreensão das regras. Conforme a conversa continua, mantenha o objetivo principal em mente e concentre-se no valor que o recurso oferece aos clientes e usuários finais. Os times geralmente descobrem que mais regras de negócios são expostas como resultado do uso de exemplos reais.

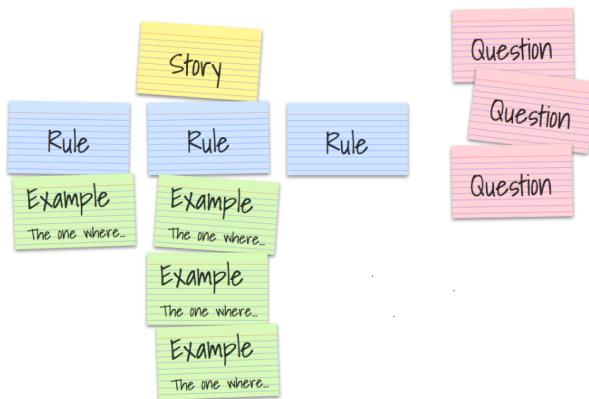


Figura 5.3: Exemplo de mapeamento

Usando o mapeamento de exemplo para obter regras de negócios, exemplos e perguntas que precisam ser respondidas é uma maneira eficaz de garantir que o time maior comece na mesma página ao planejar a iteração.

Crie confiança usando visibilidade

Os testes permitem que os times identifiquem os riscos para que os clientes possam tomar as melhores decisões, o que, por sua vez, gera confiança. Quando o time pede sua opinião, os clientes ganham a confiança de que obterão um software funcional.

Podemos usar mapas mentais, diagramas de fluxo, diagramas de contexto, diagramas de estado, ou outras ferramentas para observar as dependências e os efeitos propagadores de cada nova funcionalidade. Se nossas funcionalidades não forem facilmente entendidas e usadas por todos, elas não

podem fornecer o valor pretendido. Ficar de olho no panorama geral é um ponto forte que os testadores trazem para os times ágeis.

Desenhar em um quadro branco enquanto discute uma história é uma maneira comprovada de otimizar a comunicação. Levantar-se e movimentar-se ajuda as pessoas a pensar e aprender.

Dica: *Pegue uma caneta e alguns cartões, adesivos e um quadro branco. Faça com que as pessoas se levantem e participem ativamente, desenhando ou movendo o índice dos cartões ou post-its.*

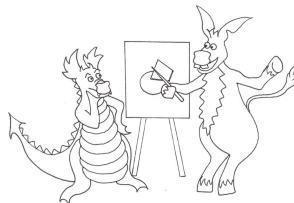


Figura 5.4: Use a visibilidade para criar confiança

Se houver participantes remotos, use ferramentas colaborativas online para ajudar. Nossos times descobriram que usar mapas mentais, seja em um quadro branco físico ou por meio de uma ferramenta colaborativa em tempo real, como Mindmup, pode ser extremamente eficaz em ajudar a identificar o desconhecido e encontrar soluções criativas.

Esses tipos de recursos visuais permitem que os times façam perguntas melhores e mais específicas. Sempre que o time encontra um problema, os membros encontram uma maneira de torná-lo visível, para que eles possam começar a pensar em experimentos para diminuir o problema.

Capítulo 6: Explore Continuamente

Mais times ágeis estão encontrando valor nos testes exploratórios, mas ainda é uma ideia nova ou desconhecida para muitos times. Começaremos explicando o propósito do teste exploratório e o que está envolvido.

Em *Explore It!*, Elisabeth Hendrickson define o teste exploratório como “...projetar e executar testes **simultaneamente** para aprender sobre o **sistema**, usando o seu entendimento do último **experimento** para informar o próximo.”.

Nos testes exploratórios, uma pessoa interage com o sistema e observa o comportamento real, projetando pequenos experimentos. Com base no que ela aprende, ela adapta o experimento e continua aprendendo mais sobre o sistema. No processo, ela pode fazer descobertas surpreendentes, incluindo implicações de interações que ninguém havia considerado. O teste exploratório expõe mal-entendidos sobre o que o software deveria fazer.

Testadores, programadores ou outros membros do time que realizam testes exploratórios precisam estar abertos para observar, aprender, usar habilidades de pensamento crítico e desafiar as expectativas. O objetivo do teste exploratório é reduzir o risco e ganhar confiança no produto. Não há roteiros ou listas de saídas esperadas. Em vez disso, uma meta é identificada, com recursos (ou variações) e uma missão. Os membros do time fazem anotações à medida que exploram, aprendem e discutem com outros membros do time e partes interessadas do negócio posteriormente. Como resultado de uma sessão de teste exploratório, o testador pode mostrar quaisquer bugs encontrados aos colegas de time ou propor novas funcionalidades ou histórias que possam ser necessárias.

O teste exploratório é uma abordagem disciplinada de teste. Não deve ser confundido com o teste ad-hoc, que é feito sem qualquer planejamento ou documentação, ou o teste de macaco (monkey testing), que envolve

inserir entradas aleatórias e ações aleatórias para ver o que quebra. Pense na diferença entre vagar aleatoriamente (talvez perdido) e explorar cuidadosamente (para ter uma visão e com um propósito). Apresentaremos algumas técnicas que podem ajudar a explorar com propósito.

Personas, tarefas e papéis

Testar um novo produto com um novo par de olhos é um presente! Não há tantas pré-concepções sobre como o produto deve se comportar, e o viés de confirmação das pessoas não é tão forte. Um novo par de olhos é mais capaz de observar o produto objetivamente, embora, claro, todos tenham tendências cognitivas inconscientes. Lisa percebeu que, quando forma pares com novos testadores em seu time, eles imediatamente notam bugs que estiveram lá o tempo todo, mas ninguém mais poderia “vê-los”!

Assumir [uma persona](#)⁹, ou um papel, permite que um membro do time teste um produto que eles conhecem de dentro pra fora com uma nova perspectiva; viéses cognitivos inconscientes, tais como [cegueira inatencional](#)¹⁰ e [viés da confirmação](#)¹¹ podem ser superados.

Persona é um usuário fictício que o time cria com características como idade, histórico educacional, experiência, peculiaridades de personalidade, profissão e assim por diante. Alguns times têm um conjunto definido de personas que representam sua base de clientes-alvo que usam ao projetar novas funcionalidades. A Figura 6.1 mostra um tipo de persona de hacker que você pode usar para teste.

⁹<https://www.stickyminds.com/article/how-pragmatic-personas-help-you-understand-your-end-user>

¹⁰http://www.theinvisiblegorilla.com/gorilla_experiment.html

¹¹https://en.wikipedia.org/wiki/Confirmation_bias



Figura 6.1: Persona de Hacker

Combinar personas com tarefas ou papéis é ainda melhor para testes exploratórios. Aqui está um exemplo.

Jill, assistente executiva, tem 30 anos, sempre apressada com coisas demais para fazer, e procura atalhos ao usar o produto. Teste o aplicativo de reserva de hotel do seu time como Jill, que está reservando um hotel de última hora para seu chefe.

Quando um testador assume a persona de Jill, é provável que ele use as características da funcionalidade de uma maneira diferente do que normalmente faria. Por exemplo, ele pode descobrir que clicar no botão “Enviar” várias vezes por impaciência causa reservas duplicadas.

Fluxos de trabalho e passeios

Uma maneira comum de explorar é percorrer os fluxos de trabalho esperados ou jornadas do usuário no aplicativo. Comece com uma jornada e, em seguida, explore as variações dela. Nesse aplicativo de reserva de hotel, uma jornada óbvia é pesquisar um local específico e um intervalo de datas para um determinado número de hóspedes, escolher um quarto e inserir informações de endereço para confirmar. Uma variação dessa abordagem seria tentar inserir um endereço de um país diferente. O formulário aceita vários formatos de código postal? Ele faz a validação do código postal contra o endereço da rua?

Outra abordagem popular é usar passeios (tours). Compare isso a fazer um passeio em um destino de viagem. Como turista, se você for a Paris, pode gostar de ver vários marcos históricos: a Torre Eiffel, o Louvre, o

Arco do Triunfo ou talvez até a Catedral de Notre Dame. Depois de fazer isso, repita o passeio, mas vá para os pontos turísticos em uma ordem diferente. As coisas podem parecer diferentes! A mesma coisa acontece no software. Tentar o [passeio histórico¹²](#) utiliza diferentes funcionalidades e capacidades em diferentes ordens e pode causar um comportamento inesperado.

Dica: Pesquise “passeios de teste exploratório” (*exploratory testing tours*) na Internet e você encontrará muitas ideias diferentes - sugerimos que você crie o seu próprio.



Figura 6.2: Passeio histórico

Riscos e valor para o cliente

Os times costumam ignorar os riscos do negócio ou o que tem valor para o cliente. Sessões de teste exploratório podem ser projetadas para focar nesses aspectos para descobrir suposições ocultas. Fazendo a pergunta: “Qual é a pior coisa que pode acontecer?” pode expor um risco que precisa ser explorado. Por exemplo, se o roubo de dados do cliente for um risco enorme, então o time deseja gastar mais tempo explorando os aspectos de segurança do produto.

O outro lado do risco é o valor, então pergunte: “Qual é a melhor coisa que pode acontecer?” e explore esse valor para o negócio.

Explore em pares ou grupos

A exploração pode acontecer a qualquer momento. Os programadores podem explorar durante a codificação para encurtar seu ciclo de feedback

¹²https://blogs.msdn.microsoft.com/james_whittaker/2009/04/06/tour-of-the-month-the-landmark-tour/

para problemas visuais, ou os testadores podem explorar a compatibilidade do navegador. No entanto, recomendamos parear com alguém para obter o máximo da experiência (Figura 6.3).

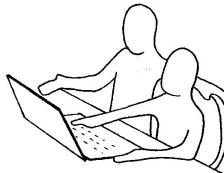


Figura 6.3: Pareamento

Uma maneira de explorar no nível da funcionalidade é com grupos. Um dos times da Lisa costumava reunir pessoas para sessões de teste ad-hoc ou exploratório para ganhar mais confiança em funcionalidades importantes ou arriscadas. Colaborar para testar problemas de concorrência é um ótimo exemplo disso. Mais olhos no problema significa maiores chances de algo ser encontrado.

Muito parecido com a programação em grupos (Mob Programming), o teste em grupos¹³ pode ser usado para teste exploratório. Isso significa que há um motorista (uma função que muda a cada poucos minutos) com várias pessoas ajudando fazendo perguntas ou sugestões. Múltiplas perspectivas podem revelar impactos em outras partes do sistema.

Estatutos (charters)

Em seu livro *Explore It!* Elisabeth Hendrickson detalha como usar estatutos para testes exploratórios eficazes. Os estatutos o ajudam a organizar as informações de que você precisa para aprender sobre sua aplicação em sessões de tempo definido apropriadamente. Eles funcionam bem em combinação com personas, tarefas e funções.

O modelo de Elisabeth se parece com este:

Explore <alvo>

¹³<https://www.stickyminds.com/article/amplified-learning-mob-testing>

Com <recursos>

Para descobrir <informação de valor para alguém>

Levar em consideração os recursos (ou tipos de variações, como Janet gosta de pensar neles) que serão usados é uma boa maneira de começar a escrever um estatuto. Por exemplo, o teste de segurança pode precisar testar vários exploits de formato. Um estatuto pode ser escrito para explorar várias páginas na interface do usuário com esses exploits. O exemplo a seguir mostra uma possibilidade.

Explore a página de inscrição do usuário por 30 minutos

Com exploits de cross-site scripting

Para descobrir quaisquer vulnerabilidades

Gostamos de definir o tempo de nossas sessões para ajudar a focar o estatuto. Uma maneira simples de fazer isso é adicionar o limite de tempo ao próprio estatuto, como em nosso exemplo anterior.

Lisa gosta de apresentar às pessoas os testes exploratórios, fazendo-as formar grupos pequenos para testar brinquedos e jogos para crianças pequenas. Eles primeiro criam uma persona, como: “Judy é uma menina de quatro anos muito forte e ativa”. Em seguida, eles testam um jogo projetado para idades de 3 a 6 anos com um estatuto:

Explore o jogo como a Judy

Usando toda a sua energia, movimentos inesperados e força

Para descobrir se o jogo é seguro ou não para a sua faixa etária

Se eles forem capazes de quebrar um pedaço pequeno, que é um risco de asfixia, o jogo pode não ser seguro. Essa é uma maneira diferente de testar do que se você simplesmente usasse o jogo como um adulto.

Existem outras maneiras de escrever estatutos. Alguns times usam mapas mentais, enquanto outros usam mnemônicos ou simplesmente escrevem uma frase sobre o que desejam explorar.

Executando, aprendendo, guiando

As pessoas geralmente ficam presas tentando escrever ou executar o seu primeiro estatuto.

Dica: *Sugerimos que caso você se encontre nessa situação (preso), não pense muito na primeira vez. Escreva e então tente. Use suas habilidades de observação, seu pensamento crítico e sua intuição.

Conforme os estatutos são executados, o explorador aprende e pode escrever novos estatutos. Nós também encorajamos anotações. Uma vantagem do emparelhamento é que a pessoa que não está dirigindo pode escrever as notas. Também acreditamos que conversar com outros membros do time após a exploração é a chave para aprender e compartilhar informações.

Técnicas adicionais

Outras técnicas nos ajudam a “pensar fora da caixa”. Por exemplo, Mike Talks tem suas [cartas de “Teste Obliquo”¹⁴](#) que podem ajudar o testador a seguir um caminho que pode não ter sido considerado antes. As cartas [TestSphere¹⁵](#) de Beren van Daele também fazem os testadores pensarem e conversarem sobre os seus testes de maneiras diferentes. Como seres humanos, nossos viéses cognitivos inconscientes podem nos impedir de ver problemas importantes. Utilizar cartas como essas podem compensar esses viéses e ajudar os times a serem mais criativos.

Aproveite as ferramentas para uma exploração eficaz

O teste exploratório é centrado no ser humano, mas scripts ou ferramentas automatizadas podem ser usados para gerar dados de teste ou definir o

¹⁴<https://leanpub.com/obliquetesting>

¹⁵<https://www.ministryoftesting.com/dojo/series/testsphere>

cenário. Outras ferramentas também podem ajudar na exploração. Por exemplo, emuladores podem ser usados para dispositivos integrados ou móveis, embora dispositivos reais também precisem ser testados e explorados. Recursos como arquivos de log podem ser usados para detectar falhas “silenciosas” ou possível perda de dados. Por exemplo, um dos times de Janet começou a destacar avisos para torná-los mais visíveis, o que expôs um grande problema em como um método era usado incorretamente. Ferramentas como gravadores podem rastrear quais páginas foram visitadas ou quais dados foram usados, para que possam ser reproduzidos se algo inesperado for encontrado.

Capítulo 7: Testando Atributos de Qualidade

Atributos de qualidade - ou, como algumas pessoas gostam de chamá-los, requisitos não funcionais - costumam ser esquecidos ao discutir uma nova funcionalidade ou história. Um atributo de qualidade define as propriedades sob as quais uma funcionalidade deve operar. Em vez de pensar neles como um “complemento”, preferimos pensar neles como uma restrição que o time deve considerar a cada funcionalidade ou história.

Definindo atributos de qualidade

Dois tipos principais de atributos de qualidade que precisam ser considerados são os atributos de desenvolvimento e operacionais. Os atributos de desenvolvimento incluem manutenibilidade, reutilização e testabilidade do código - o “como” desenvolvemos nosso código. Isso é qualidade interna ou voltada para a tecnologia e é propriedade do time de entrega de software.

Quando as pessoas falam sobre atributos de qualidade, geralmente se referem aos atributos operacionais. Ellen Gottesdiener e Mary Gorman classificam alguns dos atributos de qualidade na Figura 7.1 como atributos operacionais ou de desenvolvimento.

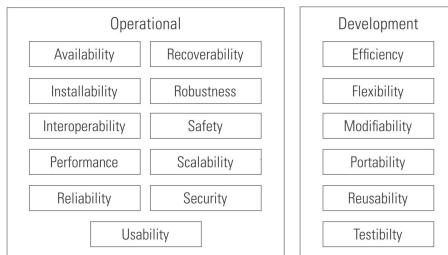


Figura 7.1: Metamodelo de atributos de qualidade

Muitos desses atributos de qualidade são voltados para a tecnologia (consulte o Capítulo 9: Quadrantes de Testes Ágeis para uma explicação). Se as partes interessadas do negócio não os entendem bem, o time de entrega pode ajudá-las a definir os níveis de qualidade adequados para cada atributo.

Mitigar riscos colaborando desde o início

Cada produto ou organização tem necessidades e riscos únicos que precisam ser avaliados. Ao considerar quais atributos de qualidade são importantes para seus clientes, o time pode falar sobre os riscos para o produto. Por exemplo, Janet trabalhou em um time onde a confiabilidade era o atributo de qualidade mais importante (embora não o único). O time se perguntou, “O que precisamos para sermos capazes de provar a confiabilidade?” Ao responder à pergunta, a organização percebeu que precisava investir em um ambiente de teste de confiabilidade completo onde a versão poderia ser implantada no final de cada iteração para executar um conjunto completo de testes automatizados junto com alguns testes exploratórios. O time trabalhou para fazer com que os testes e simulações automatizados funcionassem em todas as histórias.

Alguns times pensam sobre seus atributos de qualidade depois de entregar a funcionalidade, e eles criam histórias para os requisitos “não funcionais”. Isso geralmente não é uma boa ideia porque pode significar ter que voltar e refazer a arquitetura ou o design do código. Esses atributos podem, de fato,

ter maior prioridade do que os requisitos funcionais ou comportamentais. Funcionalidades que tem mais valor em algumas áreas não superam a falta de segurança ou desempenho em alguns domínios de negócios. Os times que esperam muito tarde no ciclo para testar esses atributos (frequentemente durante o final do jogo, pouco antes do lançamento) encontram um obstáculo total. Esses tipos de problemas são geralmente problemas de design e não podem ser corrigidos tão tarde no ciclo de lançamento.

Como um time de entrega, seja proativo. Não espere que o product owner comece a conversa. Ela provavelmente está pensando em alguma funcionalidade e considerando os atributos de qualidade como garantidos. Como um time, considere quais aspectos da qualidade são mais valiosos para os clientes e o o negócio. Uma boa maneira de começar é desenhando um diagrama de contexto da nova funcionalidade proposta para ver como ela interage com outras funcionalidades ou sistemas. Conhecer as dependências e as áreas potencialmente frágeis com antecedência significa que há tempo suficiente para tomar as decisões corretas de design e garantir que o time possui o conhecimento técnico necessário. O time pode planejar fazer um pico (um experimento ou história de investigação) para explorar projetos e arquitetura em potencial.

O planejamento de lançamentos ou funcionalidades oferece ótimas oportunidades para fazer perguntas às partes interessadas da empresa, como estas:

- Qual é a pior coisa que pode acontecer depois de lançarmos essa funcionalidade? Isso a torna de alto risco?
- Está tudo bem se o sistema ou uma funcionalidade do sistema ficar inativa por algum tempo? Se não, qual é o tempo máximo ou porcentagem de tempo que se pode ficar inativo?
- Para um aplicativo baseado na web, quais navegadores os clientes podem usar?
- Podemos presumir que os clientes usarão dispositivos móveis? Isso incluiria telefones e tablets, e isso significa Apple e Android? E sobre...?
- Como saberemos se a funcionalidade foi bem-sucedida assim que a lançarmos?

Planejando o teste de pré-lançamento

Alguns atributos de qualidade podem exigir mais testes imediatamente antes do lançamento, quando todos os componentes do conjunto de funcionalidades estão conectados. Por exemplo, o time pode fazer testes de carga ou desempenho em um ambiente de preparação (Staging) para obter uma linha de base final. Se o time usa [lançamentos azul/verde \(blue-green\)](#)¹⁶ em uma infraestrutura em nuvem, eles podem fazer este teste no ambiente de produção ocioso.

Os times podem usar Feature Flags e outras técnicas para ocultar novas funcionalidades dos clientes até que eles testem vários atributos de qualidade em produção. Uma vez que eles estão confiantes do nível de qualidade de todos os aspectos da funcionalidade, eles podem ativar a funcionalidade. (Veja mais sobre teste em produção no Capítulo 8: Teste em DevOps.)

Planejando para aprendizado posterior

Fazendo a última pergunta - Como saberemos se a funcionalidade foi bem sucedida assim que a lançarmos? - é uma ótima maneira para o time falar sobre o propósito da nova funcionalidade e como eles podem medir se ela cumpre os objetivos desejados. O provisionamento de dados para ferramentas analíticas precisa ser planejado nas histórias de cada funcionalidade. No nível da história, pense em como o time pode monitorar o sistema para medir o uso e a qualidade do atributo. O time pode precisar de novas ferramentas para registro e monitoramento apropriados.

No nível da tarefa, os programadores devem pensar sobre instrumentar o código para que o time possa medir com testes automatizados ou ferramentas de monitoramento (ver Figura 7.2). A solução pode ser tão simples quanto executar um otimizador em uma nova consulta de banco de dados ou usar uma ferramenta de análise estática para verificar se o código atende aos padrões de acessibilidade. Uma abordagem mais holística, como instrumentar cada evento no código para que os problemas

¹⁶<https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html>

de produção possam ser rapidamente identificados e corrigidos, pode ser apropriada.

Preparado	Em Progresso	Para Revisar	Pronto
História 1			
Programar UI	Instrumentar o código		
Criar dados de teste	Criar tabelas do BD		
Enviar logs para... ...	Automatizar testes ...		
História 2			

Figura 7.2: Quadro de tarefas com o futuro em mente

Conformidade regulatória

A conformidade regulatória nem sempre é considerada um atributo de qualidade, mas como os atributos de qualidade que mencionamos, você precisa considerá-la desde o início. Conformidade não significa necessariamente pilhas de documentos, mas geralmente há trabalho extra envolvido para o time, e é aconselhável planejar com antecedência.

As organizações precisam trabalhar em conjunto com auditores e agências reguladoras para entender quais informações são necessárias para demonstrar conformidade. É importante que todos tenham a mesma visão de uma abordagem devidamente disciplinada. Por exemplo, se o time tem testes automatizados que são executados todos os dias e os testes fornecem documentação viva na forma de resultados de teste, eles podem ser usados como evidências para apoiar a abordagem ou cobertura? Ambos de nós trabalhamos com times que precisavam mostrar conformidade (dispositivos médicos e financeiros) e com muito pouco trabalho “extra”. Consulte o Capítulo 21, “Agile Testing in Regulated Environments” em *More Agile Testing* para obter mais exemplos e informações.

Capítulo 8: Teste em DevOps

O desenvolvimento de software sempre incluiu o processo de colocar em produção novas mudanças no software para uso dos clientes. No passado, grande parte desse processo era manual. Existem novas ferramentas e práticas para criar artefatos de software e implantá-los em ambientes de teste e produção. Mas o processo básico é o mesmo. Os times têm muitas atividades de teste para ajudá-los a se sentirem confiantes sobre as mudanças que fazem em seu produto em produção. Existem novos termos para alguns desses testes, mas as habilidades básicas de teste ainda são relevantes.

O movimento DevOps surgiu da ideia de que algumas organizações adotaram o desenvolvimento ágil, mas deixaram todo o time de operações de fora da transição. É também um resultado da mudança para aplicações hospedadas na nuvem e infraestrutura como código, que substituiram as interfaces de linha de comando. Os papéis se adaptaram. Especialistas em operações aprendem a codificar. Os programadores assumem a responsabilidade por seu código, mesmo depois de ele estar em produção, em vez de jogar para o time de operações.

Os testadores também adaptam suas próprias habilidades e atividades. Eles contribuem de várias maneiras, como ajudando a projetar os conjuntos de testes automatizados que fornecem informações confiáveis e valiosas, otimizando o pipeline de entrega e testando o código da infraestrutura para garantir uma execução confiável.

O Capítulo 23 em *More Agile Testing* entra em detalhes sobre como os especialistas em operações podem ajudar o time de entrega a melhorar a qualidade configurando ambientes de teste, ajudando a implementar frameworks de automação de teste, gerando dados de teste e muito mais.

Entrega e implantação contínuas

Os times que praticam a entrega contínua (CD - Continuous Delivery) têm uma versão candidata de lançamento implantável cada vez que uma nova mudança é colocada no repositório de código e subsequentemente passa com sucesso por um pipeline de implantação. O pipeline começa com integração contínua, que pode incluir suítes de testes automatizados em diferentes níveis, como unidade, API e fluxo de trabalho completo por meio da interface do usuário. Pode incluir outras etapas, como análise de código estático para implantações automatizadas em vários ambientes. As partes interessadas do negócio podem decidir implantar a versão candidata para produção e podem fazer isso muitas vezes por dia. A Figura 8.1 mostra um exemplo de pipeline de entrega contínua.

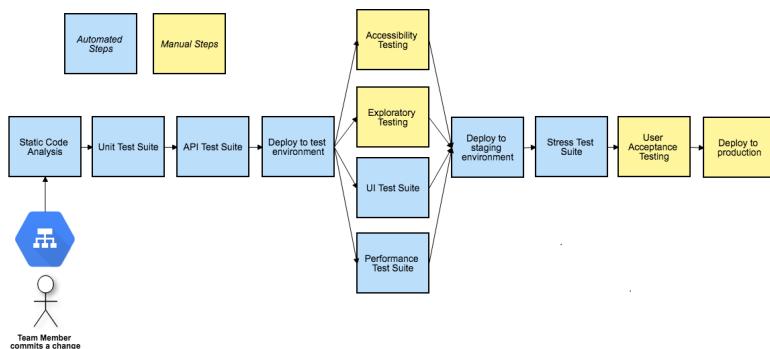


Figura 8.1: Pipeline de entrega contínua

A implantação contínua (também CD - Continuous Deployment) é o mesmo processo, exceto que cada versão candidata a lançamento bem-sucedida é implantada automaticamente em produção. A Figura 8.2 mostra um exemplo de pipeline de implantação contínua.

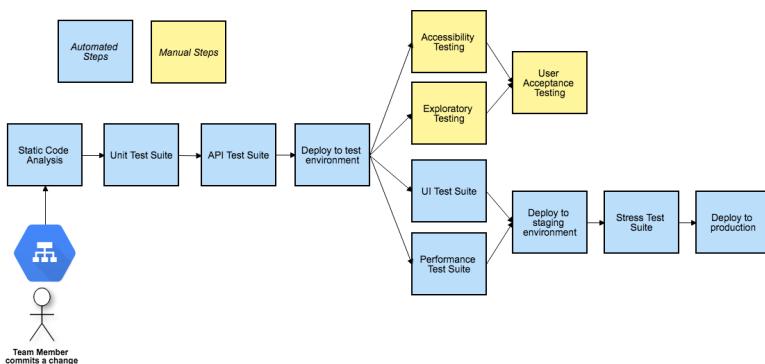


Figura 8.2: Pipeline de implantação contínua

Isso soa tão assustador quanto testar em produção, se você não tiver feito isso antes. Se você liberar várias vezes por dia, como poderá fazer todos os testes de que precisa? Testes centrados em humanos, sejam testes que o time ainda não automatizou ou atividades de teste como testes exploratórios e de acessibilidade, fazem parte de um pipeline de entrega tanto quanto os testes automatizados.

A chave é reconhecer a diferença entre **implantar** e **lançar**.

Graças a técnicas como Feature Flags, é possível ocultar as alterações dos clientes até que todos os testes necessários sejam concluídos. O teste pode ser feito de forma assíncrona.

A entrega ou implantação contínua representa um alto padrão de realização para um time. Os membros do time de entrega e as partes interessadas do negócio precisam criar um entendimento compartilhado de cada funcionalidade e história a ser entregue. Os desenvolvedores dominam as formas de ocultar funcionalidades de alguns (ou todos) clientes até que estejam prontos para lançar essa funcionalidade. O pipeline de entrega deve ser rápido para implantar potencialmente diariamente ou várias vezes por dia. Isso requer uma boa infraestrutura, o que significa mais código para construir e testar. Existem muitas novas habilidades para dominar. Quando cada membro do time traz suas **habilidades em forma de T¹⁷** e todos colaboram, o time pode resolver problemas mais facilmente e continuar a encurtar os ciclos de feedback representados no pipeline.

¹⁷<https://lisacrispin.com/tag/t-shaped-skills/>

Testando em produção

Ao mesmo tempo, o termo “teste em produção” soava como “Coloque o código em produção e deixe os clientes encontrarem os bugs e nos contarem” ou “Vamos testar em produção e esperar que não afetemos a conta de ninguém”. Infelizmente, alguns times fizeram exatamente isso. Hoje, as palavras “teste em produção” têm uma conotação menos assustadora. Testar em produção se tornou uma necessidade em muitos casos, mas isso não significa lançar código de baixa qualidade e permitir que os clientes encontrem os bugs!

Os testes em produção ajudam as empresas de várias maneiras. Normalmente, é impossível criar um ambiente de teste que se pareça exatamente com o de produção. Não há uma maneira fácil de saber realmente o que o software fará até que esteja no ambiente de produção.

Técnicas como Feature Flags permitem que os times “ativem” funcionalidades específicas para clientes específicos para obter feedback rápido. Às vezes, isso é chamado de versão de aprendizado ou produto mínimo viável (MVP - Minimum Viable Product). O teste A/B pode ser a técnica de teste em produção mais conhecida, mostrando designs diferentes para pessoas diferentes e julgando quais levam à maioria dos “cliques” ou vendas.

Análise e rastreamento (Tracing) sofisticados podem mostrar detalhes como o modo como um usuário individual navega pelo aplicativo ou agregar estatísticas como a porcentagem de clientes que está usando uma funcionalidade específica. Remover funcionalidades não utilizadas pode ser tão importante quanto adicionar novas funcionalidades populares, uma vez que cada linha de código tem um custo de manutenção e aumenta o risco. O teste em produção envolve monitoramento e observação.

Monitoramento e observabilidade

A importância de monitorar a integridade do sistema em produção existe desde que os sistemas de software passaram a registrar informações pertinentes sobre os eventos do sistema. Os times podem configurar alertas para certos tipos de erros ou para exceder um orçamento de erro - por exemplo, “Disparar um alerta quando o número de erros 503 aumentam

em 10% sobre a média.” Quando alertados, os membros do time vasculham os arquivos de log e analytics para investigar o problema e resolvê-lo.

Os testadores profissionais sabem que não é possível prever todos os erros possíveis que podem ocorrer em produção! Nos últimos anos, isso deu origem a uma nova prática chamada de observabilidade, muitas vezes referida como “o11y”, que representa os 11 caracteres entre o “o” e o “y” no alfabeto inglês. Os times que praticam o11y instrumentam e registram cada evento em seu código de produção para que possam ser analisados por ferramentas apropriadas quando necessário. Usando ferramentas e experimentos, os times estudam informações de dados de log estruturados, métricas e rastreamentos para aprender coisas diferentes sobre seu produto além do que pode ser aprendido em um ambiente de teste. Esta é uma área em que os testadores, com sua capacidade de perceber padrões incomuns e identificar riscos, contribuem com valor. Como Abby Bangser¹⁸ disse, é uma forma de teste exploratório apoiada por ferramenta.

A observabilidade permite que os times respondam rapidamente quando um usuário relata um problema que o time nunca viu antes ou quando as métricas do sistema mostram padrões incomuns que indicam um problema potencial. Os dados de log estruturados permitem que o time rastreie a atividade do usuário e identifique rapidamente onde o sistema começou a se comportar incorretamente. Isso permite que os times revertam uma alteração ou implantem uma correção, geralmente em questão de minutos. Essa capacidade de se recuperar tão rapidamente de falhas em produção permite que o time libere essas pequenas mudanças contínuas no produto sem medo, e é isso que torna a entrega ou implantação contínua uma prática segura.

A nova tecnologia nos traz novas capacidades

Nos últimos anos, o armazenamento de “Big Data” tornou-se acessível até mesmo para pequenas empresas. Os times podem registrar informações sobre todos os eventos que ocorrem em qualquer ambiente de teste

¹⁸<https://club.ministryoftesting.com/t/power-hour-curious-stuck-or-need-guidance-on-devops-or-observability/24963/3>

ou produção. Tecnologia poderosa, incluindo inteligência artificial (IA) e aprendizado de máquina (ML - Machine Learning), acelerou o processamento e a análise dessas enormes quantidades de dados. Podemos aprender rapidamente como os clientes estão usando nossos produtos, descobrir problemas antes que eles o façam e nos recuperar rapidamente de falhas.

O teste em produção é apenas uma parte da abordagem de qualquer time para construir qualidade em seu produto. Eles ainda planejam e executam todas as atividades de teste apropriadas junto com a escrita do código de produção. Eles também podem observar o uso de produção e executar testes sem riscos em produção para adicionar outro nível de confiança e confiabilidade. A Figura 8.3 é um gráfico maravilhoso de Cindy Sridharan de seu artigo *Monitoramento e Observabilidade*¹⁹ e representa como os times testam tentando simular o ambiente de produção, monitoram falhas previsíveis em produção e usam a observabilidade para detectar qualquer coisa que passe por esses outros esforços guiados pela qualidade.

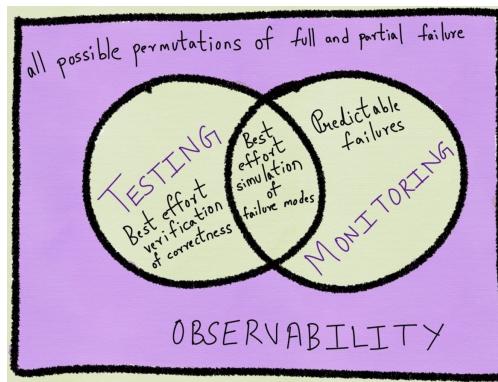


Figura 8.3: Teste, monitoramento e observabilidade de Cindy Sridharan

¹⁹<https://medium.com/@copyconstruct/testing-in-production-the-safe-way-18ca102d0ef1>

SEÇÃO 3: Modelos Úteis

Descobrimos que modelos visuais são um ingrediente essencial para ajudar os times a planejar e executar todas as atividades de teste necessárias e formular uma estratégia de automação de testes eficaz. Nesta seção, explicamos como utilizar os Quadrantes de Teste Ágil para identificar todos os tipos de testes que são necessários para cada novo conjunto de funcionalidades ou histórias e garantir que sejam feitos quando eles são mais eficazes. Em seguida, veremos como usar modelos visuais como a Pirâmide de Automação de Teste para guiar as conversas do time sobre uma estratégia de automação realista que funciona para seu contexto.

- Capítulo 9: Os Quadrantes de Testes Ágeis
- Capítulo 10: Visualizando uma Estratégia de Automação de Teste

Capítulo 9: Os Quadrantes de Testes Ágeis

Brian Marick escreveu pela primeira vez sobre os quadrantes de testes ágeis (Figura 9.1) em 2003, quando Lisa e Janet estavam tentando descobrir como falar sobre testes para a comunidade ágil, especialmente aos testadores. Naquela época, a maioria das pessoas ágeis estava discutindo apenas “testes de cliente” e “testes de programador”. Muitos times estavam focados apenas em testes de aceitação funcional - como as funcionalidades devem se comportar. Hoje, isso ainda é uma armadilha em potencial para times em transição para o Ágil. Os quadrantes nos deram uma maneira de discutir todos os diferentes tipos de teste que um time pode precisar considerar. Eles nos ajudam a ver o panorama geral.

Os quadrantes de testes ágeis são uma taxonomia de diferentes tipos de testes. Nós os usamos como uma ferramenta de raciocínio para ajudar os times a discutir quais atividades de testes eles podem precisar e garantir de que eles tenham as pessoas, recursos e ambientes certos para realizá-los.

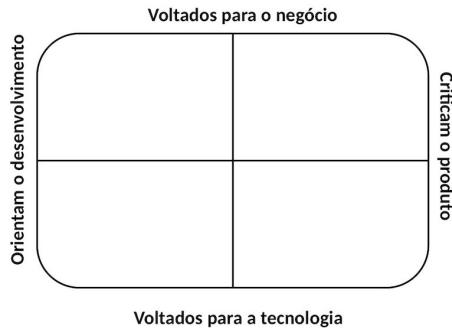


Figura 9.1: Quadrantes de Testes Ágeis

Os testes do lado esquerdo são aqueles que orientam o desenvolvimento, aqueles que são escritos antes que a codificação aconteça ou simultaneamente à medida que a codificação prossegue. Os testes do lado direito são

aqueles que criticam (avaliam) o produto após a conclusão da codificação. Testes à esquerda ajudam a prevenir defeitos. Os testes à direita encontram defeitos no código ou talvez identifiquem funcionalidades ausentes.

A metade superior dos quadrantes concentra-se em testes que podem ser lidos pelas partes interessadas do negócio. Esses testes respondem à pergunta: “Estamos construindo a coisa certa?” A metade inferior inclui testes que são escritos por e para membros do time técnico. As partes interessadas do negócio provavelmente se preocupam com os resultados finais, mas eles não tentariam ler os testes. A metade superior é sobre a qualidade externa, conforme definido pela empresa. A metade inferior é sobre código interno ou correção de infraestrutura.

Os quadrantes são numerados para facilitar a referência. Os quatro quadrantes são rotulados como:

- **Quadrante 1 (Q1):** Testes voltados para a tecnologia, que orientam o desenvolvimento
- **Quadrante 2 (Q2):** Testes voltados para negócios, que orientam o desenvolvimento
- **Quadrante 3 (Q3):** Testes voltados para negócios, que criticam o produto
- **Quadrante 4 (Q4):** Testes voltados para a tecnologia, que criticam o produto

O modelo de quadrantes de testes ágeis ajuda os times a pensar nas atividades de testes necessárias para dar confiança ao produto que estão construindo. Também ajuda a construir uma linguagem de teste comum com o time e com a organização, se usado para ajudar na comunicação entre os times. A coisa favorita de Janet sobre este modelo é que ele não apenas representa uma visão holística dos testes, mas também torna visível a responsabilidade do time inteiro pelas atividades de teste.

Cada time tem sua própria combinação distinta de domínio de negócios, produto, habilidades, maturidade do produto, pilha técnica, supervisão regulatória e muito mais. O modelo pode ser aplicado para representar os testes necessários em cada contexto. Na Figura 9.2, compartilhamos alguns tipos típicos de testes que podem ser encontrados em cada quadrante.

Dica: Lembre-se: Use estes exemplos como uma orientação. É uma ferramenta, não uma regra, e existem muitas áreas cintzentas. O contexto do seu time é único, e seus quadrantes também devem ser. Aplique o modelo ao teste que você precisa fazer.

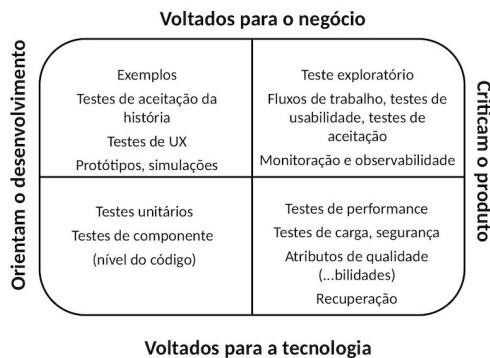


Figura 9.2: Exemplos de tipos de teste para os quadrantes

Quais testes em qual ordem?

Numeramos os quadrantes 1, 2, 3, 4 simplesmente para facilitar a referência. É complicado dizer “Testes voltados para negócios, que orientam o desenvolvimento”, então dizemos “Quadrante 2” ou “Q2”. Os números não pretendem representar que os tipos de atividades de teste devem ser realizados nesse ordem. Conforme o time planeja seus testes, ele pensará sobre o momento apropriado para fazer cada atividade de teste.

Aqui está um exemplo. Um time decide começar a usar uma nova arquitetura para as novas funcionalidades daqui para frente. Ele precisa ter certeza de que a arquitetura será dimensionada de forma adequada para acomodar um certo número de usuários e uma carga potencial no sistema. O time faz um “Spike”, que significa que eles escrevem algum código descartável apenas com o propósito de testar a arquitetura. Eles fazem testes de carga e desempenho no “Spike” para ver se atende aos requisitos de tempo de resposta e permanece estável. Se ficarem satisfeitos,

eles excluem o Spike e começam a trabalhar nas histórias para uma nova funcionalidade, desta vez seguindo suas práticas de desenvolvimento usuais. Se eles não estiverem satisfeitos, eles podem repensar a arquitetura e repetir o processo.

Para o desenvolvimento de funcionalidades, a maioria dos times provavelmente começa no Q2. Eles podem desenhar protótipos no papel e mostrá-los a clientes em potencial. O time de entrega e negócios pode ter um workshop de mapeamento de histórias ou especificação para discutir funcionalidades e dividi-los em histórias. Durante o refinamento do backlog ou workshops de preparação para a história, o time pode usar atividades como mapeamento de exemplos para extrair mais detalhes.

Assim que o time começa a desenvolver uma história, eles trabalham nos testes do Q1, escrevendo testes de unidade como parte do desenvolvimento guiado por testes. Eles podem estar trabalhando simultaneamente em testes de história no Q2. Se o suficiente de uma funcionalidade for entregue para explorar, eles podem passar para os testes do Q3. No entanto, se a segurança é a preocupação número um para os negócios e clientes, o teste de segurança no Q4 pode ser priorizado em relação ao teste funcional da história do Q2. Cada time encontra sua própria maneira de trabalhar, e isso pode mudar para diferentes tipos de funcionalidades ou projetos.

Usando os quadrantes

Janet e Lisa viram times usarem os quadrantes de várias maneiras. Alguns times colocam um pôster na parede com os quadrantes em branco. Enquanto eles planejam uma nova funcionalidade ou release, eles falam sobre todos os testes de que precisam e escrevem cada tipo no quadrante apropriado. Eles usam isso como um lembrete sobre o que eles precisam fazer ou o que podem ter esquecido.

Quadrante 1

Os times também podem usar os quadrantes para falar sobre quais testes devem ser automatizados. Os testes do Q1 são normalmente automatizados pelos desenvolvedores que escrevem o código de produção. Muitos times praticam o desenvolvimento guiado por testes (TDD), escrevendo um

pequeno teste de unidade para alguma pequena funcionalidade, então o código para fazer esse teste passar. Os testes Q1 são projetados para serem executados rapidamente já que eles testam uma pequena área de código e geralmente não incluem interação com outras camadas do aplicativo ou bancos de dados. Eles fornecem aos times o feedback rápido necessário para fazer alterações no código de forma rápida e sem medo.

Quadrante 2

Os testes voltados para os negócios, que orientam o desenvolvimento no Q2 são uma base importante para a maioria dos times. Product owners, desenvolvedores, testadores e outros se encontram com frequência para planejar funcionalidades e histórias. Eles podem usar técnicas, como mapeamento de exemplo para obter regras de negócios para cada história, juntamente com os exemplos que os ilustram. Os times que praticam o desenvolvimento guiado por comportamento (BDD), desenvolvimento guiado por testes de aceitação (ATDD), ou especificação por exemplo (SBE) transformam isso em cenários que especificam o comportamento como testes executáveis. Esses cenários podem ser automatizados à medida que o código é escrito.

Quadrante 3

Os testes no Q3 tendem a ser centrados no ser humano, descobrindo se as histórias e funcionalidades fornecem o valor pretendido para os clientes. A automação pode ser usada para facilitar esse teste por meio de dados ou configuração de estado. Está se tornando comum para os times fazerem testes exploratórios em produção, usando feature flags de release para “ocultar” novas funcionalidades dos clientes até que o teste seja concluído. Os testes do Q3 incluem formas de teste em produção, como monitoramento para entender o que realmente acontece e como os clientes usam as funcionalidades. As informações aprendidas com os testes Q3 retornam ao Q2, geralmente resultando na criação de novas histórias ou funcionalidades.

Quadrante 4

Muitos testes do Q4 dependem de automação e ferramentas, mas alguns podem exigir testes adicionais. Por exemplo, as ferramentas automatizadas para acessibilidade (geralmente abreviado para “a11y”, representando as letras inicial e final e o número de letras entre elas na palavra “Accessibility”) ainda não são tão eficazes quanto o teste exploratório manual para essas funcionalidades. Os resultados desses testes geralmente retornam às atividades do Q1 conforme o time altera o design do código para melhorar vários atributos de qualidade. Monitorar o desempenho e os erros em produção, ou testar a capacidade de recuperação, também podem ser considerados um tipo de teste que se enquadra no Q4. A tecnologia de hoje tornou viável e seguro testar em produção. Isso não significa que permitimos que os clientes encontrem bugs para nós, mas que aprendemos com certeza como o código se comporta em um verdadeiro ambiente de produção. (Detalhes sobre o teste em produção podem ser encontrados no Capítulo 8.)

Definindo “pronto”

Use os quadrantes para definir o que “Pronto” significa para o seu time. Muitos times têm dificuldade em decidir quais testes devem ser incluídos nessa definição. No Capítulo 3, falamos sobre os níveis de detalhe, e aqueles níveis se aplicam aqui. Em vez de definir “Pronto”, incentivamos os times a serem mais específicos e chamá-lo de “**História Pronta**”. Os testes que podem ser incluídos em “História Pronta” provavelmente seriam todos do Q1, todos do Q2 e talvez alguns do Q3, como testes exploratórios.

Vá um passo além e defina “**Funcionalidade Pronta**”, que incluiria todos os testes de histórias, mas talvez também inclua testes como teste de aceitação do usuário (User Acceptance Testing) e teste exploratório no nível da funcionalidade. Recomendamos que todos os atributos de qualidade que não puderam ser testados no nível da história sejam executados no nível da funcionalidade.

Você pode até escolher definir “**Release Pronto**”. Isso inclui todos os testes que se aplicam ao seu contexto, de cada quadrante.

Dica: Lembre-se, quando falamos sobre “pronto” nos níveis da história, funcionalidade, e release, não exigimos que os testes sejam feitos em uma determinada ordem. Em vez disso, considere quais testes guiam o desenvolvimento (prevenindo defeitos) e quais estão criticando o produto (encontrando defeitos no código).



Figura 9.3: Localizando e “corrigindo” bugs

Encontre os modelos que se encaixam no seu contexto

Muitos times consideram os quadrantes de testes ágeis úteis no planejamento das suas atividades de teste. Outros adaptaram o modelo de quadrantes para melhor atender às suas necessidades, e há muitas variações úteis dos quadrantes. Você pode encontrar o Capítulo 8 de *More Agile Testing* disponível para download em agiletester.ca²⁰, e ele inclui várias adaptações dos quadrantes. Verifique nossa lista de recursos para mais informações.

Qualquer que seja o modelo que funcione melhor para você, certifique-se de mantê-lo visível e usá-lo para estimular conversas sobre como melhorar continuamente seus testes.

²⁰<https://agiletester.ca>

Capítulo 10: Visualizando uma Estratégia de Automação de Teste

A automação de testes é um desafio constante para os times de software em todos os lugares. Muitos times ainda não têm testes de regressão automatizados. Alguns times sentem que já dominaram o processo, mas então atualizam seus produtos para incorporar novas tecnologias que suas ferramentas de automação existentes não conseguem lidar. Freqüentemente, eles lutam para manter o equilíbrio entre valor e custo de manutenção.

Onde quer que um time esteja em sua jornada de automação, é útil dar um passo para trás e pensar sobre as prioridades e quais melhorias focar em seguida. Este capítulo não pretende fornecer uma introdução extensiva à automação, mas mostrar como o uso de modelos visuais pode ajudar os times a projetar sua estratégia de automação.

Usando modelos visuais

Envolver todo o time na formulação de uma estratégia para atender às diferentes necessidades de automação e executar essa estratégia é a chave para o sucesso com a automação. Os modelos visuais ajudam a guiar essas conversas.

O modelo de quadrantes de teste ágil abordado no Capítulo 9 pode ajudar os times a planejar sua estratégia de automação enquanto discutem os diferentes tipos de teste que são necessários, bem como quais habilidades, ferramentas e infraestrutura eles precisarão para concluir-los. Neste capítulo, examinamos alguns modelos adicionais que podem ajudar os times a encontrar uma estratégia de automação bem-sucedida.

Dica: Lembre-se, essas são ferramentas de raciocínio, para serem usadas para iniciar conversas sobre como seu time deseja automatizar testes.

A pirâmide clássica de automação de teste

A pirâmide de automação de teste de Mike Cohn tem ajudado muitos times desde o início dos anos 2000. Nós o ajustamos ligeiramente desde então (Figura 10.1) para deixar nossa intenção clara, incluindo a bolha da nuvem no topo para representar que nem todos os testes de regressão podem ser automatizados. Às vezes, precisamos de testes centrados no ser humano, que incluem testes exploratórios (TE).

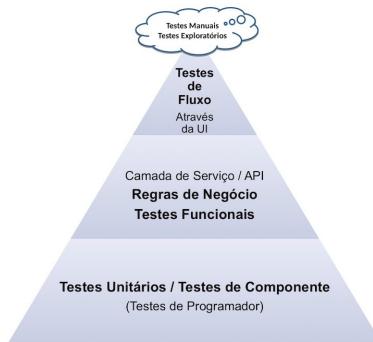


Figura 10.1: A clássica Pirâmide de automação de teste

Esse modelo ajuda os times a entender que, na maioria dos contextos, vale a pena automatizar os testes no nível mais granular da aplicação possível, para fornecer proteção adequada contra falhas de regressão. Os times que praticam o desenvolvimento guiado por teste (TDD) constroem uma base sólida de testes em nível de unidade e componente que ajudam a guiar o design do código. Esses testes são executados muito rapidamente, por isso fornecem um feedback rápido ao time.

Com a maioria dos aplicativos, é necessário testar as interações entre as diferentes camadas da arquitetura. Por exemplo, a lógica de negócios

geralmente requer interação com o banco de dados. Fazer o máximo de automação desse tipo no nível de serviço ou API sem passar por uma interface de usuário (UI) geralmente é a maneira mais eficiente.

Algumas regressões ocorrem apenas quando duas ou mais camadas do aplicativo estão envolvidas. Isso pode exigir testes de fluxo de trabalho por meio da UI que envolvem o servidor, banco de dados e/ou um sistema externo. Na maioria dos contextos, é melhor minimizar o número de testes de fluxo de trabalho de ponta a ponta. Esses testes são executados lentamente, geralmente são os mais frágeis e geralmente exigem mais manutenção. A nuvem no topo da pirâmide inclui atividades centradas no ser humano, como testes exploratórios e outras tarefas que não podem ser automatizadas.

A pirâmide de automação de teste nos ajuda a pensar em maneiras de “empurrar os testes para baixo”, maximizar os testes de regressão que são isolados em uma parte de um aplicativo e minimizar aqueles que envolvem várias partes do sistema. Quando os times planejam o teste de uma funcionalidade, eles podem olhar para a pirâmide para ver onde cada teste pode ser automatizado de forma mais adequada.

A pirâmide clássica não significa que haja um certo número ou porcentagem de testes automatizados em cada nível. Seb Rose concebe o modelo de maneira um pouco diferente, conforme mostrado na Figura 10.2.

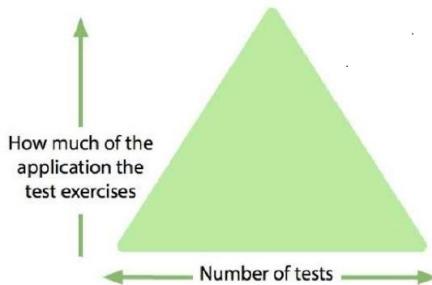


Figura 10.2: Versão da pirâmide de Seb Rose, número de testes vs. cobertura do teste

O modelo de Seb esclarece que o que torna um teste mais caro é o número de camadas do aplicativo que ele requer para ser executado. Por exemplo, é

possível ter um teste de nível de unidade da UI que não envolve nenhuma outra camada do aplicativo. É possível usar TDD para cada camada isolada do aplicativo, seja o servidor, a API, a UI ou um microsserviço.

Houveram muitas adaptações do modelo da pirâmide (e sim, o clássico é realmente um triângulo) ao longo dos anos. O capítulo 15 de *More Agile Testing* inclui várias adaptações, incluindo as de Alister Scott e Sharon Robson.

Os times que possuem testes automatizados podem desenhar a “forma” de sua pirâmide para visualizar onde seus testes atuais se encaixam. Muitos times começam principalmente com testes de UI e uma pirâmide “de cabeça para baixo” ou “casquinha de sorvete²¹”. Outros podem ter uma ampulheta. Nenhuma dessas formas está necessariamente errada, mas se a automação atual não atender às necessidades do time, os visuais podem ajudar a imaginar quais mudanças são necessárias.

As conversas em torno de um modelo visual ajudam os times que estão apenas começando seus esforços de automação a decidir seu objetivo final e suas primeiras prioridades.

Testes automatizados como documentação viva

O tempo, o custo e o esforço necessários para automatizar diferentes tipos de testes em diferentes níveis do aplicativo não são as únicas considerações ao montar uma estratégia de automação. Outra consideração é lembrar quem precisa ser capaz de ler e entender os testes. O iceberg de automação de teste de Seb Rose (Figura 10.3) nos lembra que um dos atributos mais valiosos dos testes automatizados é a documentação viva que eles fornecem. Eles estão sempre atualizados porque o time os mantém passando o tempo todo, então você pode dizer a qualquer momento exatamente o que o seu sistema faz.

²¹<https://watirmelon.blog/testing-pyramids/>

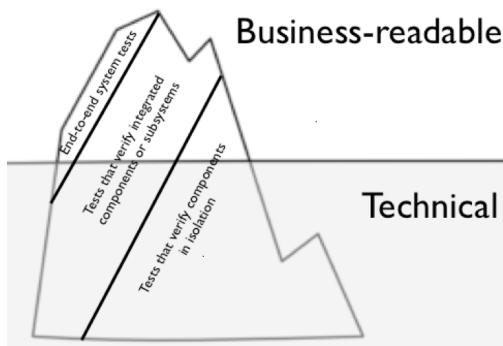


Figura 10.3: Iceberg de automação de teste de Seb Rose

As partes do iceberg acima da linha da água são testes legíveis para as pessoas de negócios, enquanto as que estão abaixo não (são escritas em uma linguagem técnica). A quantidade de testes varia de acordo com o time; por exemplo, Lisa trabalhou em um time cujo produto se destinava a outros times de entrega. Todos no time, incluindo o product owner, podiam entender os testes de regressão automatizados escritos em código de baixo nível. Eles não precisavam de testes “legíveis para as pessoas de negócios”. Em outros domínios, é fundamental que as partes interessadas do negócio possam entender os testes de aceitação. Este modelo ajuda a nos lembrar de pensar sobre o que é necessário no contexto do time.

Estendendo o modelo

Mesmo a cobertura de teste de regressão automatizada mais diligente pode falhar ao identificar algumas falhas de regressão. Nenhum ambiente de teste é exatamente como em produção. Alguns bugs podem ser encontrados por meio de “testes em produção”, conforme discutido no Capítulo 8.

Em seu livro *A Practical Guide to Testing in DevOps*, Katrina Clokie apresenta seu filtro de bug DevOps. É um visual útil que mostra que os testes de unidade só podem filtrar os pequenos bugs, enquanto diferentes níveis de integração e testes ponta a ponta detectam outros cada vez

maiores. Para encontrar os bugs totalmente formados, os times precisam de logs, alertas e monitoramento para o seu sistema de produção.

Responsabilidade compartilhada

Encorajamos os times a compartilhar a responsabilidade de testar as regras de negócios e níveis mais elevados de integração no nível da API. Os testadores também devem ter visibilidade dos testes de unidade e de componentes escritos pelos desenvolvedores. Como membro do time, é importante entender os aplicativos de seu time e o funcionamento interno ao abordar sua estratégia de automação.

Como a automação de testes por meio da UI tende a ser mais demorada, há uma tentação de entregar isso a um time de automação separada ou deixar que os testadores do time assumam total responsabilidade por isso. Recomendamos que os desenvolvedores, que são bons em escrever códigos eficientes e sustentáveis, trabalhem em conjunto com os testadores, que são bons em especificar casos de teste, para automatizar testes por meio da UI, bem como todas as outras camadas acima do nível básico da pirâmide.

Lembre-se, como todos os outros modelos, a pirâmide de automação de teste é um guia. Os times que reúnem membros em todas as funções para fazer perguntas, conversar sobre as respostas, desenhar no quadro branco e projetar experimentos têm o melhor sucesso com a automação. Modelos visuais como os exemplos neste capítulo ajudam os times a falar sobre por que estão automatizando os testes, quais são seus maiores pontos fracos de automação, como pessoas com diferentes habilidades podem ajudar e quais devem ser seus próximos experimentos. Em nossa experiência, uma abordagem passo a passo funciona melhor.

SEÇÃO 4: Teste Ágil Hoje

O básico do teste ágil - como o uso da abordagem do time inteiro, guiar o desenvolvimento com exemplos e colaborar entre funções para a construção da qualidade - são tão eficazes hoje como foram há 20 anos. Existe algo que precisamos mudar para enfrentar os desafios de hoje? Nessa seção compartilhamos o que alguns dos principais profissionais de teste ágil veem mudando para o papel dos testadores. Vamos encerrar com vários ingredientes para ajudar os times a terem sucesso com testes ágeis.

- Capítulo 11: O Novo Papel de um Testador
- Capítulo 12: Ingredientes Para o Sucesso

Capítulo 11: O Novo Papel de um Testador

Muitos times lutam com apenas um testador como parte do time - ou ainda pior, um testador que oferece suporte a mais de um time de entrega. Se o testador for o único realizando atividades de teste, geralmente ele cria um gargalo. Times ágeis que entregam mudanças em produção a cada iteração (ou ainda mais frequentemente se eles estiverem implementando entrega contínua) não podem ter um único testador para fazer todos os testes.

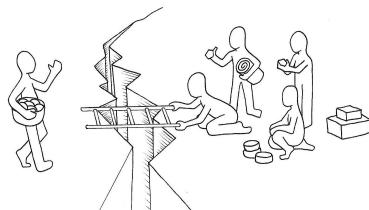
Não podemos prever o futuro, mas é informativo olhar ao redor para ver como o papel de um testador está mudando. Perguntamos a outros coaches experientes de teste ágil e instrutores para descobrir o que eles pensam sobre a mudança do papel de um testador. Essas diferentes perspectivas podem ajudá-lo a entender como os testadores e os times podem se adaptar e ajudar a construir uma cultura de qualidade.

Testadores são a cola de qualidade do time

Alex Schlaubeck - Alemanha

Quando comecei, a função de testador em um time (se o testador fizesse parte do time) costumava ser a única responsável pela automação da UI e testes manuais. Ao longo dos últimos 12 anos, tenho visto uma grande diversificação da função, e sempre de um forma dependente do contexto com base em como o time operava. Vejo testadores trabalhando em mais tarefas de automação, até mesmo trabalhando em pares no nível da unidade com desenvolvedores. Eu os vejo envolvidos em todos os pontos do processo. Eu vejo eles organizando sessões de teste de mob com o time para realizar testes exploratórios. Eu os vejo fazendo campanha por melhores ciclos de

feedback. Eu até vi testadores começarem a corrigir bugs ou implementar funcionalidades.



Faça conexões

Para mim, essa diversificação e indefinição de papéis é uma grande liberdade e uma grande responsabilidade. Isso significa que temos que nos perguntar: “Como eu, minhas habilidades e meu potencial para aprender se encaixam melhor no contexto deste time?”. Nós nos tornamos muito “cola de qualidade e comunicação”, identificando e preenchendo as lacunas de qualquer time.

Comecei a usar o termo “engenheiro de qualidade integrado” ou “consultor de qualidade integrado” para esta função. O problema com o título “testador” é que contém o nome de uma das muitas atividades que fazemos, então você ouve perguntas como: “Se todos estão envolvidos no teste, por que precisamos de um testador?” ou declarações como “Os desenvolvedores estão automatizando os testes, então não precisamos de uma função de testador”. O teste é apenas uma das muitas coisas que um testador faz. Na minha opinião, precisamos lutar contra a ideia que um time ágil deve consistir em “quimeras”: uma mistura de diferentes funções ou um membro do time canivete suíço que pode fazer tudo: requisitos, UX, teste, segurança, front e backend. Os times ágeis precisam ser diversificados e multifuncionais, o que significa que precisamos de pessoas com diferentes origens, interesses e especializações, sem que ninguém seja um silo ou gargalo. Eu acho que é um equilíbrio que pode ser alcançado.

Vejo a função de testador consistindo em várias atividades. Existem coisas que sempre fizeram parte da função, como trabalhar com as partes interessadas, trazer experiência para atividades de teste, trabalhar em pares com desenvolvedores e outros testadores, apoiar o product owner, organizar e adaptar a estratégia geral de qualidade, obtendo bons dados de teste

e identificar riscos. Acho que haverá ainda mais tarefas que podem ser assumidas ou suportadas por testadores no futuro. Alguns deles podem ser: trabalhar com o time para garantir a testabilidade e observabilidade para teste e monitoramento em produção, fazer perguntas ao sistema de produção para explorar como ele está sendo usado, aprimorar nosso desempenho e ensinar habilidades para testes exploratórios, ajudando o time a focar no valor (e às vezes no minimalismo, ou seja, o valor de algo não feito). Também vejo testadores começando a adicionar “saúde do time” aos seus atributos de qualidade, olhando para os níveis de comunicação e estresse do time como um todo. Afinal, são coisas que podem afetar em muito a qualidade.

Resumindo, acredito que a função de testador continua sendo importante. Eles são as pessoas do time cuja principal prioridade é a qualidade. Eles também são as pessoas com paixão por qualidade e testes, e eles defendem e lutam por eles.

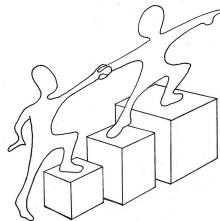
Jornada profissional de um testador ágil

Paul Carvalho - Canadá

Quando treino times ágeis, ajudo o time inteiro a aprender a trabalhar em conjunto e a desenvolver os pontos fortes uns dos outros. Um product owner traz conhecimento de negócios e da indústria, um programador traz forte habilidades de codificação e desenvolvimento, um designer traz insights sobre as perspectiva e experiência do usuário, e um testador traz algo de valor para a mesa também.

A jornada profissional de um testador começa com o afastamento de testes acidentais e aleatórios, para um design inteligente de testes por meio de modelos, técnicas e outras habilidades e conhecimentos especializados. O ágil concentra-se fortemente no trabalho próximo a outras pessoas, então os testadores precisam pensar além ao pensar em testar e encontrar sua voz para ajudar os outros membros do time a compreenderem as muitas maneiras de gerar informações de qualidade sobre os sistemas em desenvolvimento. Um ótimo testador ágil passa mais tempo com um marcador de quadro branco na mão e trabalhando em pares com outros membros do

time do que qualquer outra coisa. É ajudar o resto do time a ver e entender mais sobre o sistema, antes que ele seja construído. “Build Quality In” (Construir com qualidade) é mais do que um slogan; é uma realidade que grandes testadores podem ajudar a habilitar times colaborativos de alto desempenho.



Mentor

A jornada profissional de um testador ágil progride da forma aleatória de testar, para a compreensão das técnicas e design de bons testes, para encontrar uma voz e expressar essas ideias para que possam ajudar a elevar a performance do resto do time.

Como testador, tente aprimorar e aumentar a compreensão de todos sobre os sistemas e soluções por meio de exploração cuidadosa. Se você abandonar a noção de que você deve ser o único a escrever casos de teste ou automação de testes, pense sobre onde suas habilidades e percepções exclusivas podem ajudar a conduzir seu time.

O caminho fascinante da evolução como testadores

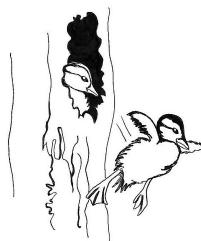
Claudia Badell - Uruguai

Como testadores, podemos contribuir e agregar valor a partir de diferentes perspectivas: como facilitadores e evangelistas do teste e qualidade em um time de produto, como coaches, como consultores de teste, como especialistas em certos tipos de teste (teste de usabilidade, teste de acessibilidade, teste de segurança, teste de desempenho, entre outros) e muito mais.

Os testadores não são mais vistos como guardiões da qualidade, então podemos ser vistos e valorizados como defensores da qualidade.

Hoje em dia é cada vez mais frequente que os testadores façam parte do time de desenvolvimento e façam contribuições desde o início do processo de desenvolvimento. Na minha experiência, o papel de um testador neste contexto está evoluindo. Além de realizar atividades de teste para oferecer suporte a testes manuais e verificações automatizadas, os testadores colaboram para construir pontes dentro do time, a fim de chegar a um entendimento e engajamento comum sobre os testes. Eles também definem, acompanham e ajustam as estratégias de teste a serem aplicadas pelo time inteiro. Além disso, eles compartilham e divulgam seu conhecimento sobre testes dentro do time.

Conforme a tecnologia, as metodologias e os processos evoluem e os times e comunidades amadurecem, acredito que é importante termos uma atuação proativa para nos adaptarmos a essas mudanças. O futuro trará novos desafios e oportunidades. Dependendo do contexto, diferentes habilidades e diferentes atividades de teste podem ser necessárias, mas, em minha experiência, existem habilidades básicas que são necessárias para acompanhar o desenvolvimento de software.



Experimente e encontre novas oportunidades

Essas habilidades são:

- ser impaciente para aprender e tentar novos experimentos para aprimorar as estratégias de teste no time.
- ser um excelente questionador durante todo o ciclo de vida do produto. Dependendo do tipo de informação que coletamos, as perguntas podem ser formuladas de maneiras diferentes. Elas podem ser feitas verbalmente ou por escrito, então habilidades de

comunicação clara e bem estruturada são importantes. Elas também podem ser feitas programaticamente; por exemplo, se quisermos verificar certas respostas entre dois serviços, as habilidades técnicas são importantes.

- ter habilidades de modelagem como uma forma de entender o que testar e definir estratégias de teste que cobrem os diferentes aspectos necessários.
- ter algum grau de conhecimento técnico para colaborar na definição dos aspectos de testabilidade da solução enquanto o software está sendo desenvolvido - por exemplo, para dar suporte a testes de unidade e teste de integração automatizados.
- ter uma atitude de compartilhamento e colaboração.

Estamos em um momento empolgante em que podemos moldar parte do nosso futuro. Como você está se preparando para isso?

Seja tudo o que você pode ser

Mike Talks - Nova Zelândia

Nos últimos seis anos, meu time de teste passou de um único grupo trabalhando em um projeto em cascata por vez para indivíduos trabalhando em vários times.

Fala-se muito sobre uma abordagem de time inteiro para qualidade e teste mas os testadores, como especialistas, são considerados para liderar nessa área. Isso significa criar uma abordagem de primeira passagem em uma nova história ou funcionalidade, mas também é importante facilitar uma discussão com o time maior sobre essas abordagens para obter feedback e explorar a abordagem. Também significa que se uma tarefa de teste for muito onerosa para os testadores concluírem sozinhos, eles podem ajudar a organizar uma divisão de trabalho entre os membros dispostos do time

Acho que um candidato frequente para isso é o teste entre navegadores e dispositivos. Frequentemente testamos histórias na iteração usando nossos dispositivos principais, mas ocasionalmente visitaremos nosso produto em uma variedade muito mais ampla de dispositivos. É aqui que o time e seus

novos pares de olhos podem ajudar, e um pouco de organização do testador pode fazer uma grande diferença.

Embora a maioria das conversas sobre o teste de um produto aconteça dentro de um time, também é útil “se atualizar” com outras pessoas nas mesmas disciplinas para compartilhar ideias e o que está funcionando em outros times. Isso também pode se transformar em mentoría para ajudar os indivíduos a lidar com problemas específicos, bem como a se tornarem mais ousados para experimentar novas ideias.



Seja tudo o que você pode

Comece com uma conversa

Kathleen Naughton - Estados Unidos

O teste de software simultaneamente evoluiu e parou no tempo. Ele evoluiu a tal ponto que algumas organizações reduziram o número ou até mesmo eliminaram testadores em seus times. Ele parou no tempo porque essas mesmas organizações tiveram dificuldades para valorizar as habilidades especiais que um testador traz para os times.

Na minha experiência, onde os testadores foram reduzidos ou eliminados, os times tentaram preencher, aproximando-se das práticas de TDD para que tivessem um grande número de testes de unidade automatizados. Eles tentam fazer alguns teste intra-time (testando o código uns dos outros) e dependem fortemente de sua pipeline de entrega contínua para executar seus testes automatizados. O que muitas vezes acaba sendo esquecido são os testes de integração e de experiência do usuário, que são essenciais para produtos de alta qualidade. Se houver testadores nessas organizações, eles estarão presentes para fazer o teste manual após a conclusão do código. Quaisquer insights ou sugestões feitas por esses testadores tendem a ser despriorizados no backlog do produto em relação ao desenvolvimento de funcionalidades.



Seja relevante e influencie aqueles ao seu redor

Acredito que uma habilidade essencial que os testadores precisam para serem relevantes é ser capaz de ler e compreender o código. Isso permite que eles entendam quais testes de unidade ou outros testes automatizados estão verificando e permite a identificação de lacunas de teste que o testador pode preencher. Também permite a redução da sobreposição de teste. Se já houver testes de unidade ou integração presentes, a abordagem de teste pode ser mais direcionada às atividades do usuário final que não são cobertas. Outra habilidade essencial que acredito ser necessária pode ser considerada uma habilidade suave. A habilidade de conversar com programadores sobre seus testes de unidade e integração é poderosa. Essas conversas podem levar a influenciar as decisões de design que, por sua vez, permitem entregas de maior qualidade. Trazer conhecimento sobre como ter conversas cruciais permite que todo o time produza um software melhor.

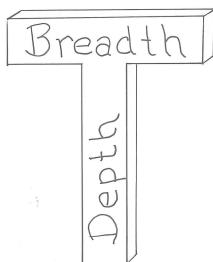
O mundo não precisa de mais validadores

Aldo Rall - Nova Zelândia

Os testes evoluíram com o passar dos anos, e a indústria desenvolveu habilidades e práticas de engenharia de teste, embora não de maneira uniforme. Acho que entramos em uma era de iluminação de testes e há uma grande mudança na forma como as organizações e testadores agora pensam na função de teste.

Observo um movimento crescente em direção à importância das habilidades. Quanto mais habilidades um indivíduo possui, mais valiosas ele se

torna para um time ou organização. Aqueles indivíduos com habilidades que vão além das habilidades de engenharia de testes são aqueles que podem contribuir mais do que alguém que possui um conjunto básico de habilidades de design e execução de teste. Essa ideia é enfatizada pelo pensamento sobre generalização de especialistas conforme discutido por Scott Ambler (<http://www.agilemodeling.com/essays/generalizingSpecialists.htm>) ou habilidades em forma de T, conforme discutido por Lisa e Janet em seus livros. Se você deseja preparar sua carreira para o futuro, desenvolva suas habilidades de engenharia de teste, bem como como habilidades fora do mundo tradicional de testes. Esqueça os títulos; daqui a dez anos não vai significar muito para ninguém se você foi chamado de “engenheiro de teste”, “especialista em teste”, “testador” ou “analista de teste”. Habilidades são, em última análise, mais valiosas do que os títulos de trabalho coletados, eu certamente descobri que isso é verdade em minha própria carreira.



Habilidades em forma de T

Eu olharia para uma abordagem holística caracterizada por conversas “e” inclusivas. O melhor exemplo que posso pensar é combinar sua coleção de habilidades de maneiras únicas que se adaptem ao seu contexto específico, sabendo que mesmo isso mudará. Como você pode combinar um conjunto de habilidades de análise e engenharia de teste em uma determinada situação? Como você pode combinar um conjunto de habilidades de negociação com habilidades de ensino? Como você pode combinar diferentes habilidades de engenharia de teste para obter uma melhor cobertura de teste? Pense em “e”.

Acredito que uma das principais habilidades que um profissional moderno deve ter é a capacidade de compreender um contexto, compreender sua

natureza mutante e se ajustar de acordo. Os verdadeiros mestres do futuro local de trabalho serão aqueles que serão capazes de observar um contexto, aplicar a combinação adequada de habilidades e, em seguida, ajustar continuamente a combinação de habilidades à medida que o contexto evolui. Essa intuição leva tempo para se desenvolver e é um domínio em constante mudança. Aprender novas habilidades enriquecerá a capacidade e o valor que essa pessoa traz para as organizações e times.

Trazemos muitas habilidades diferentes do que apenas habilidades de teste de engenharia. Eu gostaria de sugerir que consideremos (ao tomar emprestado descaradamente de outras pessoas) uma abordagem multifacetada. Chame isso de “Habilidades holísticas de teste ágil” ou “Os dez chapéus do pensamento do teste ágil” ou qualquer outra coisa que faça sentido para você. Algumas dessas facetas podem ser:

- **Consultor:** Sim, às vezes teremos que “prestar consultoria” dentro de nosso time ou com outro time para ajudar a resolver problemas e questões.
- **Especialista em engenharia de teste:** Precisamos saber diferenciar nossos alhos e bugalhos no teste. Nós precisamos ter habilidades de engenharia de teste boas e sólidas.
- **Acadêmico Ágil:** Continue estudando e aprendendo sobre métodos ágeis, traga ideias para o time e experimente.
- **Coach:** Temos grandes oportunidades de treinar o time ou até mesmo colegas de trabalho (dentro e fora do time). Esta é uma habilidade para a vida, em minha opinião.
- **Mentor:** Às vezes temos que ser o mentor de alguém no teste.
- **Facilitador:** Às vezes, só exige assumir o papel de facilitador para uma decisão, discussão, explicação, etc.
- **Agente de mudança:** Às vezes podemos provocar mudanças e revoltas, defender uma nova prática, técnica ou método para experimentar e aprender.
- **Líder:** Sim, às vezes podemos ser obrigados a tomar a iniciativa e desempenhar a liderança em nome do time.
- **Professor:** Isso nem é preciso dizer, especialmente se houver falta de habilidades de teste no time ou organização.

- **Estudioso do domínio de negócios / defensor do senso comum / pensador do panorama geral:** Às vezes é bom se afastar e ver a floresta a partir das árvores.

Pensamentos de Lisa e Janet

Esperamos que você tenha gostado de ler os pensamentos de outras pessoas sobre o que elas consideram ser a função de um testador. Agora vamos compartilhar o nosso. Incentivamos os testadores a ajudar seus colegas de time não testadores a aprender habilidades de teste. Quando todo o time assume a responsabilidade pela qualidade e teste, cada membro do time precisa de alguma base para as habilidades de teste. Para conseguir isso, identificamos habilidades que recomendamos que os especialistas em teste devem aprender.



Use suas habilidades de pensamento

- **habilidades de colaboração** para participar ativamente de práticas como mapeamento mental ou mapeamento de exemplo
- **habilidades de facilitação** para ajudar os membros do time a se comunicarem melhor, facilitar reuniões, assim como retrospectivas ou workshops para ajudar não testadores a aprender habilidades de teste
- **habilidades de ensino** para compartilhar seus conhecimentos com outros membros do time
- **habilidades de coaching** para ajudar o time a identificar problemas e projetar experimentos para melhorias
- **habilidades de comunicação** para dar e receber feedback de forma eficaz (consulte o Capítulo 4, “Thinking Skills For Testing”, em *More Agile Testing* para informações adicionais)

Vemos uma necessidade crescente de testadores atuarem como consultores de teste para seus times. Muitos times têm uma baixa proporção de testadores dedicados para desenvolvedores e outras funções que não sejam de teste. Nós testadores podemos agregar mais valor ajudando todos a terem as competências necessárias para realizar atividades de testes essenciais. Todos no time vão pensar mais sobre os testes e estarão mais conscientes da necessidade de construir qualidade desde o início.

Capítulo 12: Ingredientes Para o Sucesso

Cada time de entrega de software ágil tem a sua própria jornada de aprendizado. Nossa objetivo é melhorar continuamente nossa capacidade de entregar valor aos nossos clientes com frequência, mantendo o padrão de qualidade desejado para o nosso negócio. Cada time está fazendo isso com uma combinação única de domínio de negócios, produto de software, stack de tecnologia, estruturas e práticas.

Ao longo dos anos, descobrimos que, entre todas essas diferenças, certos ingredientes para o sucesso beneficiam todas os times.

Fatores de sucesso

Em nosso primeiro livro, *Agile Testing*, nosso capítulo resumido abrangeu sete fatores de sucesso que consideramos necessários (embora não suficientes) para ter sucesso na entrega de um produto de qualidade. É fácil ficar sobrecarregado planejando e executando atividades de teste durante ciclos de entrega curtos. Abaixo está uma pequena lista dos principais fatores de sucesso e principais práticas de teste ágil para orientar seus times.

“Abordagem do time inteiro”

Elisabeth Hendrickson nos ensinou que “o teste é uma atividade, não uma fase”. O teste é parte integrante do desenvolvimento de software, junto com a programação e muitas outras atividades. Com essa perspectiva, é fácil para todos ajudarem nas tarefas de teste conforme necessário.

Os testadores podem ensinar habilidades a outros membros do time, como obter exemplos concretos de comportamento desejado e indesejado de especialistas de negócios, avaliar diferentes atributos de qualidade ou fazer testes exploratórios.

Os programadores podem ajudar os testadores a entender a arquitetura do sistema para fazer testes melhores ou até mesmo ensiná-los construções básicas de programação. Cada membro do time pode transferir algumas de suas habilidades profundas para outros membros do time, independentemente da função.

Quando os times percebem que o teste e a qualidade são um problema do time, eles podem incorporar seus diversos conjuntos de habilidades e desenvolver uma atmosfera de confiança e segurança, bem como criar um ambiente de aprendizagem onde podem experimentar e melhorar continuamente.

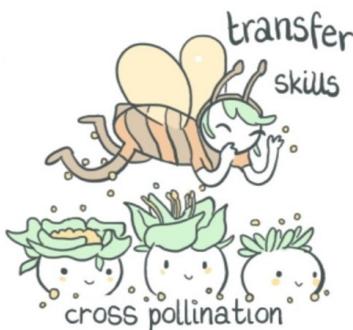


Desenho de Constance Hermit

Mentalidade de teste ágil

Os testadores não são mais a “polícia da qualidade”, determinando as decisões “vai/não vai”. Os testadores ou membros do time que estão realizando atividades de teste podem explicar os riscos e impactos dos resultados dos testes para que a empresa possa tomar uma decisão informada sobre a liberação para produção.

Como um membro do time com uma mentalidade de teste ágil, isso significa que você é curioso e deseja aprender mais sobre tudo para ajudá-lo a fazer seu trabalho. Isso significa que você aplica princípios e valores ágeis. Significa colaborar com os membros do time técnico e de negócios, mantendo o panorama geral em mente ao juntar os pequenos incrementos de funcionalidade. Você está focado na prevenção de bugs, então não precisa perder muito tempo encontrando bugs mais tarde.



Desenho de Constance Hermit

Automatize seus testes de regressão

Há algumas coisas a serem lembradas quando seu time começa a automatizar. É um problema do time, então pense no “time inteiro” e colabore para automatizar em todos os níveis. Os programadores são bons em escrever código, os testadores são bons em especificar testes e pessoas com outras habilidades especializadas no time podem ajudar com dados de teste, infraestrutura e muito mais. A pirâmide de automação de teste é um bom modelo visual para formar e desenvolver a estratégia de automação do time. Ao manter os testes simples e de fácil manutenção, um time pode trabalhar para ter testes de regressão suficientes para dar confiança para o lançamento.

A automação de teste é uma verificação para garantir que você não se esqueceu de mudar algo, ou seja, é um detector de mudanças. Uma boa estratégia de automação dá a você tempo para realizar testes exploratórios para encontrar problemas antes que o seu cliente o faça.

Forneça e obtenha feedback

O desenvolvimento de software bem-sucedido depende de um feedback rápido. Os times precisam saber imediatamente se uma mudança causou uma falha não intencional. Eles querem saber como os clientes reagem a uma nova funcionalidade. Os testadores são fundamentais para criar e

continuar a encurtar os vários ciclos de feedback, incluindo a criação de testes automatizados, participando de testes exploratórios e observando o uso em produção para aprender como os clientes usam o produto.



As pessoas também precisam de feedback para si mesmas, para que possam encontrar mais maneiras de agregar valor. As habilidades de escuta e observação' são fundamentais.

Dica: *Ao colaborar com outros membros do time, pergunte a eles quais lacunas você pode preencher e como pode contribuir de forma mais eficaz.*

Construa uma base de práticas essenciais

Existem práticas essenciais que se mostraram eficazes em ajudar os times a incorporar qualidade em seus produtos.

- Cada time precisa de **integração contínua** para entregar software com sucesso em uma cadência frequente ao longo do tempo. Cada vez que um membro do time faz uma alteração no repositório do código-fonte, ele deve iniciar um processo de build que integra todas as alterações de código e as verifica com testes automatizados. Cada time tem um pipeline de implantação para criar um candidato ao lançamento e implantá-lo em um ambiente de teste ou produção - mesmo que inclua estágios manuais.
- Muitos times ainda lutam para ter **ambientes de teste** confiáveis que se assemelhem ao máximo possível ao de produção e que permitam controlar facilmente qual versão é implantada. A infraestrutura em nuvem de hoje oferece ainda mais opções para criar

ambientes de teste temporários para testar uma versão específica e implantar automaticamente novas versões em ambientes de teste permanentes.

- A dívida técnica é como a dívida do cartão de crédito; continua a crescer se apenas o valor mínimo ou parte dos juros forem pagos. A dívida técnica continua a crescer se o time não reservar tempo para refatorar o código, criar testes de regressão automatizados adequados, atualizar estruturas e gerenciar outras infraestruturas necessárias. Com o tempo, até mesmo a menor alteração no código apresenta grandes riscos e requer muito tempo gasto com tarefas de teste manuais. Invista tempo para **gerenciar dívidas técnicas** no código e em seus testes automatizados.



- Alterações pequenas e frequentes geralmente são menos arriscadas do que alterações grandes e infreqüentes. Menos coisas podem dar errado e as falhas podem ser diagnosticadas rapidamente. Os times que dividem grandes ideias de funcionalidades em pequenos “lançamentos de aprendizagem” e histórias ainda menores têm mais probabilidade de entregar o que seus clientes desejam em tempo hábil.
- **Programação e teste são parte de um processo.** Este é o cerne da abordagem do time inteiro para teste e qualidade. O teste e a programação acontecem juntos, de mãos dadas, desde a ideia da funcionalidade até a avaliação da funcionalidade em produção.
- **Sinergia entre as práticas** vem de fazer todas essas práticas principais juntas. O desenvolvimento guiado por testes, a propriedade coletiva do código e a integração contínua garantem consistência e feedback rápido. A refatoração depende de testes de regressão automatizados. As práticas ágeis são testadas e comprovadas e são projetadas para serem realizadas em conjunto.

Colabore com os clientes

Os testadores falam a linguagem de domínio das partes interessadas do negócio e a linguagem técnica dos membros do time de entrega. Também é importante reunir as pessoas certas quando uma conversa é necessária sobre como uma funcionalidade deve se comportar ou como deve ser a aparência de um design. Os testadores podem ajudar os product owners a articular regras de negócios para cada história e ilustrá-las com exemplos concretos. Esta é uma das formas que tem mais valor pelas quais os testadores contribuem nos times.

Olhe para o panorama geral

Embora os times ágeis se concentrem em pequenas mudanças e pequenas fatias de funcionalidade a qualquer momento, eles também precisam manter o panorama geral em mente. Os testadores são talentosos em identificar quais partes do sistema podem ser afetadas por uma pequena mudança em particular, e eles têm uma perspectiva do cliente.

O modelo de quadrantes de teste ágil ajuda muito o time a manter o panorama geral em mente ao planejar as atividades de teste. O teste exploratório é um exemplo de atividade de teste que pode descobrir consequências inesperadas de uma nova funcionalidade da aplicação. Temos boas maneiras de analisar como os clientes estão usando nosso produto. Tudo isso nos ajuda a nos concentrar em entregar o valor certo.

Práticas de criação de confiança

Em nosso segundo livro, *More Agile Testing*, identificamos algumas práticas fundamentais de teste que ajudam os times a criar a confiança necessária para liberar frequentemente mudanças para produção. Essas práticas são especialmente importantes à medida que os times se movem em direção à entrega contínua ou implantação contínua.

Use exemplos

Exemplos concretos de como uma funcionalidade da aplicação deve se comportar ajudam todos no time a entender as regras de negócios. Esses

exemplos podem ser transformados em testes que guiam o desenvolvimento. Eles podem ser automatizados para que o time saiba quando terminar com uma história ou funcionalidade. Os testes automatizados se tornam parte dos conjuntos de testes de regressão que fornecem feedback rápido se uma nova mudança afetou o comportamento de produção existente. Os exemplos ajudam os times a se manterem no caminho certo.



Teste exploratório

Testes de regressão automatizados deixam mais tempo para testes exploratórios, uma das melhores maneiras de encontrar as “incógnitas desconhecidas” que podem causar falhas terríveis em produção. Os programadores podem aprender a fazer testes exploratórios em cada história antes de considerarem seu trabalho “concluído”. Este é outro loop de feedback rápido. Todos no time podem aprender e usar habilidades de teste exploratório. Eles não apenas identificarão problemas inesperados, mas também encontrarão funcionalidades ausentes que alimentam novas ideias de funcionalidades.

Teste de funcionalidades

É essencial testar em todos os níveis de detalhe. Como os times ágeis se concentram na história, eles precisam se lembrar de testar também no nível da funcionalidade. Uma parte importante disso é identificar o que a funcionalidade realmente precisa incluir. Os testadores podem ajudar a descobrir o que tem valor para os clientes, fazendo perguntas sobre o que a empresa deve abandonar em favor de fornecer outras funcionalidades que tem mais valor.

Aprendizagem continuada

O sucesso do time depende da segurança psicológica, confiança e tempo para aprender. O time precisa trabalhar em conjunto para identificar o maior obstáculo para entregar o nível de qualidade desejado, seja automação de teste inadequada, feedback que leva muito tempo ou criação de funcionalidades que ninguém queria. Então, eles podem criar pequenos experimentos para começar a superar esse obstáculo. Os testadores podem ajudar o time a aprender a aumentar a qualidade transferindo habilidades de teste. Outros membros do time podem ajudar os testadores a aprimorar suas habilidades em forma de T para que possam contribuir de mais maneiras.



Sensibilidade ao contexto

Cada time está trabalhando em seu contexto único. O tamanho da empresa, o domínio de negócios e seu ambiente regulatório, a tecnologia envolvida, as necessidades de infraestrutura - essas são apenas algumas considerações para um time que considera como melhorar sua capacidade de entregar valor aos clientes com frequência. Não adote uma ferramenta ou prática porque é o que o Google ou o Facebook fazem - use o que for apropriado para o seu contexto.



Mandando a real

Os testadores são excelentes em fornecer feedback. Pode ser difícil dar más notícias. Mas é importante permanecer aterrado na realidade. Se uma mudança é arriscada e o time não mitigou adequadamente esse risco com testes e outras atividades, as partes interessadas do negócio precisam saber. Os testadores podem atuar como consultores para ajudar todos em seu time a melhorar suas habilidades de teste e tornar as questões de qualidade visíveis para a empresa. Quando o time tiver gargalos com os testes, torne-os visíveis, torne-os um problema do time para resolver. Pode ser tentador encobrir os problemas para manter os executivos do negócio felizes, mas eles não ficarão felizes se os clientes sentirem dor.

Também é importante para a moral do time saber que eles podem dizer não. Por exemplo, “Não, não podemos aceitar mais histórias” ou “Não, não podemos adicionar uma nova história a menos que você remova uma das outras”. Mantenha a real!

Caminhos para o sucesso

Sabemos que há muitos testadores e times por aí que se sentem estressados, especialmente em times que ainda estão em transição para o uso de valores, princípios e práticas de desenvolvimento ágil. Por exemplo, a gerência ainda não descobriu como sua função precisa mudar e pode exigir entregas mais frequentes e impor prazos não realistas. O teste ainda deve ser feito e, em muitos casos, os testadores ainda têm total responsabilidade por todas as atividades de teste. Gostaríamos de pensar que existe alguma ferramenta mágica que vai resolver todos os nossos problemas, mas sabemos que isso não é a realidade!

Transformar problemas de teste em problemas para todo o time de entrega resolver é vital para aprender como construir qualidade em seus produtos e alcançar o sucesso sustentável. Esses fatores-chave de sucesso e práticas de criação de confiança fornecem uma estrutura para ajudar o time a decidir suas próximas etapas em um caminho de melhoria.

Em nossa experiência, leva anos para um time de entrega atingir o nível desejado de desempenho e qualidade. Podemos adicionar um oitavo

fator-chave de sucesso: paciência! Retrospectivas frequentes no time (recomendamos pelo menos uma por semana para times novos) também são fundamentais para identificar o maior problema relacionado à qualidade e projetar um pequeno experimento para começar a eliminá-lo. As diversas habilidades e experiências em um time multifuncional tornam a solução desses problemas muito mais fácil.

Um exemplo

O time está frustrado porque o product owner rejeita uma porcentagem alta das histórias que eles entregam. O constante retrabalho, às vezes dias depois que o time pensou que a história estava “acabada”, está os atrasando. O tempo de ciclo - o tempo desde quando eles começam a trabalhar em uma história até quando ela é implantada em produção - é muito mais longo do que eles gostariam. Que fator chave de sucesso pode ajudar?

A abordagem do time inteiro é óbvia. Vamos reunir todo o time, ou um grupo representativo, incluindo todas as funções, para discutir o assunto. Que prática de criação de confiança ajudaria? Quando o product owner rejeita uma história, geralmente é porque o comportamento daquela parte da aplicação não é o que ele queria. O time entendeu mal os requisitos. O time está aprendendo continuamente (uma das práticas de criação de confiança), e um dos testadores acabou de aprender sobre o mapeamento de exemplo. Eles decidem experimentar o mapeamento de exemplo para ver se ele vai criar uma entendimento compartilhado melhor de cada história. Eles levantam a hipótese de que o mapeamento de exemplos reduzirá a taxa de rejeição de histórias em 20% nas próximas duas semanas, resultando em uma economia de 10% do tempo médio de ciclo.

Eles medem e experimentam para ver se sua hipótese é verdadeira. Neste exemplo real, o experimento foi um sucesso. As metas de redução da taxa de rejeição e do tempo de ciclo foram superadas. Em dois meses, a taxa de rejeição e o tempo de ciclo foram reduzidos em 50%. O time encontrou mais benefícios com o mapeamento de exemplo, pois ajudou a especificar cenários para guiar o desenvolvimento na forma de testes de BDD. Mas se o experimento tivesse falhado, o time teria projetado outro experimento, guiada pelos fatores de sucesso e práticas de criação de confiança.

Quando nossos próprios times se sentem presas a um problema, recorremos aos fatores-chave de sucesso e às práticas de criação de confiança, junto com os dez princípios para teste ágil do Capítulo 1, para nos ajudar a planejar nossas próximas etapas. Eles nos guiarão ao longo da nossa jornada de aprendizado à medida que melhorarmos nossa capacidade de entregar mudanças pequenas e de valor para nossos clientes com freqüência e sustentabilidade.



Obrigado por ler e esperamos que você tenha sucesso em sua própria jornada.

Glossário

Teste ad-hoc: Uma atividade de teste informal onde se procura bugs de forma não estruturada e sem nenhum planejamento prévio.

Diagrama de Contexto: Um diagrama de alto nível que representa todas as entidades que podem interagir com um sistema, incluindo outros sistemas, ambientes e atividades.

Cliente: A Programação Extrema (XP - Extreme Programming) usa o termo “cliente” para se referir a uma parte interessada do negócio, pessoa de produto ou usuário final que se reúne com o time de programação para definir prioridades, responder perguntas e tomar decisões sobre o comportamento das funcionalidades. Em times ágeis modernos, o termo pode representar todas as partes interessadas de negócio, membros do time de produto, usuários finais e qualquer pessoa que ajude a orientar o desenvolvimento e aceitar histórias entregues.

Final do Jogo: O final do jogo é o tempo antes do lançamento, quando o time de desenvolvimento aplica os toques finais no produto. Não é período de uma correção de bugs ou de “fortalecimento”, é uma oportunidade de trabalhar com grupos fora do desenvolvimento para ajudar a colocar o software em produção. Exemplos de atividades de final de jogo incluem testes adicionais de migração de banco de dados e instalação.

Iteração: Tempo limitado utilizado para planejamento, com a intenção de que haja um “produto potencialmente entregável” no final. O termo no Scrum para isso é “sprint”. Planejar em prazos de duas semanas é uma prática comum hoje, mesmo em times que fazem entrega contínua e implantam em produção com mais frequência.

Kanban: Uma abordagem de planejamento derivada da manufatura enxuta, na qual os times trabalham de maneira baseada em fluxo. São utilizados limites de trabalho em andamento (WIP - work in progress), puxando novas histórias que estão “prontas” para preencher um slot WIP recém vazio. O time planeja, conforme necessário, algumas novas histórias de cada vez.

Lançamento de Aprendizado: As primeiras versões entregues ao cliente para receber feedback para aprender e ajustar (https://medium.com/@Ardita_K/the-learning-release-70374d2450b3).

Mapa Mental: Um diagrama visual utilizado como ferramenta de brainstorming, especialmente quando várias pessoas estão colaborando ao mesmo tempo. Ele começa com um conceito, ideia ou tópico no nó raiz, com ideias conectadas ao nó raiz e umas às outras conforme são geradas. Mapas mentais podem funcionar bem para o planejamento de testes e outras atividades.

Pareamento (programação em pares, testes em pares): Duas pessoas trabalhando lado a lado na mesma estação de trabalho, de preferência com dois monitores espelhados, dois teclados e dois mouses, para escrever código de produção ou de teste ou fazer outras atividades de teste. Ter duas pessoas, cada uma com uma perspectiva e conjunto de habilidades diferentes, ajuda a detectar problemas imediatamente e chegar a melhores soluções. No pareamento de estilo forte, uma pessoa, o navegador, é livre para observar e sugerir ideias, enquanto a outra pessoa atua como o “motorista”; as funções do motorista/navegador mudam com frequência.

Diagrama de Estados: Uma técnica visual utilizada para dar uma descrição abstrata do **comportamento²²** do **sistema²³** em resposta a vários eventos. Esse comportamento é analisado e representado como uma série de eventos que podem ocorrer em um ou mais estados possíveis.

Desenvolvimento Guiado por Testes (TDD): No desenvolvimento guiado por testes o programador escreve e automatiza um pequeno teste de unidade, que inicialmente falha, antes de escrever a quantidade mínima de código que fará o teste passar. O código é refatorado conforme necessário para atender aos padrões aceitáveis. O código de produção é feito funcional um teste de cada vez. TDD, também conhecido como Design Guiado por Testes (Test Driven Design), é mais uma prática de design de código do que uma atividade de teste e ajuda a construir um código robusto e de fácil manutenção.

²²<https://en.wikipedia.org/wiki/Behavior>

²³<https://en.wikipedia.org/wiki/System>

Recursos Para Aprender Mais

Geral

Agile Testing: A Practical Guide for Testers and Agile Teams, e *More Agile Testing: Learning Journeys for the Whole Team*, Lisa Crispin e Janet Gregory, <https://agiletester.ca>²⁴

Bibliografia do livro *More Agile Testing*, digitalizada por Kristine Corbus

- <https://testretreat.com/2018/01/28/more-agile-testing-introduction/>
- <https://testretreat.com/2018/01/29/more-agile-testing-learning-better-testing/>
- <https://testretreat.com/2018/01/29/more-agile-testing-planning/>
- <https://testretreat.com/2018/01/30/more-agile-testing-business-value>
- <https://testretreat.com/2018/02/15/more-agile-testing-test-automation/>

Desenvolvendo um Entendimento Compartilhado - Colaboração

Discovery: Explore behavior using examples, Seb Rose e Gáspár Nagy, 2017, <http://bddbooks.com/>

User Story Mapping: Building Better Products Using Agile Software Design, Jeff Patton, O'Reilly Media, 2014.

Impact Mapping, Gojko Adzic, <http://impactmapping.org>

“Experiment with Example Mapping”, <https://lisacrispin.com/2016/06/02/experiment-example-mapping/>

²⁴[https://agiletester.ca/](https://agiletester.ca)

“Introduction to Example Mapping”, Matt Wynne, <https://cucumber.io/blog/example-mapping-introduction/>

“Three Amigos Strategy”, George Dinwiddie, <https://www.agileconnection.com/article/three-amigos-strategy-developing-user-stories>

“Our team’s first mobbing session”, Lisi Hocke, <https://www.lisihocke.com/2017/04/our-teams-first-mobbing-session.html>

“The Driver-Navigator in Strong-Style Pairing”, Maaret Pyhäjärvi, <https://medium.com/@maaret.pyhajarvi/the-driver-navigator-in-strong-style-pairing-2df0ecb4f657>

Strong-Style Pair Programming e Mob Programming Guidebook, Maaret Pyhäjärvi, <https://leanpub.com/u/maaretp>

Testes Exploratórios

Explore It: Reduce Risk and Increase Confidence with Exploratory Testing, Elisabeth Hendrickson, 2013, <https://pragprog.com/book/ehxta/explore-it>

Exploratory Testing, Maaret Pyhäjärvi, <https://leanpub.com/exploratorytesting>

DevOps, Monitoramento, Observabilidade

A Practical Guide to Testing in DevOps, Katrina Clokie, <https://leanpub.com/testingindevops>

“Testing in production the safe way”, Cindy Sridharan, <https://medium.com/@copyconstruct/testing-in-production-the-safe-way-18ca102d0ef1>

“Monitoring and observability”, Cindy Sridharan, <https://medium.com/@copyconstruct/monitoring-and-observability-8417d1952e1c>

“Charity Majors on Observability and Understanding the Operational Ramifications of a System”, entrevista com Charity Majors para a InfoQ, <https://www.infoq.com/articles/charity-majors-observability-failure>

Google Site Reliability Engineering, <https://landing.google.com/sre/>

“What is Chaos Engineering²⁵?” Joe Colontonio (*incluir link para o podcast com Tammy Butow*), <https://www.joecolantonio.com/chaos-engineering/>

“Tracing vs Logging vs Monitoring: What’s the Difference?” Chrissy Kidd, <https://www.bmc.com/blogs/monitoring-logging-tracing/>

Automação de Testes

“Keep your automated tests simple and avoid anti-patterns”, Lisa Crispin, <https://www.mabl.com/blog/keep-your-automated-testing-simple>

“Test automation: Five questions leading to five heuristics”, Joep Shuurkes, <https://testingcurve.wordpress.com/2015/03/24/test-automation-five-questions-leading-to-five-heuristics/>

“Powerful test automation practices”, partes 1 e 2, Lisa Crispin e Steve Vance, <https://www.mabl.com/blog/powerful-test-automation-practices-pt-1>, <https://www.mabl.com/blog/powerful-test-automation-practices-pt-2>

“Test Suite Design”, Ashley Hunsberger, <https://github.com/ahunsberger/testSuiteDesign>

Accelerate: The Science of Lean and DevOps, Nicole Forsgren, et al, <https://itrevolution.com/book/accelerate/>

“Analyzing automated test failures”, Lisa Crispin, <https://www.mabl.com/blog/lisa-webinar-analyzing-automated-ui-test-failures>

“The Testing Iceberg”, Seb Rose, <http://claysnow.co.uk/the-testing-iceberg/>

“Lower level automation and testing? Be more precise! The automation triangle revisited again!” Toyer Mamoojee, <https://toyerm.wordpress.com/2018/10/16/lower-level-automation-and-testing-be-more-precise-the-automation-triangle-revisited-again>

The Team Guide to Software Testability, Ash Winter e Rob Meaney, <https://leanpub.com/softwaretestability>

²⁵<https://www.joecolantonio.com/chaos-engineering/>

Sobre as Autoras

Janet Gregory é coach de testes ágeis e consultora de processos da DragonFire Inc. Seus colegas a elegeram como a Pessoa Mais Influente em Testes Ágeis em 2015. Janet é especialista em mostrar aos times ágeis como as práticas de teste são necessárias para desenvolver produtos de boa qualidade. Ela dá cursos de teste ágil em todo o mundo e gosta de passar os verões nas montanhas fora de Calgary.

Lisa Crispin tem espalhado a alegria ágil para o mundo dos testes e a alegria dos testes para o mundo ágil há duas décadas e seus colegas a elegeram como a Pessoa Mais Influente em Testes Ágeis de 2012. Seus interesses atuais incluem ajudar os times a ter sucesso com entrega contínua e implantação, e ela é uma defensora de testes que trabalha na mabl para explorar as principais práticas de teste na comunidade de software. Lisa mora com o marido, burros, cães e gatos na bela Vermont.

Janet e Lisa são autoras de *Agile Testing Condensed: A Brief Introduction* (2019), *More Agile Testing: Learning Journeys for the Whole Team* (2014), *Agile Testing: A Practical Guide for Testers and Agile Teams* (2009), o curso de vídeo LiveLessons Agile Testing Essentials e o curso de treinamento de 3 dias “Agile Testing for the Whole Team” oferecido pela Agile Testing Fellowship (Irmandade de Testes Ágeis). Juntas, elas fundaram a irmandade para desenvolver uma comunidade de profissionais que se preocupam com a qualidade.

Janet Gregory: janetgregory.ca²⁶ Twitter: @janetgregoryca

Lisa Crispin: lisacrispin.com²⁷ Twitter: @lisacrispin

Agile Testing Fellowship:
agiletestingfellow.com²⁸ agiletester.ca²⁹

²⁶<https://janetgregory.ca/>

²⁷<https://lisacrispin.com/>

²⁸<https://agiletestingfellow.com/>

²⁹<https://agiletester.ca/>

Sobre os Tradutores

Felipe Knorr Kuhn se dedica às áreas de qualidade e agilidade desde 2005, buscando ver o teste sob diferentes perspectivas e como ele pode ser aplicado nos mais variados contextos. Sua carreira envolve projetos em startups e grandes empresas no Brasil e no Vale do Silício, nas áreas de logística, finanças, sistemas governamentais, hardware, mobile, entre outras. Participa das comunidades brasileira e internacional para sempre se manter atualizado e dividir o que aprende com os desafios da área. Atualmente mora nos Estados Unidos e trabalha no time de TV da Netflix.

Paulo Gonçalves possui grande paixão em software Open Source e trocas de conhecimento com a comunidade de qualidade brasileira, realizando lives, palestras e mantendo ferramentas que auxiliam nos estudos sobre automação de testes, como o ServeRest. Também acredita que é importante que todo analista de testes tenha conhecimento técnico e seja, resumidamente, um agile tester. É mineiro e deficiente auditivo.

Felipe Knorr Kuhn e **Paulo Gonçalves** fazem parte do [coletivo Agile Testers³⁰](#), que mantém vários canais de compartilhamento de conhecimento sobre qualidade de software, como o podcast [QAnsei³¹](#) e a revista [AssertQA³²](#).

Felipe Knorr Kuhn: [Site³³](#) Twitter: [@knorrium³⁴](#) LinkedIn: [/knorrium³⁵](#)

Paulo Gonçalves: Github: [/PauloGoncalvesBH³⁶](#) Twitter: [@paulorgoncalves³⁷](#) LinkedIn [/paulo-goncalves³⁸](#)

³⁰<https://agiletesters.com.br/>

³¹https://anchor.fm/qansei?utm_source=livro-agiletesting-condensed

³²https://medium.com/assertqualityassurance?utm_source=livro-agiletesting-condensed

³³<https://knorrium.info>

³⁴<https://twitter.com/knorrium>

³⁵<https://www.linkedin.com/in/knorrium>

³⁶<https://github.com/PauloGoncalvesBH>

³⁷<https://twitter.com/paulorgoncalves>

³⁸<https://www.linkedin.com/in/paulo-goncalves>